EMET 4.1 Uncovered

@0xdabbad00 (Dabbadoo)

0xdabbad00.com

2013-11-18

Introduction

EMET (Enhanced Mitigation Experience Toolkit) is a free application available from Microsoft. It adds security mitigations, primarily to thwart memory corruptions exploits. This document seeks to better aggregate the information on these mitigations and better describe how these protections work than is currently available. This paper focuses specifically on the limitations and defeats for these protections so that EMET can be better understood and improved.

In addition to this paper it is recommended you read the following:

- EMET 4.1 Users Guide.pdf[1] [22]: At 42 pages this describes many of the protections in-depth.
- Inside EMET 4.0[2] [3]: Presentation from REcon 2013 by Elias Bachaalany from Microsoft.
- Runtime Prevention of Return-Oriented Programming Attacks[3] [10] by Ivan Fratric: One of the winners of the 2012 BlueHat Prize for ROP mitigations. Some of these mitigations were added to EMET 3.5.

Limitations of this paper

I did not investigate the ROPGuard additions thoroughly. The main reason for this is these are largely copied from the open-source ROPGuard[4] [9] project. Also this paper is already long enough and these are harder to reverse engineer. ;) These ROPGuard protections are LoadLib, MemProt, Caller, SimExecFlow, and StackPivot. I hope to follow-up on this paper with additional work which may investigate these more thoroughly.

The additional knowledge for this paper was obtained through static reverse engineering and some windbg analysis. I have not yet tested my assumed defeats.

This paper is mostly interested in the benefits provided for modern software on Windows 8.1. This contrasts with one of the goals of EMET which is to backport security features to older software and versions of Windows.

Download

Download EMET 4.1.5064.16886 from http://www.microsoft.com/en-us/download/details.aspx?id=41138
MD5 hash for `EMET Setup.msi`: `59e2da5e2b6633f8422be54bdb5ecf3e`

History

EMET 1.0.2 (2009-10-27)

The initial EMET 1.0.2 release [17] was command-line based, and provided the following mitigations:

- SEHOP (system and per process)
- DEP (system and per process)
- Null page allocation
- Heap spray allocation

EMET 2.0 (2010-09-02)

The EMET 2.0 release [26] introduced the GUI and added mitigations for:

- System ASLR and per-process MandatoryASLR
- Export Address Table Filtering (EAF)

[1] http://www.microsoft.com/en-us/download/details.aspx?id=41138
[2] http://recon.cx/2013/slides/Recon2013-Elias%20Bachaalany-Inside%20EMET%204.pdf
[3] https://ropguard.googlecode.com/svn/trunk/doc/ropguard.pdf
[4] https://code.google.com/p/ropguard/source/browse/

Minor bug fix 2.0.0.3 also released [28].

EMET 3.0 (2012-05-15)

The EMET 3.0 release [24] was capable of enterprise wide deployment with protection profiles able to be imported and exported. Writes to event log and shows tooltip when mitigations catch exploits. No mitigations added.

EMET 3.5 (2012-07-25)

The EMET 3.5 release [19] applied ROP mitigations from ROPGuard, one of the BlueHat prize winners. These mitigations are:

- Caller check
- Execution flow simulation
- Stack pivot
- Special function checks for Memory Protections and LoadLibrary

This release also added the protection Bottom Up ASLR

EMET 4.0 (2013-04-18)

The EMET 4.0 release [20] added an audit mode and new mitigations:

- Deep Hooks
- Anti-detours
- Banned Functions
- Certificate pinning

EMET 4.1 (2013-11-12)

The EMET 4.1 release [25] was a very minor update. The relevant changes for this paper were the default protection profiles had some features disabled for certain applications for FP (False Positive) issues, and the number of certificates trusted for the certificate pinning was expanded.

General notes

Basics of how it works

EMET is installed at `C:\Program Files (x86)\EMET 4.1\` The per process protections are accomplished by getting `EMET.dll`, or `EMET64.dll` for 64-bit processes, to run within the processes to be protected. This means those protections will be solely user-land. The system wide protections are accomplished by registry tweaks.

The DLL's are loaded via Application Compatibility [2, 18]. Because of this, EMET can not be loaded into and protect every new process on the system automatically, like most defensive software. You have to use EMET to configure the Application Compatibility database to load the EMET DLL based on part of the process path. This usability issue often means that even users who have installed and taken steps to configure EMET still end up with processes that are not protected.

EMET configuration information is stored in the registry key `HKLM\Software\Microsoft\EMET\`.

On the GUI screen, EMET knows which processes it is protecting because each time the EMET DLL is loaded into a process it creates a global event named "EMET_PID_1234" where "1234" is the process ID.

