White Paper

A Tour Beyond BIOS Memory Practices in UEFI

Jiewen Yao
Intel Corporation

Vincent J. Zimmer
Intel Corporation

Matt Fleming
Intel Corporation

June 6, 2015

Executive Summary

This paper introduces the memory map security practices in a UEFI BIOS.

Prerequisite
This paper assumes that audience has basic EDKII/UEFI firmware development experience.

Table of Contents

Overview

The main job of BIOS is to initialize the platform hardware and report information to a generic operating system (OS). The memory map is one of the most important pieces of information. The operating system can only load a kernel, driver or application in the right place if it knows how memory is allocated.
In [UEFI Memory Map], we introduced the memory map design in UEFI BIOS, and saw how a typical platform reports the memory map to an OS. In this paper we will discuss how to enhance the memory map reporting and provide security practice for memory protection to harden platforms.

Summary
This section provided an overview of the UEFI memory map.

Existing technology for memory protection

Data Execution Protection (DEP)

DEP is intended to prevent an application or service from executing code from a non-executable memory region. This helps prevent certain exploits that store code via a buffer overflow, for example.

According to UEFI/PI specification, a platform may set a memory range to be read protected, write protected, or execution protected. On x86 systems, those attributes can be set using the CPU page table. For example, if a memory range has the EFI_MEMORY_XP attribute, the consumer (OS loader) may set the IA32_EFER.NXE (No-eXecution Enable) bit in IA32_EFER MSR, then set the XD (eXecution Disable) bit in the CPU PAE page table. DEP protects against some program errors, and helps prevent certain malicious exploits, especially attacks that store executable instructions in a data area via a buffer overflow. Some OS already have Data Execution Protection (DEP) support by setting XD in the page table. Research shows 14 of 19 exploits from popular exploit kits fail with
DEP enabled. See [DEP].

Address space layout randomization (ASLR)

ASLR is intended to prevent an attacker from reliably jumping to a particular exploited function in memory. It involves randomly arranging the positions of key data areas of a program, including the base of the executable and the positions of the stack, heap, and libraries, in a process's address space. ASLR can be used with DEP. See [DEP].

PE/COFF image

UEFI specification requires that the executable image uses the PE/COFF format. The typical PE/COFF image has a DOS header, PE header, section headers and section data.

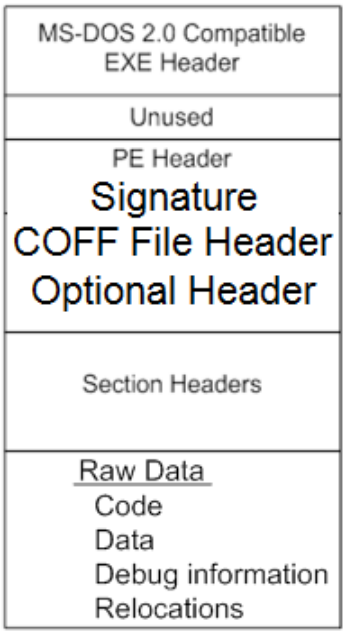

Figure 1

Each section header has a Characteristics field, which may have the SectionFlags below:

- IMAGE_SCN_CNT_CODE: The section contains executable code.
- IMAGE_SCN_MEM_EXECUTE: The section can be executed as code.
- IMAGE_SCN_MEM_READ: The section can be read.
- IMAGE_SCN_MEM_WRITE: The section can be written to.

For DEP, the PE/COFF loader may parse this information and set NX for the data section and set RO for the read-only section.

In addition, the COFF optional header has a DLL Characteristics field, which includes

- IMAGE_DLL_CHARACTERISTICS_NX_COMPAT: Image is NX compatible
- IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE: DLL can be relocated at load time.

A DEP implementation may consult the above field to enable NX or use address space layout randomization (ASLR).

Summary

This section introduces the existing technologies for memory protection.

Security Practice for memory protection in UEFI

Using DEP for UEFI/PI and the limitation

In order to user DEP in a UEFI BIOS, we hope to have a memory map like figure 2. All data regions are marked to be non-executable, and all code regions are marked to be write-protected.

