Intel SMEP overview and partial bypass on Windows 8

ARTEM SHISHKIN

Positive Research Center

Moscow
2012

POSITIVE TECHNOLOGIES

Table of contents

ABSTRACT

This paper provides an overview of a new hardware security feature introduced by Intel and covers its support on Windows 8. Among the other common features it complicates vulnerability exploitation on a target system. But if these features are not properly configured all of them may become useless. This paper demonstrates a security flaw on x86 version of Windows 8 leading to a bypass of the SMEP security feature.

1. INTRODUCTION

With a new generation of Intel processors based on the Ivy Bridge architecture a new security feature has been introduced. It is called SMEP which stands for “Supervisor Mode Execution Prevention”. Basically it prevents execution of a code located on a user-mode page at a CPL = 0. From an attacker’s point of view this feature significantly complicates an exploitation of kernel-mode vulnerabilities because there’s just no place for a shellcode to be stored. Usually while exploiting some kernel-mode vulnerability an attacker would allocate a special user-mode buffer with a shellcode and then trigger vulnerability gaining control of the execution flow and overriding it to execute prepared buffer contents. So if an attacker is unable to execute his shellcode, the whole attack is meaningless. Of course, there are some other techniques like return-oriented programming available to exploit vulnerabilities with effective payload. But there are also certain cases when the execution environment allows bypassing the security features when it is not properly configured. Let’s take a closer look to this technology and its software support by Windows 8 operating system which introduces SMEP support.

2. HARDWARE SUPPORT OF SMEP

This section includes an overview of SMEP hardware support.

SMEP is a part of a page-level protection mechanism. In fact it uses the already existing flag of a page-table entry - the U/S flag (User/Supervisor flag, bit 2). This flag indicates whether a page is a user-mode page, or a kernel-mode. The page's owner flag defines if this page can be accessed, that is, if a page belongs to the OS kernel which is executed in a supervisor mode, it can't be accessed from a user-mode application.

SMEP is enabled or disabled via CR4 control register (bit 20). It slightly modifies the influence of the U/S flag. Whenever the supervisor attempts to execute a code located on a page with the U value of this flag, indicating that this is a user-mode page, a page fault is generated by the hardware due to the violation of an access right (the access rights are described in Volume 3, chapter 4.6 [1]).

As you can see, it doesn't generate #GP but #PF instead, so the software has to process SMEP mechanism violation in a page-fault handler. We'll use this point later when analyzing software support of this mechanism.

3. SOFTWARE SUPPORT OF SMEP

SMEP support can be detected via the "cpuid" instruction. As stated in [1] the result of a "cpuid" level 7 (sublevel 0) query indicates whether the processor supports SMEP feature – the 7^{th} bit of the EBX register has to be tested for that.

The x64 version of Windows 8 checks SMEP feature presence during the initialization of boot structures, filling in the "KeFeatureBits" variable:

KiSystemStartup() → KiInitializeBootStructures() → KiSetFeatureBits()

The same is done on x86 version of Windows 8:

KiSystemStartup() → KiInitializeKernel() → KiGetFeatureBits()

The variable "KeFeatureBits" is then used in handling a page fault.

If SMEP is supported on the current processor, it is enabled. On the x86 version it is enabled also during the startup, at phase 1 in the KiInitMachineDependent() function, and later it is initialized per processor core issuing an IPI which eventually calls KiConfigureDynamicProcessor() function. The same happens on the x64 OS version except of the fact that there is no KiInitMachineDependent() function.

So, we have SMEP enabled and “KeFeatureBits” initialized at system startup. The other part of software feature support is a code of the page fault handler. A new shim function has been added in Windows 8 – MI_CHECK_KERNEL_NOEXECUTE_FAULT(). The access fault due to SMEP or NX violation is performed inside it. The result of SMEP or NX violations is a bugcheck and a blue screen of death with a code “ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY”:

KiTrap0E()/KiPageFault() → MmAccessFault() → … →

→ MI_CHECK_KERNEL_NOEXECUTE_FAULT()

The previously mentioned function is implemented in Windows 8 only.