When EMET is protecting a process, it will create an environmental variable called `EMET_Settings` which contains the settings for that process. I suspect this is used for crash dump analysis. As an example:

```
EMET_Settings={Nov  8 2013 13:30:09};ReportingSettings:7;ExploitAction:1;DEP:1;SEHOP:2;
NullPage:1;HeapSpray:1;EAF:1;MandatoryASLR:1;BottomUpASLR:1;StackPivot:1;Caller:1;
LoadLib:1;MemProt:1;DeepHooks:1;BannedFunctions:1;AntiDetours:1;SimExecFlow:1=15
```

Default protecteded processes

By default, EMET will protect processes according to the policy defined at `C:\Program Files (x86)\EMET 4.0\Deployment\Protection Profiles`. The default protected processes are:

- Internet Explorer
- Office products
- Adobe Reader and Acrobat
- Java

There is an additional profile for Popular Software that builds on this, with some mitigations disabled by default. The additional software is:

- Windows Media Player (no Mandatory ASLR or EAF)
- Skype (no EAF)
- Microsoft Lync Communicator
- Microsoft Live Essentials
- Google Chrome (SEHOP only for Windows 7 and up)
- Google Talk (SEHOP only for Windows 7 and up[1], no DEP)
- Mozilla Firefox and Thunderbird
- Adobe Photoshop
- Nullsoft Winamp
- Opera browser
- WinRAR
- WinZip
- VLC
- RealPlayer
- mIRC
- 7-zip (no EAF[1]).
- Apple Safari
- Apple Quicktime
- Apple iTunes (no Caller[1])
- Pidgin
- Foxit Reader

[1] The denial of certain protections is in the .xml file, but this conflicts with what the User Guide says that shows all protections are enabled for these processes. The truth is in the XML file.

These protections are based on the file location (with some regexing). This means if Java 8 is installed in a folder called `jre8` (Java 7 is installed in `jre7`), it will not be protected.

System Wide Mitigations

Data Execution Prevention(DEP)

This calls `bcedit.exe /set nx AlwaysOn` and on Windows 8 it additionally sets the MitigationOptions registry value (see the Appendix). If you run Windows XP or 2003, it will call `bootcfg.exe`.

What it protects against

DEP has been exhaustively covered elsewhere (for example, on my site[5] [1]).

Modern software should all be compiled with DEP enabled by default. There is a flag in the `DLLCharacteristics` of the PE header which tells Windows to DEP protect the process. If that flag is not enabled Windows will not protect that process unless the steps EMET performs have been taken to force the protection.

When that flag is not enabled it often means the software was compiled with an old compiler. A lot of software, even software that has been compiled recently, still uses out-dated compilors or build configurations. For example putty, 7-zip, cygwin, and many other applications are not compiled with DEP support, which implies they were not compiled with modern compilers.

Weaknesses

DEP mostly forces memory exploits to use ROP, which is the focus of most of the additions to EMET 3.5 and up. The ROP chain is normally used to first disable DEP (via `SetProcessDEPPolicy()`), to set memory as executable (via `VirtualProtectEx()`), or to download and give execution to a binary that performs the remainder of the attack. All of these actions are done to avoid DEP. Interestingly, although the latter options are protected in some way by EMET, the first option (calling `SetProcessDEPPolicy()`) is not.

Although a process might be configured to use DEP, the process can disable DEP on itself manually. As an example, the recent Office TIFF exploit CVE-2013-3906[6] [36] as explained here[7] [16] loaded a DLL named VBE6.dll which called `ntdll!ZwSetInformationProcess` to disable DEP on the process it was loaded into, unless EMET forced DEP using the "AlwaysOn" option.

Impact

64-bit software is already forced to use DEP, so this is only useful for 32-bit software that was compiled with old compilors. Although there are other ways to enable DEP enforcement, I would argue that this ability of EMET is useful enough to make EMET a required install.

Structured Exception Handling Overwrite Protection (SEHOP)

The best explanation of SEHOP is the article by Matt Miller (skape) on Uninformed [33] and the implementation in his open-source project WehnTrust [11] which worked on Windows 2000, XP and 2003. SEHOP as applied by EMET does not work on those OS's, but instead works on Vista and above.

In the attacks that SEHOP seeks to prevent, the exception handler pointer may be over-written to point to shellcode. This breaks the chain of exception handlers. SEHOP will ensure the chain is unbroken before passing execution to the exception handler/shellcode.

It works on Windows 8 by setting the registry MitigationOptions value. On Windows Vista and 7, it works by setting the following to `0` as explained here[8] [23]:

`HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\kernel\DisableExceptionChainValidation`

[5] http://0xdabbad00.com/2012/12/07/dep-data-execution-prevention-explanation/
[6] http://blogs.technet.com/b/srd/archive/2013/11/05/cve-2013-3906-a-graphics-vulnerability-exploited-through-word-documents.aspx
[7] http://blogs.mcafee.com/mcafee-labs/solving-the-mystery-of-the-office-zero-day-exploit-and-dep
[8] http://support.microsoft.com/kb/956607

What it protects against

This protects against SEH overwrites, which are stack based buffer overflows that would obtain execution from the SEH pointer, as opposed to the return pointer. SEH overwrites are often preferred to stack overwrites, because they require a `jmp esp` instruction sequence in memory, as opposed to a `pop/pop/ret` which is less common, and also they can be more reliable because they work when the program causes an exception, so it's not only ok for the attack to cause an exception but is required.