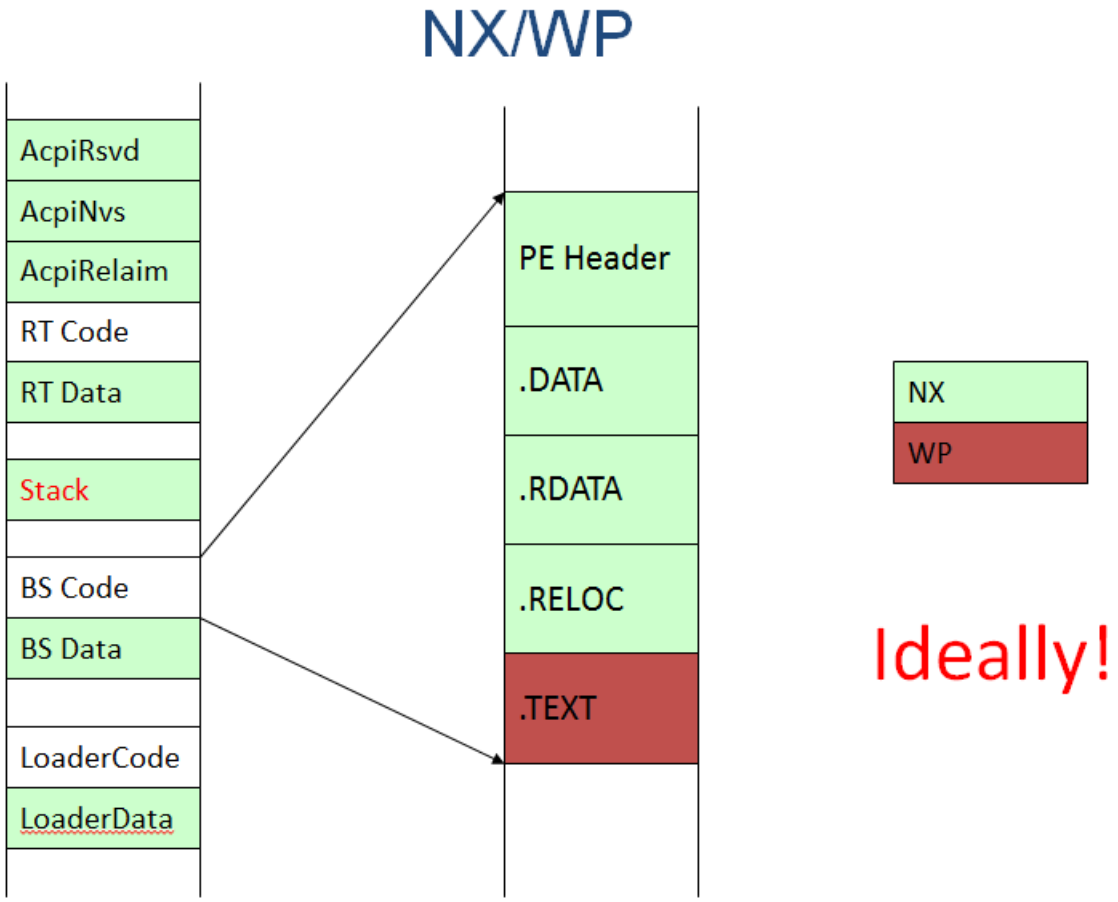

Figure 2

However, the reality is that we can currently only protect the stack, runtime data (RT Data), and ACPI reclaim memory regions. See figure 3. The known limitations are described below.