4. THE WAY TO BYPASS SMEP ON WINDOWS AND ITS MITIGATION

It is natural to conclude that if you can’t store your shellcode in the user-mode, you have to find a way to store it somewhere in the kernel space. The most obvious solution is using windows objects such as WinAPI (Events, Timers, Sections etc) or GDI (Brushes, DCs etc). They are accessed indirectly from the user-mode via WinAPI that uses system calls. The point is that the object body is kept in the kernel and somehow some object fields can be modified from the user-mode, so an attacker can transfer the needed shellcode bytes from the user-mode memory to the kernel-mode.

It is also obvious that an attacker needs to know where the used object’s body is located in the kernel. For that, certain information disclosure is needed. As we remember a user-mode application is unable to read kernel-mode memory. Certain source of information about the kernel space is available in Windows [2].

So it is theoretically possible to bypass SMEP on Windows due to the kernel space information disclosure. But SMEP is backed up by the fact that kernel pools where the objects are kept are now protected with NX flag (not executable) in Windows 8.

A number of WinAPI and GDI objects have been tested for being suitable to serve as a shellcode delivery tool. WinAPI objects are stored in the paged or the non-paged pool. GDI objects are stored in the paged session pool. All of them happen to be non-executable now. Moreover, according to the results of scanning page tables, there is a miserable number of

pages used from executable pools. All data buffers are now non-executable. Most of the executable (f.e. driver images) pages are not writable.

4.1. The flaw

As mentioned above, all of the objects in Windows 8 are now kept in non-executable pools. It is true for x64 version of Windows 8, and partially true for x86 version of Windows 8. The flaw is the paged session pool. It is marked as executable on the x86 version of Windows 8. So a suitable GDI object can be used to store the shellcode in a kernel memory.

The most convenient object for this purpose is a GDI palette object. It is created with CreatePalette() fuction and a supplied LOGPALETTE structure. This structure contains an array of PALETTEENTRY structures that define the color and usage of each entry in the logical palette [5]. The point is that there is no parameter validation for this palette unlike the other GDI functions that create various objects. An attacker can store any colors he wants in his palette. So he can also store any shellcode bytes there. The kernel address of palette object can be revealed through the shared GDI handle table. The contents of the palette are stored within some offset (0x54 in our case). It is not nessesary to know this offset for sure because the shellcode can be stored somewhere in the middle of spreaded NOP instructions. A schematic view of SMEP bypass is presented on Figure 1.

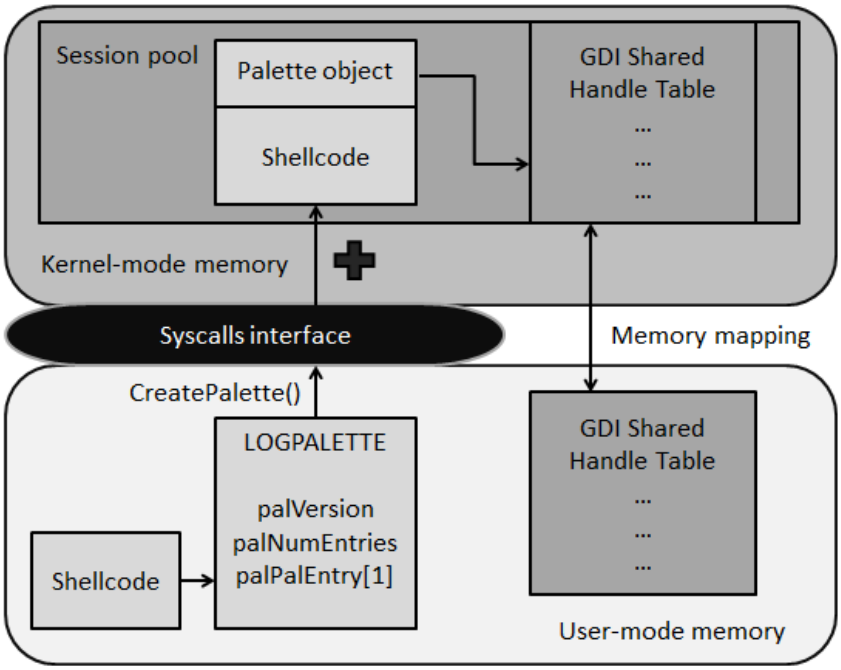

Figure 1. Schema of SMEP bypass in Windows 8 x86

A palette object provides enough space to store a big shellcode. But in fact all an attacker needs is to disable SMEP. It can be easily done by reseting 20th bit of CR4 control register and then he'll be able to execute a shellcode stored in a user-mode memory without a size limit.