Weaknesses

As proven by Aaron Portnoy in his recent presentation "Bypassing All Of The Things" [30] on slide 58, if you can read the process memory in order to know the value of the next handler in the chain, then you can corrupt the SEH chain to call your handler when the exception occurs, while still maintaining the integrity of the SEH chain.

This weakness was pointed out by Matt Miller when he initially proposed the idea, in his Uninformed article in the section on Design, he acknowledges the limitations when he asks "what's to stop an attacker from simply overwriting the Next pointer with the value that was already there. [...] First of all, it will be common that the attacker does not know the value of the Next pointer." At the time, ASLR did not exist for Windows (Mr. Miller also implemented that in WehnTrust), which meant attackers were not as concerned then with info leaks. As info leaks have gained in importance, his assumptions no longer hold true. This is evidenced by Ivan Fratric's recent statement that "the two important elements that constitute a modern browser exploit: The ability to read arbitrary memory and to gain control over RIP." [8]

Impact

SEHOP is unnecessary for 64-bit process (see this[9] [29] article for why).

SEHOP is enabled by default on recent server editions of Windows (2008 and 2012), but not the consumer versions (Vista, 7, and 8). Further this protection is only important when all the binaries used by application were not compiled with /SafeSEH which compiles into the binary what exception handlers are allowed.

Due to the spread of ASLR, attacks more commonly need to use info leaks which can then allow for them to more easily bypass SEHOP. So although SEHOP is great, it could be argued that it no longer plays as important a role as it once did.

Address Space Layout Randomization (ASLR)

The ASLR setting can be either "Disabled" or "Opt-in", unless you enable unsafe settings by manually setting the registry value `HKLM\SOFTWARE\Microsoft\EMET\EnableUnsafeSettings` to `1`. This allows for the "AlwaysOn" setting, and on Windows 8 it additionally allows for an "Opt-out" option.

On Windows 8, when this protection is enabled it sets the `MANDATORYASLR` bits of the MitigationOptions value. On previous OS's (Vista and up) it sets `HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\MoveImages`.

What it protects against

Once a process has DEP enabled, an attacker needs to use ROP to bypass it. To use ROP, they need know the locations of data in memory, which ASLR seeks to prevent.

[9] https://www.osronline.com/article.cfm?article=469

Weaknesses

A critical weakness to understand about ASLR is that for it to do anything to the image of the executable file (DLL or EXE) loaded into memory, that executable file must have been built with relocations so that it can be relocated to a different address. Often times old executables, or those that do not use modern compilers, were not built with this.

As an example, the most recent version of `putty.exe` (version 0.63 released on 2013-08-06) was compiled without relocation information. Because of this, the `putty.exe` image will always load at the address `0x00400000` in memory. You can turn every EMET setting up to the maximum protection, and the same data will always be at `0x00400000` in memory. This problem is not specific to putty, but is provided merely as an example. An image needs to have a `.reloc` section. The bit flag `DllCharacteristics` in the PE header controls whether this image will use ASLR by default, but without the `.reloc` section, it will be ineffective.

ASLR can be weakened accidentally when developers call `VirtualAlloc` with specific addresses.

Impact

Windows Vista and up already uses ASLR for any executables that were compiled to be compatible with it, so the setting EMET changes is only enforcing ASLR for executables that were not compiled to be compatible with ASLR, but were compiled with a relocations section, which I assume is rare. This therefore will have little impact.

Certificating Trust (Pinning)

Although this is listed as a System Wide setting, it actually only protects Internet Explorer. This can be configured to protect more processes by manually setting the registry values (see User Guide section 4.1.1.3 [25]).

The concept behind this is to ensure that the certificate chain for an SSL certificate for a website leads back to an expected root Certificate Authority. This feature was not analyzed, because no one I know uses Internet Explorer, and Chrome and Firefox already implement this or can do something similar through plugins (ex. http://convergence.io/). This provides the most benefit for those using localized browsers (Yandex, Tencent, etc.) or Opera. More information about this feature is available from Microsoft here [21].

I do know this works using the DLL `EMET_CE.dll` and `EMET_CE64.dll`, and this DLL exports the single function `extCVCCP` which is referring to the Windows function `CertVerifyCertificateChainPolicy`. Then, according to Neil Sikka, who is a developer for EMET and presented a talk titled "EMET 4.0 PKI Mitigation" at Defcon 21 (2013) [32], the certificate is then passed to EMET_Agent.exe which determines whether or not to trust the cert.

Due to how this works, only applications which use the certificate checks of Windows will benefit from this. Chrome can thus be protected with this, but Firefox can not.

The default profile protects the following domains:

- login.live.com (4 CAs)
- login.microsoftonline.com (4 CAs)
- login.skype.om (4 CAs)
- login.yahoo.com (7 CAs)
- secure.skype.com (4 CAs)
- twitter.com (17 CAs)
- www.facebook.com (16 CAs).

Note that although many CAs are listed (which looks bad), many of these are just different certificates owned by the same CA company (which is ok).

What it protects against

This protects against the attacks resulting from CA compromises such as the Turktrust incident[10] [6]. These are largely threats from nation states.