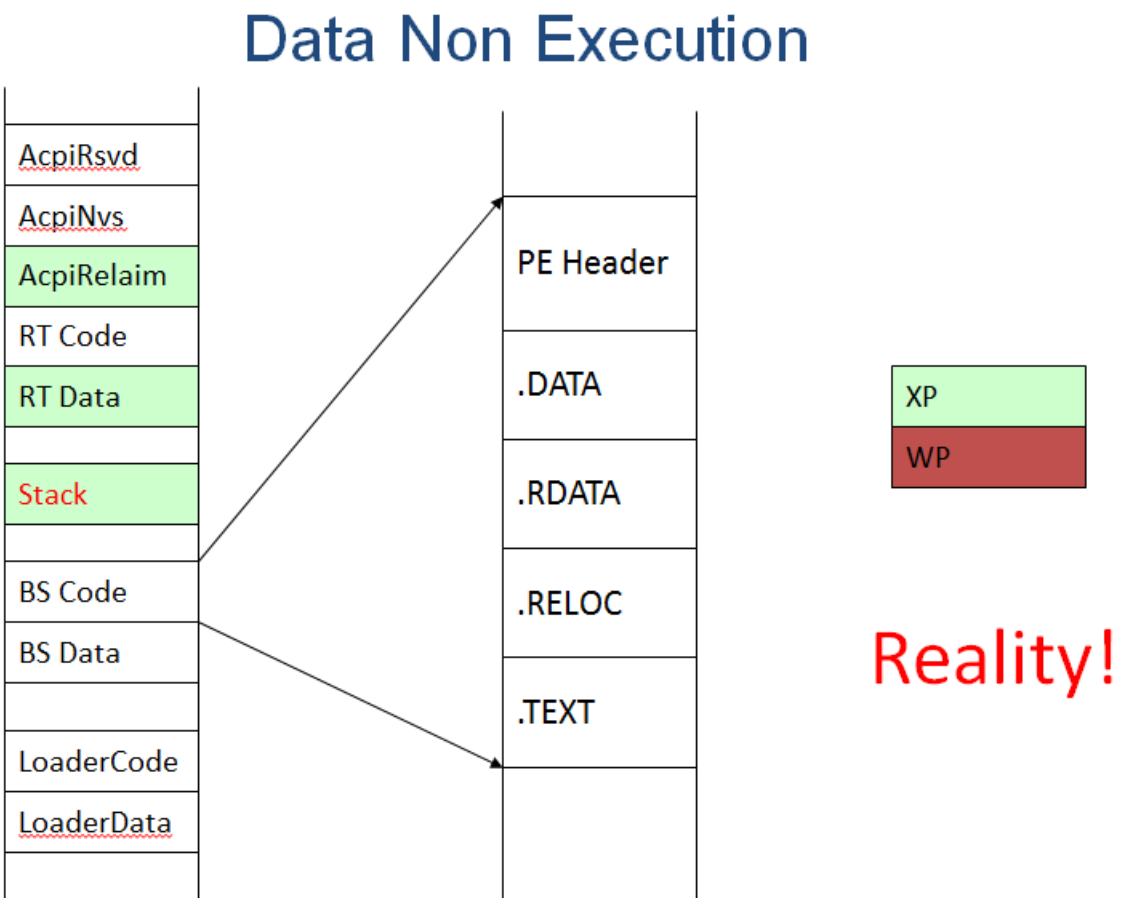

Figure 3

Specification limitation:

1) Some platforms use EFI_MEMORT_WP as a cache attribute, instead of a memory attribute.
   UEFI 2.5 specification clarifies this. It defines EFI_MEMORY_RO as awrite protection memory attribute, and uses EFI_MEMORY_WP as a cache attribute.

2) Previous UEFI specification needed paging disabled for IA32 platforms, but NX requires paging.
   UEFI 2.5 specification clarifies this. It allows a 1:1 mapped PAE page table for IA32 platforms.

Firmware limitation:

3) PE/COFF loader uses BootServicesCode/RuntimeServicesCode for both the code and data segment of a PE image.
   This implementation limitation can be resolved by updating the PE/COFF loader.

4) DxeIpl uses BootServicesData for code (DxeCore)
   This implementation limitation can be resolved by updating DxeIpl.

5) Some DXE drivers have self-modified-code.

This implementation limitation can be resolved by adding a new API to enable and disable DEP.

6) Some drivers are relocated to ACPINvs (BootScript).
   This implementation limitation can be resolved by not setting this specific region to be XD.

7) Some drivers are loaded into reserved memory for execution.
   This implementation limitation can be resolved by not setting this specific region to be XD.

8) Some ASM codes use TEXT segment keyword for code section, but it is data section in final PE image.
   This implementation limitation can be resolved by update ASM code to use .CODE as the segment name.

9) Some platforms use /MERGE:.data=.text /MERGE:.rdata=.text (link) to mix PE code section with data section.
   This implementation limitation can be resolved by removing this link option.

10) Most platforms use /ALIGN:32 (link), PE sections are not page aligned
   This implementation limitation can be resolved by overriding /ALIGN:4096. This can be done for compressed DXE driver or PEI driver. For uncompressed PEI driver, we still recommend using /ALIGN:32 to save flash image size. NOTE: There is no need to update /FILEALIGN. It can still be 32.

11) Some drivers assume PE /ALIGN is the same as /FILEALIGN, when they do PE relocation inside the driver by themselves.
   This implementation limitation can be resolved by updating code to remove the assumption.

OS limitation:

12) Some OS loaders use LoadData for code (UEFI Vista64)
   This implementation limitation can be resolved by latest OS loader.