Of course, there are some limitations when using paged session pool. Firstly, it is paged, so we need to consider IRQL when exploiting a certain kernel-mode vulnerability. Secondly, the session pool is mapped per user session, so we also have to consider the current session when exploiting kernel-mode vulnerability. And thirdly, in a multiprocessor environment control registers are duplicated per core, so an attacker has to use thread affinity to disable SMEP on a certain processor core.

4.2. Other SMEP bypassing attack vectors

As mentioned before, return-oriented programming can be succesfully used to bypass SMEP security feature due to the fact that this way doesn't neccesarily have to store a custom shellcode, it uses pieces of a code that already exists somewhere in the kernel memory.

There is also an opportunity of using custom OEM drivers which are not aware of using NX-compatible kernel pools.

5. CONCLUSION

In this paper we have reviewed the functioning of SMEP and its software support in Windows 8. We also have shown how it can be bypassed in certain cases because of a Windows kernel address space information disclosure and partial applying of security features. Still, the way SMEP is implemented in the x64 version of Windows 8 happens to be reliable and can be successfully used to prevent different attacks exploiting kernelmode vulnerabilities.

6. FUTURE WORK

The future work is related to inspecting custom driver modules that still use executable pools and the ways of an effective kernel information disclosure that can be used for exploiting such drivers. It is considered now as the best direction of researching SMEP bypass methods.

REFERENCES

[1] Intel: *Intel® 64 and IA-32 Architectures Developer's Manual: Combined Volumes*. Intel Corporation, 2012.

[2] Mateusz “j00ru" Jurczyk: *Windows Security Hardening Through Kernel Address Protection.* http://j00ru.vexillium.org/blog/04_12_11/Windows_Kernel_Address_Protection.pdf

[3] Mateusz ‘j00ru’ Jurczyk, Gynvael Coldwind: *SMEP: What is it, and how to beat it on Windows.* http://j00ru.vexillium.org/?p=783

[4] Ken Johnson, Matt Miller: *Exploit Mitigation Improvements in Windows 8.* Slides, Black Hat USA 2012.

[5] MSDN: *Windows GDI. http://msdn.microsoft.com/en-us/library/windows/desktop/dd145203(v=vs.85).aspx*

[6] Feng Yuan: *Windows Graphics Programming Win32 GDI and DirectDraw®.* Prentice Hall PTR, 2000.

[7] Mark Russinovich, David A. Solomon, Alex Ionescu: *Windows® Internals: Including Windows Server 2008 and Windows Vista, Fifth Edition*. Microsoft Press, 2009.

ABOUT POSITIVE TECHNOLOGIES AND POSITIVE RESEARCH CENTER

Positive Technologies is at the cutting edge of IT Security. A specialist developer of IT Security products, Positive Technologies has over a decade of experience in detecting and managing vulnerabilities in IT systems. Positive Technologies has more than 300 employees at its offices and research centres in London and Moscow. Its technology partners include IBM, Oracle, Cisco, Microsoft and HP.

Positive Research – Our innovation division, Positive Research, is one of the largest and most dynamic security research facilities in Europe. This award-winning centre carries out research, design and analytical work, threat and vulnerability analysis and error elimination. Our experts work alongside industry bodies, regulators and universities to advance knowledge in the field of information security and to assist in the development of industry standards. Naturally, this knowledge is also applied to improving the company's products and services.

Positive Research identifies over 100 0-day vulnerabilities per year in leading products such as operating systems, network equipment and applications. It has helped manufacturers including Microsoft, Cisco, Google, SAP, Oracle, Apple, and VmWare to eliminate vulnerabilities and defects that threatened the safety of their systems.

Our Product: **MaxPatrol Vulnerability and Compliance Management System**

MaxPatrol is the most accurate, comprehensive and affordable way for corporations to reduce IT security risk and meet their varied compliance requirements including PCI DSS, ISO and SOX.

Positive Hack Days

Positive Hack Days is an international forum on practical information security issues organized by the Positive Technologies company.

Company Site: www.ptsecurity.com

PHDays Forum Site: www.phdays.com

Positive Research Blog: blog.ptsecurity.com

www.ptsecurity.com
pt@ptsecurity.com
+7 (495) 744 01 44