Weaknesses

- Only protects Internet Explorer, or via a complex manual step, can protect Chrome or alternative browsers, but not Firefox.
- Only protects a limited number of sites without manual user configuration (ex. gmail is not protected).
- Does not block processes from trusting the certificates. EMET only shows a tooltip when it's checks fail.

Impact

Conceptually, certificate pinning is a good idea, but better ideas exist, such as http://convergence.io/.

Mitigations Applied to all EMET Protected Processes

Deep Hooks

EMET hooks various functions that it views as "Critical". The full list can be found in the Appendix.

The concept behind this is explained in Ivan Fratric's ROPGuard paper [10]. Function hooking first happened with EMET in version 3.5, when it incorporated the concepts from ROPGuard.

Weaknesses

The limitations with hooking are well-known, specifically in 2004 Jamie Butler and two other anonymous authors described their weaknesses in Phrack in their article "Bypassing 3rd Party Windows Buffer Overflow Protection" [7].

The general weakness is simply to jump into code past the point where the hooks are. In the case of deep hooking, if you are hooking the function `VirtualProtect` in `kernel32.dll`, you should also hook `VirtualProtect` in `kernelbase.dll` and also the function `VirtualProtectEx`. Shahriyar Jalayeri quickly bypassed the `VirtualProtect` hook in EMET 3.5 by calling directly into `kernelbase.dll` [15]. EMET 4.0, with it's deep hooking, now does a better job of hooking the deeper functions.

A weakness specific to deep hooking would be to trick EMET into thinking that the executing code has already been checked. What I mean is that for performance reasons, if you call `kernel32:VirtualProtect`, EMET will check this call, but should set a flag to avoid checking things again when it enters `kernelbase.dll`. If this flag can be found and set, you may be able to bypass these deep hook checks.

Impact

My valuation of the impact on this mitigation is purely based on whether deep hooking should be done versus not hooking the deeper functions. In that regard, deep hooking is important, and EMET does this in a performant way.

Anti Detours

Critical function are protected by EMET by using a `jmp` hook at the start function. Microsoft even has a library called Detours[11] for providing this sort of hooking, which is why this protection is called "Anti

[10] http://www.securelist.com/en/blog/208194063/TURKTRUST_CA_Problems
[11] http://research.microsoft.com/en-us/projects/detours/

Detours". The `jmp` points to code that performs checks on how the function is being called and what called it. The bypass for this is instead of making a `call` to the start of the function, you `jmp` to the location after the hook. This is shown in figures 1, 2, and 3.

In Figure 2, where a detour hook has been performed, the attacker should jump to the address `76bab50f` to avoid the hook. By applying Anti Detours in Figure 3, EMET has inserted int 3 instructions which will cause exceptions if code tries to jump to those offsets.

```
0:001> u KERNEL32!LoadLibraryA
KERNEL32!LoadLibraryA:
76bab50a 8bff            mov     edi,edi
76bab50c 55              push    ebp
76bab50d 8bec            mov     ebp,esp
76bab50f 837d0800        cmp     dword ptr [ebp+8],0
76bab513 53              push    ebx
76bab514 56              push    esi
```

Figure 1: Unprotected critical function

```
0:003> u KERNEL32!LoadLibraryA
KERNEL32!LoadLibraryA:
76bab50a e9f151a1c0      jmp     375c0700
76bab50f 837d0800        cmp     dword ptr [ebp+8],0
76bab513 53              push    ebx
76bab514 56              push    esi
```

Figure 2: EMET hooked critical function

```
0:003> u KERNEL32!LoadLibraryA
KERNEL32!LoadLibraryA:
76bab50a e9f151a1c0      jmp     375c0700
76bab50f cc              int     3
76bab510 cc              int     3
76bab511 cc              int     3
76bab512 cc              int     3
76bab513 53              push    ebx
76bab514 56              push    esi
```

Figure 3: Anti-Detours protected critical function

Weaknesses

The obvious weakness of this solution is to simply jump further into the function. So now instead of setting your `jmp` for `76bab50f`, you set it for 4 bytes further to `76bab513`, or even further than that.

Impact

Easily bypassed once you know what it does.

Banned Functions

The only currently banned function is `LdrHotPatchRoutine`. The issue with this was discussed by Yang Yu at CanSecWest 2013 [38].

Impact

The function that was banned isn't exactly bad, but the problem was there was a fixed memory address for finding it. The was fixed in MS13-063[12] [37]. Because this has been patched, there isn't much use of it being banned by EMET anymore.

Per Process Mitigations

DEP

Calls `SetProcessDepPolicy()`.

Impact

Assuming you've already set the system wide DEP policy to always on, this has no benefit.

SEHOP

See the discussion on system-wide SEHOP for an explanation of SEHOP.

The per-process SEHOP is set-up by calling `AddVectoredExceptionHandler()` to add a process wide exception handler, as opposed to the local per-thread exception handlers that occur with try/catch statements. As Microsoft explains, "When an exception occurs in Windows XP, the vectored exception handler list is processed before the normal SEH list" [27]. So EMET's exception handler gets called, and then can scan through the SEH list to ensure it hasn't been broken.