13) Some OS loaders reverse the virtual address in memory map when calling SetVirtualAddressMap(). For example: We observed the below virtual address mapping in UEFI Win8. (The PhysicalStart is from low to high, while the VirtualStart is from high to low.

```
==============================================================================
Type       PhysicalStart    PhysicalEnd      VirtualStart     Attributes
RT_Code    000000007A157000-000000007A226FFF FFFFFFFFFFB30000 800000000000000F
RT_Data    000000007A227000-000000007A246FFF FFFFFFFFFFB10000 800000000000000F
MMIO       00000000E00F8000-00000000E00F8FFF FFFFFFFFFFB0F000 8000000000000001
MMIO       00000000FED1C000-00000000FED1FFFF FFFFFFFFFFB0B000 8000000000000001
MMIO       00000000FFA00000-00000000FFFFFFFF FFFFFFFFFF50B000 8000000000000001
==============================================================================
```

This will cause a runtime services error when we separate PE Code and Data for OS, because PE image assumes that PE sections are in sequence order.

This implementation limitation can be resolved by latest OS loader to set virtual address according to physical memory order from lowest to highest, instead of the sequence order in UEFI memory map.

Prepare DEP in UEFI for OS runtime

As an alternative, if we think the threat in firmware is low, but the threat in the OS is high, we can compromise by not using DEP in UEFI, and instead preparing the DEP environment for use by the OS runtime.

The assumptions here are:

1) OS/loader only needs to setup DEP at runtime. No protection is needed at UEFI boot time.
2) OS/loader needs to setup DEP to protect 3rd party code if possible. Since UEFI runtime services are considered to be 3rd party code, OS needs to setup DEP for PE non-code segments in UEFI runtime drivers. No boot time images need to be protected because they are gone during UEFI runtime.
3) OS/loader needs to setup DEP to protect unknown memory regions, so OS may setup DEP for reserved memory.

For example, the old UEFI memory map might look like this.

```
==========================================================================
Type        Start             End                 # Pages          Attributes
......
BS_Data     0000000077453000-0000000077453FFF 0000000000000001 000000000000000F
BS_Code     0000000077454000-0000000077456FFF 0000000000000003 000000000000000F
BS_Data     0000000077457000-000000007A1CBFFF 0000000000002D75 000000000000000F
BS_Code     000000007A1CC000-000000007A1CCFFF 0000000000000001 000000000000000F
BS_Data     000000007A1CD000-000000007A1DBFFF 000000000000000F 000000000000000F
RT_Code     000000007A1DC000-000000007A349FFF 000000000000016E 800000000000000F
RT_Data     000000007A34A000-000000007A369FFF 0000000000000020 800000000000000F
Reserved    000000007A36A000-000000007A869FFF 0000000000000500 000000000000000F
ACPI_NVS    000000007A86A000-000000007A8B6FFF 000000000000004D 000000000000000F
ACPI_Recl   000000007A8B7000-000000007A8E2FFF 000000000000002C 000000000000000F
BS_Data     000000007A8E3000-000000007A8FFFFF 000000000000001D 000000000000000F
Available   0000000100000000-0000000100DFFFFF 0000000000000E00 000000000000000F
MMIO        00000000E00F8000-00000000E00F8FFF 0000000000000001 8000000000000001
MMIO        00000000FED1C000-00000000FED1FFFF 0000000000000004 8000000000000001
MMIO        00000000FFA00000-00000000FFFFFFFF 0000000000000600 8000000000000001

  Reserved  :          1,282 Pages (5,251,072 Bytes)
  LoaderCode:            219 Pages (897,024 Bytes)
  LoaderData:              0 Pages (0 Bytes)
  BS_Code   :         40,958 Pages (167,763,968 Bytes)
  BS_Data   :         19,111 Pages (78,278,656 Bytes)
  RT_Code   :            366 Pages (1,499,136 Bytes)
  RT_Data   :             32 Pages (131,072 Bytes)
  ACPI_Recl :             75 Pages (307,200 Bytes)
  ACPI_NVS  :             77 Pages (315,392 Bytes)
  MMIO      :          1,541 Pages (6,311,936 Bytes)
  MMIO_Port :              0 Pages (0 Bytes)
  PalCode   :              0 Pages (0 Bytes)
  Available :        443,384 Pages (1,816,100,864 Bytes)
             --------------
```

```
Total Memory:          1,969 MB (2,065,293,312 Bytes)
================================================================================
```