Impact

Assuming you've already set the system wide SEHOP policy to always on, this has no benefit.

NullPage

This works by simply allocating a guard page at the address `0x00000000` by running the following code:

```
PVOID *BaseAddress = 1; // The address can not be NULL (0),
                        // else NtAllocateVirtualMemory allocs a random page
PULONG RegionSize = 0x1000;
NtAllocateVirtualMemory(GetCurrentProcess(),
    &BaseAddress, 0, &RegionSize, MEM_RESERVE, PAGE_NOACCESS);
```

You can see the result of this by attaching to a protected process in windbg and running `!address 0` and seeing it protected with `PAGE_NOACCESS`.

The are no known attacks that this protects against. Null page exploits are all privilege escalations. For example j00ru has found some [13, 14]. If the shellcode or payload has enough control in the system that it can make an attempt to escalate it's privileges, it should be able to just create a new process that does not have EMET protecting it, and escalate privileges in that process.

[12] http://blogs.technet.com/b/srd/archive/2013/08/12/mitigating-the-ldrhotpatchroutine-dep-aslr-bypass-with-ms13-063.aspx

What it protects against

No known attacks.

Impact

There are no known attacks for this, so there is no point other than a very pro-active defense-in-depth mindset and there is no system impact by doing this. Windows 8 does perform this protection by default without the use of EMET.

HeapSpray

This works the same as the Null Page protection but allocs memory at the addresses:

- `0x0a040a04`
- `0x0a0a0a0a`
- `0x0b0b0b0b`
- `0x0c0c0c0c`
- `0x0d0d0d0d`
- `0x0e0e0e0e`
- `0x04040404`
- `0x05050505`
- `0x06060606`
- `0x07070707`
- `0x08080808`
- `0x09090909`
- `0x20202020`
- `0x14141414`

Impact

This is effective at preventing many drive-by attacks that use heap sprays, but it can be easily bypassed by simply choosing different addresses to direct the instruction pointer to.

EAF

Export Address Table Filtering (EAF) is accomplished by setting hardware breakpoints using the debug registers (`DR0` and `DR1`) on the export tables for `ntdll.dll` and `kernel32.dll`. First the code finds the export table addresses in memory by parsing the PE headers of the images. Then a new thread is spawned that calls a function every 100ms. This function will set the debug registers for every thread so exceptions will be thrown whenever something tries to read the export tables. This loop every 100ms is how new threads become protected.

To see the effect of this, attach windbg to a process, and do the following:

```
0:000> ********************************************************************************
0:000> * Change thread context to something other than the one you broke in on
0:000> !threads
Index  TID  TEB         StackBase   StackLimit  DeAlloc     StackSize   ThreadProc
0  000014f4 0x7ffdd000  0x00190000  0x00184000  0x00090000  0x0000c000  0x44f125: PUTTY
1  00001838 0x7ffda000  0x00650000  0x0064f000  0x00550000  0x00001000  0x6e4bd204: EMET
2  00001810 0x7ffd7000  0x01e70000  0x01e6a000  0x01d70000  0x00006000  0x6e4b4527: EMET
3  000019a8 0x7feaf000  0x02630000  0x0262c000  0x02530000  0x00004000  0x7797fdc4: ntdll!DbgUiRemoteBreakin
Total VM consumed by thread stacks 0x00017000
0:000> ~0s
0:000> ********************************************************************************
0:000> * Show where the dr0 register is pointing
0:000> r dr0
dr0=76a1854c
0:000> !address 76a1854c

Usage:                  Image
Base Address:           769b0000
...
Image Path:             C:\Windows\SYSTEM32\KERNEL32.DLL
...
More info:              !dh 0x76930000
0:000> ********************************************************************************
0:000> * Check out the file header for KERNEL32.dll
0:000> !dh -f 0x76930000
...
   E8530 [    CBD0] address [size] of Export Directory
...
0:000> ?0x76930000+0xE8530
Evaluate expression: 1990296880 = 76a18530
```

So the `IMAGE_EXPORT_DIRECTORY` structure starts at `0x76a18530` and at `0x76a1854c` (where `dr0` points) is the RVA from the image base to the `AddressOfFunctions`. So when you find the address offset in your shellcode and go to add that to the `AddressOfFunctions` you'll cause an exception when `AddressOfFunctions` gets read in for the addition.

Windbg shows the value of `DR7` as 0 to me, which is incorrect. `DR7` controls whether the breakpoints are enabled and the size of the data to protect. From the code, this value should be `0x00f00004 | 0x000f0001 = 0x00ff0005`. This means `DR0` and `DR1` are enabled, and 4 bytes are protected from reads and writes at these addresses.

The exception handler for this will check if the code that touched this value is coming from a loaded module or not. If it is not, then EMET identifies this as an attack attempt.

Conceptually, this protection does the same thing as Piotr Bania's Protty project discussed in Phrack 0x0b [5] under the section heading "FEATURE: EXPORT SECTION PROTECTION". However, the implementation is very different. Piotr admits his implementation has performance issues, but it is a more thorough protection, and doesn't use up the limited quantity of debug registers.

Weaknesses

Back when EMET 2.0 came out, Skywing showed how to defeat EAF in his post [34] and accompanying shellcode [35].

Aaron Portnoy explained he grabs the function pointers by reading the .idata import section from other modules [30].

The cleanest bypass is by Piotr Bania who simply disables the exceptions from occurring anymore [4].

In addition to these bypasses, it may be the case that the process is not protected anyway by EAF. The User Guide states the following:

- "Some virtual machines do not support debug registers (and consequently EAF)."
- "EAF mitigation should not be applied to: programs and libraries protected that use packers or compressors, DRM or software with anti-debugging code, debuggers, and security software such as antivirus, sandbox, firewalls, etc."
- The protection profile provided by EMET disables EAF for Windows Media Player, Skype, and 7-zip.

Impact
This can be easily bypassed.

MandatoryASLR

See the discussion on system-wide ASLR.

I was unable to find where this is accomplished in the binary. According to the User Guide [22], it states "EMET's mitigations only become active after the address space for the core process and the static dependencies has been set up. Mandatory ASLR does not force address space randomization on any dependencies has been set up. Mandatory ASLR does not force address space randomization on any." The Recon presentation [3] on EMET states "EMET intercepts calls to `ntdll!NtMapViewOfSection()`" which is called by `LoadLibrary()`. So the base executable and any DLL's it depends on will not be affected by Mandatory ASLR, but if the application loads a plugin at run-time using `LoadLibrary()`, then EMET can identify if this DLL can be relocated, and if so it will allocate memory at that address, which will force the OS to load the binary somewhere besides the default location.

Impact
This will only randomize the base address for DLL's that are loaded through something like `LoadLibrary()` and only to those DLL's that have relocation data. Therefore, this will have little impact for most executables.

BottomUpASLR

This randomizes the heap by making a random number of allocations when the process starts up.

```
ULONG numAllocs = RtlRandom(GetCurrentProcessId() ^ GetTickCount()) & 0xff;
for (; numAllocs > 0; numAllocs--) {
    PVOID *BaseAddress = 0;
    PULONG RegionSize = 0x10000;
    NtAllocateVirtualMemory(GetCurrentProcess(),
        &BaseAddress, 0, &RegionSize, MEM_RESERVE, PAGE_NOACCESS);
}
```

Impact
This is effective for adding some randomization to the heap to old OS's, but has no impact for newer operating systems.

ROP Protections

I did not investigate the ROPGuard additions as these are largely copied from the open-source ROPGuard project [9]. One important limitation of all these is that none of them work on 64-bit processes.

LoadLib

The concept for this protection is to deny `LoadLibrary()` and it's brethren from loading libraries over UNC paths.

Aaron Portnoy showed [30] how to bypass this by using `MoveFileA()` to copy the remote executable, using a UNC path, then executed it locally. Hooking more functions could help with this, but ultimately this will likely always be a "nuisance" protection for attackers to bypass.

MemProt

This denies calls to memory protection APIs (ex. `VirtualProtectEx()`) to avoid marking memory as executable when the target address belongs to the thread's stack. I do not believe it protects calls to `SetProcessDEPPolicy()`.

Caller

Ensures the Critical Functions are called from `Call` instructions and not `jmp`or `ret`. Uses the Microsoft's internal MSDIS[13] [31] library for disassembling backwards from the return address, with many special cases [3].

SimExecFlow

Uses the MSDIS disassembler library to execute simulation at the return address when Critical Functions are called. Execution is simulated for 15 instructions to ensure the execution does not `ret` into another Critical Function [3].

StackPivot

Ensures "ESP register point to attacker call stack" [32].

Conclusion

EMET is a great tool that I highly recommend for protecting Windows systems. Some of the protections are very important and this is an easy and free way to enforce those. Some of the protections can be easily bypassed and are thus only useful at blocking basic exploitation attempts. However, a failed exploitation attempt is an opportunity for detection, and EMET does a good job of ensuring these failures are logged. For a casual user, logging exploit failures may have little impact, but these could be very useful in a corporate environment with a security team to investigate and correlate these incidents.

EMET successfully accomplishes it's goals of raising the cost for the attacker, while having little performance impact, and largely being universal protections that do not require code changes.

Suggestions for Improvements

- Make it easier to automatically protect any new, unprotected processes.
- Identify when protections will not be effective, such as when an image without relocations is loaded.
- Port the ROP protections to 64-bit.

[13] http://www.rsdn.ru/forum/winapi/451589.hot

- Although EMET is largely focused on back-porting security techniques to older OS's and ensuring a similar security posture across all OS's, on Windows 8 systems, please provide access to the new security protections there as documented in the presentation "Windows 8 Security and ARM" by Alex Ionescu [12].

Greetz

Thank you to the following:

- The EMET team for developing the tool.
- Matt Miller (Skape) for implementing the example of ASLR and SEHOP in WehnTrust and making it open-source, and inventing SEHOP.
- Ivan Fratric for ROPGuard and making it open-source. Also his great posts at http://ifsec.blogspot.com/
- Piotr Bania for his advances in exploit protections, and sort of making them open-source, if you count hand-coded assembly as open-source.