Then after we adopt the new practice to prepare DEP environment, the new UEFI memory map could look like this:

```
================================================================================
Type       Start            End              # Pages          Attributes
......
BS_Data    0000000077453000-0000000077453FFF 0000000000000001 000000000000000F
BS_Code    0000000077454000-0000000077456FFF 0000000000000003 000000000000000F
BS_Data    0000000077457000-000000007A1CBFFF 0000000000002D75 000000000000000F
BS_Code    000000007A1CC000-000000007A1CCFFF 0000000000000001 000000000000000F
BS_Data    000000007A1CD000-000000007A1DBFFF 000000000000000F 000000000000000F
RT_Data    000000007A1DC000-000000007A225FFF 000000000000004A 800000000000400F
RT_Code    000000007A226000-000000007A228FFF 0000000000000003 800000000002000F
RT_Data    000000007A229000-000000007A22DFFF 0000000000000005 800000000000400F
......
RT_Data    000000007A340000-000000007A340FFF 0000000000000001 800000000000400F
RT_Code    000000007A341000-000000007A344FFF 0000000000000004 800000000002000F
RT_Data    000000007A345000-000000007A349FFF 0000000000000005 800000000000400F
RT_Data    000000007A34A000-000000007A369FFF 0000000000000020 800000000000400F
Reserved   000000007A36A000-000000007A869FFF 0000000000000500 000000000000400F
ACPI_NVS   000000007A86A000-000000007A8B6FFF 000000000000004D 000000000000400F
ACPI_Recl  000000007A8B7000-000000007A8E2FFF 000000000000002C 000000000000000F
BS_Data    000000007A8E3000-000000007A8FFFFF 000000000000001D 000000000000000F
Available  0000000100000000-0000000100DFFFFF 0000000000000E00 000000000000000F
MMIO       00000000E00F8000-00000000E00F8FFF 0000000000000001 8000000000004001
MMIO       00000000FED1C000-00000000FED1FFFF 0000000000000004 8000000000004001
MMIO       00000000FFA00000-00000000FFFFFFFF 0000000000000600 8000000000004001

  Reserved  :          1,282 Pages (5,251,072 Bytes)
  LoaderCode:            219 Pages (897,024 Bytes)
  LoaderData:              0 Pages (0 Bytes)
  BS_Code   :         40,971 Pages (167,817,216 Bytes)
  BS_Data   :         19,650 Pages (80,486,400 Bytes)
  RT_Code   :            119 Pages (487,424 Bytes)
  RT_Data   :            279 Pages (1,142,784 Bytes)
  ACPI_Recl :             75 Pages (307,200 Bytes)
  ACPI_NVS  :             77 Pages (315,392 Bytes)
  MMIO      :          1,541 Pages (6,311,936 Bytes)
  MMIO_Port :              0 Pages (0 Bytes)
  PalCode   :              0 Pages (0 Bytes)
  Available :        442,832 Pages (1,813,839,872 Bytes)
             -------------
Total Memory:          1,969 MB (2,065,293,312 Bytes)
================================================================================
```

There are 2 key differences:

1) The old UEFI memory map only has one RuntimeCode entry. It includes both code segment and data segment in PE image. (See figure 4) The new UEFI memory map splits the PE image and uses RuntimeCode for code segment, and RuntimeData for PE header and data segment. (See figure 5) Then the OS may have a chance to set DEP for this runtime image.
2) The old UEFI memory map does not set the EFI_MEMORY_XP or EFI_MEMORY_RO attribute for Reserved memory, ACPI NVS memory, RuntimeData, RuntimeCode and MMIO. The new UEFI memory map sets EFI_MEMORY_RO for RuntimeCode. The new UEFI memory map also set EFI_MEMORY_XP for RuntimeData and MMIO unconditionally, and for Reserved memory and ACPI NVS based on platform policy. For

example, a platform may split reserved memory and set EFI_MEMORY_XP for some of the range but clear EFI_MEMORY_XP for the other range, because it contains code.

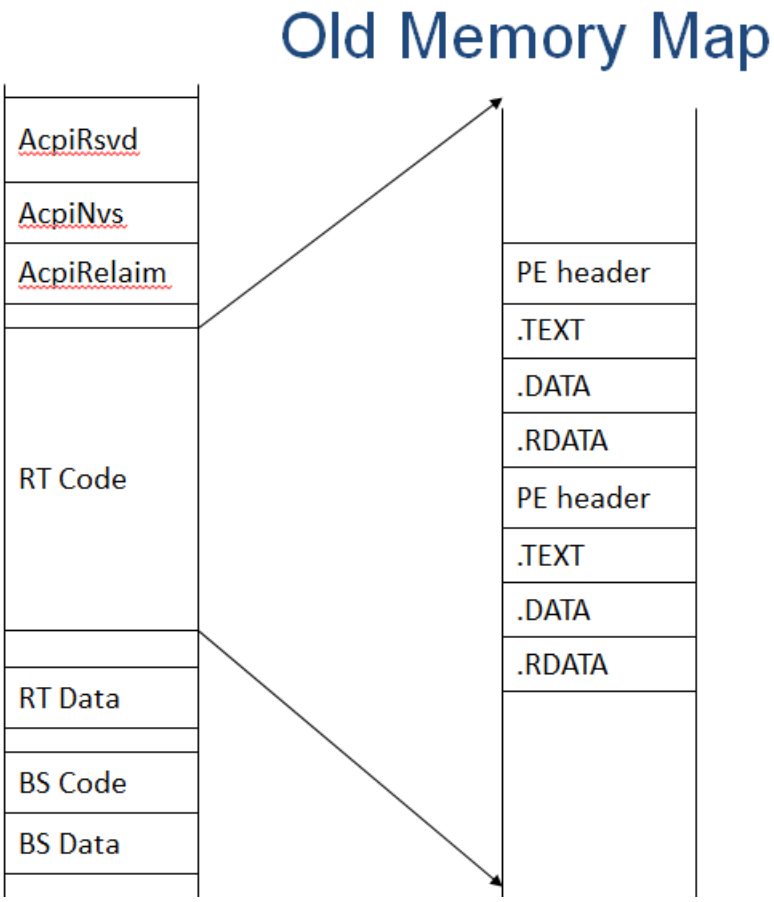

Figure 4

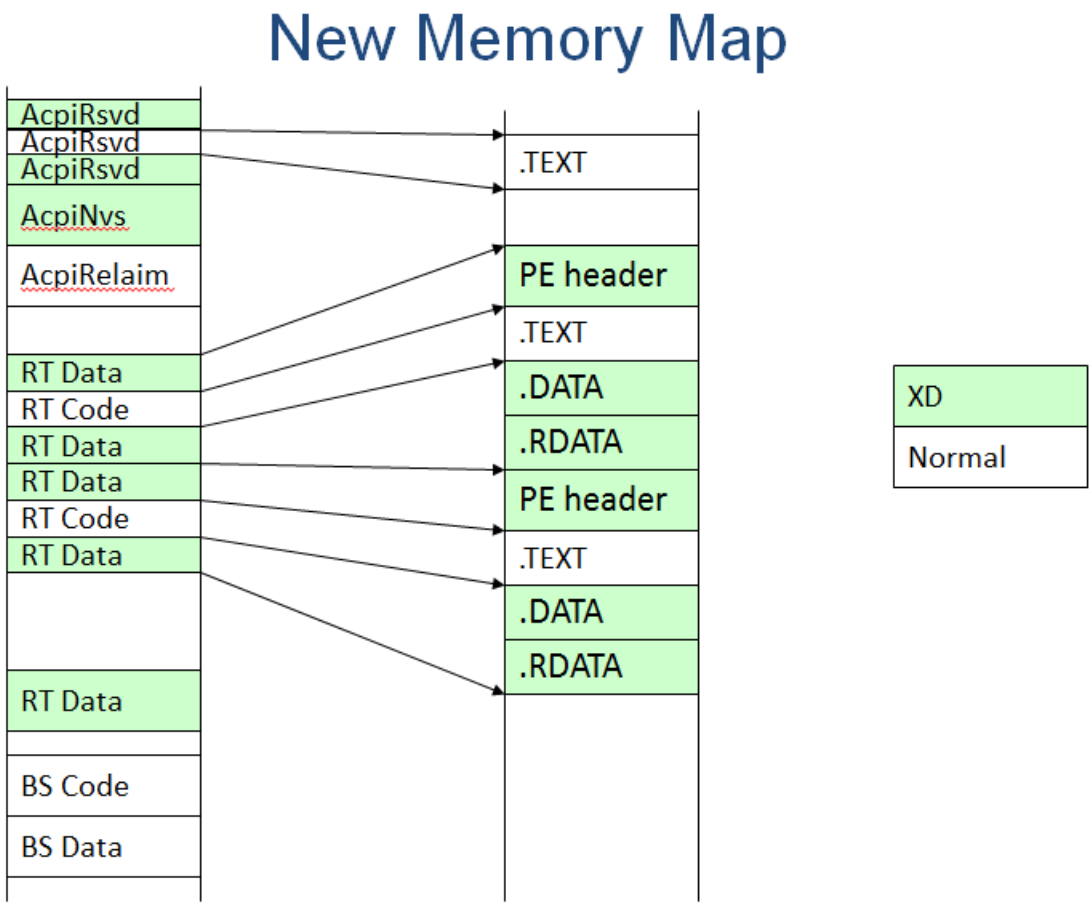

Figure 5

Using DEP at OS runtime

If firmware prepared the DEP environment the OS may setup its page table based on the UEFI memory map. The question is: How does the OS know if the firmware prepared the DEP environment?