Appendix

MitigationOptions registry value

This section describes the registry value:
`HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Kernel\MitigationOptions`

I believe this is undocumented, so let's try to document it. Luckily MitigationInterface.dll, which is responsible for setting that value, is a .NET application which decompiles nicely, and the function of interest is `SetMitigationState`. This value only exists on Windows 8 and up. This value is broken up into 4-bit sections with the sections representing:

```
DEP             = MitigationOptions & 0x0000000f >> 0;
SEHOP           = MitigationOptions & 0x000000f0 >> 4;
MANDATORYASLR   = MitigationOptions & 0x00000f00 >> 8;
HEAP_TERMINATE  = MitigationOptions & 0x0000f000 >> 12;
BOTTOMUPASLR    = MitigationOptions & 0x000f0000 >> 16;
HA_ASLR         = MitigationOptions & 0x00f00000 >> 20;
```

For each mitigation, the possible values are:

```
off      = 6; // 0b110                      0b100 = PS_POLICY_BIT_ON
optIn    = 2; // 0b010                      0b001 = PS_MITSTATE_DISABLE
optOut   = 1; // 0b001                      0b010 = ~PS_MITSTATE_DISABLE
alwaysOn = 5; // 0b101
```

Critical Functions

The following functions are considered "critical functions" and are hooked by EMET.

```
kernel32.MapViewOfFileFromApp
ntdll.NtMapViewOfSection
kernelbase.MapViewOfFileEx
kernelbase.MapViewOfFile
kernel32.MapViewOfFileEx
kernel32.MapViewOfFile
ntdll.NtCreateSection
kernelbase.CreateFileMappingW
kernelbase.CreateFileMappingNumaW
kernel32.CreateFileMappingW
kernel32.CreateFileMappingA
ntdll.NtCreateFile
kernelbase.CreateFileW
kernel32.CreateFileW
kernel32.CreateFileA
kernel32.WinExec
ntdll.NtWriteVirtualMemory
kernelbase.WriteProcessMemory
kernel32.WriteProcessMemory
ntdll.NtCreateThreadEx
kernelbase.CreateRemoteThreadEx
kernel32.CreateRemoteThreadEx
kernel32.CreateRemoteThread
ntdll.NtCreateProcess
ntdll.NtCreateUserProcess
kernel32.CreateProcessInternalW
kernel32.CreateProcessInternalA
kernel32.CreateProcessW
kernel32.CreateProcessA
ntdll.RtlCreateHeap
kernelbase.HeapCreate
kernel32.HeapCreate
ntdll.NtAllocateVirtualMemory
kernelbase.VirtualAllocEx
kernelbase.VirtualAlloc
kernel32.VirtualAllocEx
kernel32.VirtualAlloc
ntdll.LdrLoadDll
kernelbase.LoadLibraryExW
kernelbase.LoadLibraryExA
kernel32.LoadPackagedLibrary
kernel32.LoadLibraryExW
kernel32.LoadLibraryExA
kernel32.LoadLibraryW
kernel32.LoadLibraryA
ntdll.NtProtectVirtualMemory
kernelbase.VirtualProtectEx
kernelbase.VirtualProtect
kernel32.VirtualProtectEx
kernel32.VirtualProtect
ntdll.LdrHotPatchRoutine
```

References

[1] 0xdabbad00. DEP (Data Execution Prevention) explanation by example. http://0xdabbad00.com/2012/12/07/dep-data-execution-prevention-explanation/.

[2] 0xdabbad00. How emet works. http://0xdabbad00.com/2010/09/12/how-emet-works/.

[3] Elias Bachaalany. Inside EMET 4.0. http://recon.cx/2013/slides/Recon2013-Elias%20Bachaalany-Inside%20EMET%204.pdf.

[4] Piotr Bania. BYPASSING EMET Export Address Table Access Filtering feature. http://piotrbania.com/all/articles/anti_emet_eaf.txt.

[5] Piotr Bania. NT shellcodes prevention demystified. http://www.phrack.org/issues.html?issue=63&id=15.

[6] Kurt Baumgartner. TURKTRUST CA Problems. http://www.securelist.com/en/blog/208194063/TURKTRUST_CA_Problems.

[7] Jamie Butler, Anonymous1, and Anonymous2. Bypassing 3rd Party Windows Buffer Overflow Protection. http://www.phrack.org/issues.html?issue=62&id=5.

[8] Ivan Fratric. Exploiting Internet Explorer 11 64-bit on Windows 8.1 Preview. http://ifsec.blogspot.com/2013/11/exploiting-internet-explorer-11-64-bit.html.

[9] Ivan Fratric. ROPGuard source code. https://code.google.com/p/ropguard/source/browse/.

[10] Ivan Fratric. Runtime Prevention of Return-Oriented Programming Attacks. https://ropguard.googlecode.com/svn/trunk/doc/ropguard.pdf.

[11] Wehnus from Matt Miller aka skape. WehnTrust. http://wehntrust.codeplex.com/.

[12] Alex Ionescu. Windows 8 Security and ARM. https://ruxconbreakpoint.com/assets/Uploads/bpx/alex-breakpoint2012.pdf.

[13] j00ru. CVE-2011-1282: User-Mode NULL Pointer Dereference. http://j00ru.vexillium.org/?p=932.

[14] j00ru. ZeroNights 2013 and NTVDM vulnerabilities. http://j00ru.vexillium.org/?p=2179.

[15] Shahriyar Jalayeri. Bypassing EMET 3.5′s ROP Mitigations. https://repret.wordpress.com/2012/08/08/bypassing-emet-3-5s-rop-mitigations/.

[16] McAfee. Solving the Mystery of the Office Zero Day Exploit and dep. http://blogs.mcafee.com/mcafee-labs/solving-the-mystery-of-the-office-zero-day-exploit-and-dep.

[17] Microsoft. Announcing the release of the Enhanced Mitigation Evaluation Toolkit. http://blogs.technet.com/b/srd/archive/2009/10/27/announcing-the-release-of-the-enhanced-mitigation-evaluation-toolkit.aspx.

[18] Microsoft. Demystifying Shims. http://blogs.technet.com/b/askperf/archive/2011/06/17/demystifying-shims-or-using-the-app-compat-toolkit-to-make-your-old-stuff-work-with-your-new-stuff.aspx.

[19] Microsoft. EMET 3.5 Tech Preview leverages security mitigations from the BlueHat Prize. http://blogs.technet.com/b/srd/archive/2012/07/24/emet-3-5-tech-preview-leverages-security-mitigations-from-the-bluehat-prize.aspx.

[20] Microsoft. EMET 4.0 now available for download. http://blogs.technet.com/b/srd/archive/2013/06/17/emet-4-0-now-available-for-download.aspx.

[21] Microsoft. EMET 4.0's Certificate Trust Feature. http://blogs.technet.com/b/srd/archive/2013/05/08/emet-4-0-s-certificate-trust-feature.aspx.

[22] Microsoft. EMET 4.1 Users Guide.pdf. http://www.microsoft.com/en-us/download/details.aspx?id=41138.

[23] Microsoft. How to enable SEHOP in Windows operating systems. http://support.microsoft.com/kb/956607.

[24] Microsoft. Introducing EMET v3. http://blogs.technet.com/b/srd/archive/2012/05/15/introducing-emet-v3.aspx.

[25] Microsoft. Introducing Enhanced Mitigation Experience Toolkit (EMET) 4.1. http://blogs.technet.com/b/srd/archive/2013/11/12/introducing-enhanced-mitigation-experience-toolkit-emet-4-1.aspx.

[26] Microsoft. The Enhanced Mitigation Experience Toolkit 2.0 is Now Available. http://blogs.technet.com/b/srd/archive/2010/09/02/enhanced-mitigation-experience-toolkit-emet-v2-0-0.aspx.

[27] Microsoft. Under the Hood: New Vectored Exception Handling in Windows XP. http://msdn.microsoft.com/en-us/magazine/cc301714.aspx.

[28] Microsoft. Updated EMET Version 2.0.0.3 Released. http://blogs.technet.com/b/srd/archive/2010/11/17/emet-update-2-0-0-3-released.aspx.

[29] OSR Online. Exceptional Behavior - x64 Structured Exception Handling. https://www.osronline.com/article.cfm?article=469.

[30] Aaron Portnoy. Bypassing All Of The Things. https://www.exodusintel.com/files/Aaron_Portnoy-Bypassing_All_Of_The_Things.pdf.

[31] rsdn.ru. msdis.dll. http://www.rsdn.ru/forum/winapi/451589.hot.

[32] Neil Sikka. EMET 4.0 PKI Mitigation. https://www.defcon.org/images/defcon-21/dc-21-presentations/Sikka/DEFCON-21-Sikka-EMET-4.0-PKI-Mitigation-Updated.pdf.

[33] Matt Miller (skape). Preventing the Exploitation of SEH Overwrites. http://www.uninformed.org/?v=5&a=2.

[34] Skywing. Bypassing Export address table Address Filter (EAF). http://skypher.com/index.php/2010/11/17/bypassing-eaf/.

[35] Skywing. w32-msgbox-shellcode.asm. https://code.google.com/p/w32-msgbox-shellcode/source/browse/trunk/w32-msgbox-shellcode.asm.

[36] swiat. CVE-2013-3906: a graphics vulnerability exploited through Word documents. http://blogs.technet.com/b/srd/archive/2013/11/05/cve-2013-3906-a-graphics-vulnerability-exploited-through-word-documents.aspx.

[37] swiat. Mitigating the LdrHotPatchRoutine DEP/ASLR bypass with MS13-063. http://blogs.technet.com/b/srd/archive/2013/08/12/mitigating-the-ldrhotpatchroutine-dep-aslr-bypass-with-ms13-063.aspx.

[38] Yang Yu. DEP/ASLR bypass without ROP/JIT. http://cansecwest.com/slides/2013/DEP-ASLR%20bypass%20without%20ROP-JIT.pdf.