The answer to this problem can be found in the UEFI 2.5 specification that defines the **EFI_PROTERTIES_TABLE** below:

```
      #define EFI_PROTERTIES_TABLE_GUID {0x880aaca3, 0x4adc, 0x4a04, 0x90,
0x79, 0xb7, 0x47, 0x34, 0x8, 0x25, 0xe5}
```

EFI_PROPERTIES_TABLE
This table is published if the platform meets some of the construction requirements listed in the MemoryProtectionAttributes.

```
typedef struct {
  UINT32    Version;
  UINT32    Length;
  UINT64    MemoryProtectionAttribute;
} EFI_PROPERTIES_TABLE;
```

Version This is the revision of the table. Successive versions may populate additional bits and grow the table length. In the case of the latter, the *Length* field will be adjusted appropriately

```
#define EFI_PROPERTIES_TABLE_VERSION  0x00010000
```

Length This is the size of the entire EFI_PROPERTIES_TABLE structure, including the version. The initial version will be of length 16.

MemoryProtectionAttribute This field is a bit mask. Any bits not defined shall be considered reserved. A set bit means that the underlying firmware has been constructed responsive to the given property.

```
//
// Memory attribute (Not defined bits are reserved)
//
#define
EFI_PROPERTIES_RUNTIME_MEMORY_PROTECTION_NON_EXECUTABLE_PE_DAT
A     0x1
```

This bit implies that the UEFI runtime code and data sections of the executable image are separate and aligned to at least a 4KiB boundary. This bit also implies that the data pages do not have any executable code.

The result of this table publication is that the platform provider needs to ensure that there is no executable code in the EFI data section and that the code isn't self-modifying, so that data can be made execute-protectable and code made read-only.

But what about other memory regions in the system, such as the other EFI_MEMORY_TYPES that persist into UEFI Runtime.

If the attributes are not listed for protectability, then the runtime environment that invokes ExitBootServices() and makes subsequent calls into the UEFI runtime cannot assert those protection properties in its memory management unit (MMU), such as page tables.

In other words, EFI reserved or ACPI regions cannot be made DEP or RO if the attributes do not allow for this.

The reason is that the platform firmware may need unfettered access to those regions and applying such protections could 'break' the firmware.

EDKII support

Current EDKII implementation already supports UEFI 2.5 properties table.

1) Platforms can use gEfiMdeModulePkgTokenSpaceGuid.PropertiesTableEnable (https://github.com/tianocore/edk2-MdeModulePkg/blob/master/MdeModulePkg.dec) to control whether this feature is enabled/disabled.

2) DXE Core PropertiesTable.c will create properties table based on the above policy PCD. (https://github.com/tianocore/edk2-MdeModulePkg/blob/master/Core/Dxe/Misc/PropertiesTable.c) It will force policy for RuntimeCode(EFI_MEMORY_RO), RuntimeData(EFI_MEMORY_XP), and MMIO(EFI_MEMORY_XP).

   PropertiesTable.c will hook the GetMemoryMap() service, and parse PE/COFF images in RuntimeCode regions. If the runtime code is page aligned, RuntimeCode entries are split into multiple RuntimeData entries and RuntimeCode entries according to PE Section Table Headers Characteristics , IMAGE_SCN_CNT_CODE SectionFlags. The benefit is that no PE/COFF loader needs to be updated, and no DXE Core BIN algorithm needs to be updated.

3) A standalone PropertiesTableAttributesDxe driver will set memory attributes for ACPINvs and Reserved memory. (https://github.com/tianocore/edk2-MdeModulePkg/tree/master/Universal/PropertiesTableAttributesDxe) It will use the default policy for ACPINvs(EFI_MEMORY_XP) and Reserved(EFI_MEMORY_XP).

Call for action

In order to support entry splitting in the new enhanced memory map, the firmware must do the following:

1) Override link flags by using /ALIGN:4096 for runtime drivers, so that the PE Sections are page aligned. /FILEALIGN can still be 32.
2) If a platform does not have runtime executable in ACPI NVS or Reserved memory regions, the platform can use default EDKII PropertiesTableAttributesDxe driver.
3) If a platform finds that ACPI NVS or Reserved memory regions are not suitable to be marked as EFI_MEMORY_XP, a platform may use other PropertiesTableAttributesDxe drivers to set proper attributes based upon platform needs.

Summary

This section introduces the security practice for memory protection.

Linux Usage

IA32, X64 and Aarch64 Linux implementations map EFI runtime data regions with the non-executable (NX) page table bit set if it is supported by the hardware platform, irrespective of any of the memory protection attributes for the EFI memory descriptor. EFI runtime code regions are marked executable.

Itanium maps both the EFI runtime data regions and EFI runtime code regions as executable irrespective of any of the memory protection attributes in the EFI memory descriptor.

Cacheability EFI descriptor attributes are honored for all architectures on Linux.

Conclusion

Memory map is important information from firmware to OS. This paper describes how to enable page level execution protection to harden platforms.

Glossary

ACPI – Advanced Configuration and Power Interface. The specification defines a new interface to the system board that enables the operating system to implement operating system-directed power management and system configuration.

ASLR – Address Space Layout Randomization.

DEP – Data Execution Protection.

MMIO – Memory Mapped I/O.

NX – No Execution. See DEP.

PE/COFF – Portable Executable and Common Object File Format. The executable file format for UEFI.

PI – Platform Initialization. Volume 1-5 of the UEFI PI specifications.

UEFI – Unified Extensible Firmware Interface. Firmware interface between the platform and the operating system. Predominate interfaces are in the boot services (BS) or pre-OS. Few runtime (RT) services.

XD – Execution Disable. See DEP.

References

[ACPI] ACPI specification, Version 5.1 www.uefi.org

[DEP] Exploit Mitigation Improvements in Windows 8, http://media.blackhat.com/bh-us-12/Briefings/M_Miller/BH_US_12_Miller_Exploit_Mitigation_Slides.pdf

[EDK2] UEFI Developer Kit www.tianocore.org

[IA32SDM] Intel® 64 and IA-32 Architectures Software Developer's Manual, www.intel.com

[PE/COFF] Microsoft Portable Executable and Common Object File Format Specification, Revision 8.3 https://msdn.microsoft.com/library/windows/hardware/gg463119.aspx

[SB] Nystrom, Nicoles, Zimmer, "UEFI Networking and Pre-OS Security," Intel Technology Journal, Volume 15, Issue 1, October 2011
http://www.intel.com/content/www/us/en/research/intel-technology-journal/2011-volume-15-issue-01-intel-technology-journal.html

[UEFI] Unified Extensible Firmware Interface (UEFI) Specification, Version 2.5
www.uefi.org

[UEFI Book] Zimmer, et al, "Beyond BIOS: Developing with the Unified Extensible Firmware Interface," 2nd edition, Intel Press, January 2011

[UEFI Memory Map] Yao, Zimmer, "A Tour Beyond BIOS Memory Map in UEFI_BIOS", February 2015
http://firmware.intel.com/sites/default/files/resources/A_Tour_Beyond_BIOS_Memory_Map_in%20UEFI_BIOS.pdf

[UEFI Overview] Zimmer, Rothman, Hale, "UEFI: From Reset Vector to Operating System," Chapter 3 of *Hardware-Dependent Software*, Springer, February 2009

[UEFI PI Specification] UEFI Platform Initialization (PI) Specifications, volumes 1-5, Version 1.3 www.uefi.org

[WHCK System] Windows Hardware Certification Requirements for Client and Server Systems
http://msdn.microsoft.com/en-us/library/windows/hardware/jj128256.aspx

Authors

Matt Fleming (matt.fleming@intel.com) is the upstream Linux kernel maintainer for UEFI with the Software and Services Group at Intel Corporation. He is also the creator of the Linux UEFI Validation OS.

Jiewen Yao (jiewen.yao@intel.com) is an EDKII BIOS architect, EDKII TPM2 module maintainer, ACPI/S3 module maintainer, and FSP package owner with the Software and Services Group at Intel Corporation.

Vincent J. Zimmer (vincent.zimmer@intel.com) is a Senior Principal Engineer and chairs the UEFI networking and security sub-team with the Software and Services Group at Intel Corporation.