

Revisiting Memory Protections in EDK2

Agenda

Intro & Overview

- Current State
- Minimal System Protections
- Firmware Memory Mitigations: Present & Future

Memory Protections in Production

- Branch details
- Problems & Solutions with Memory Protections Today
- Paging Audit Example

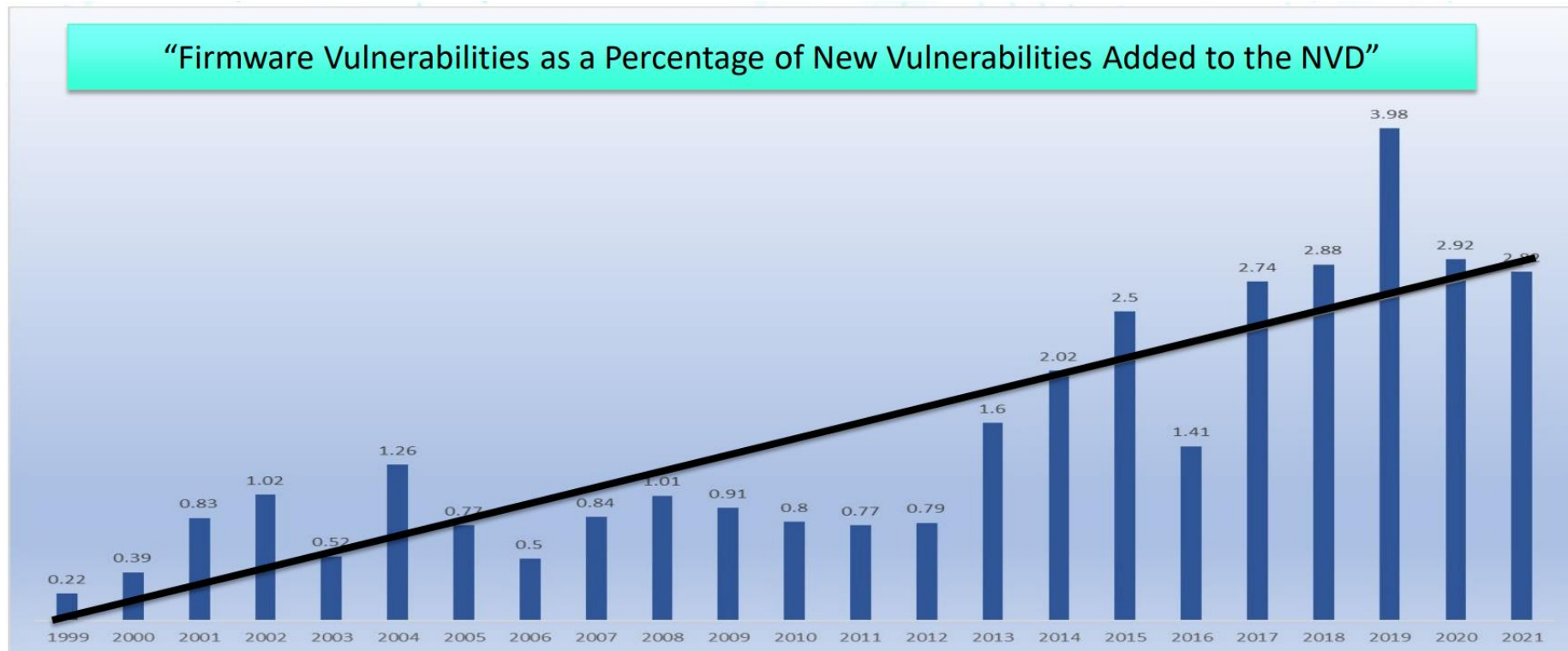
Tools & Tests

Future Work: Challenges and Proposals

- Simplifying Memory Protections

Intro & Overview

Firmware Attacks on the Rise



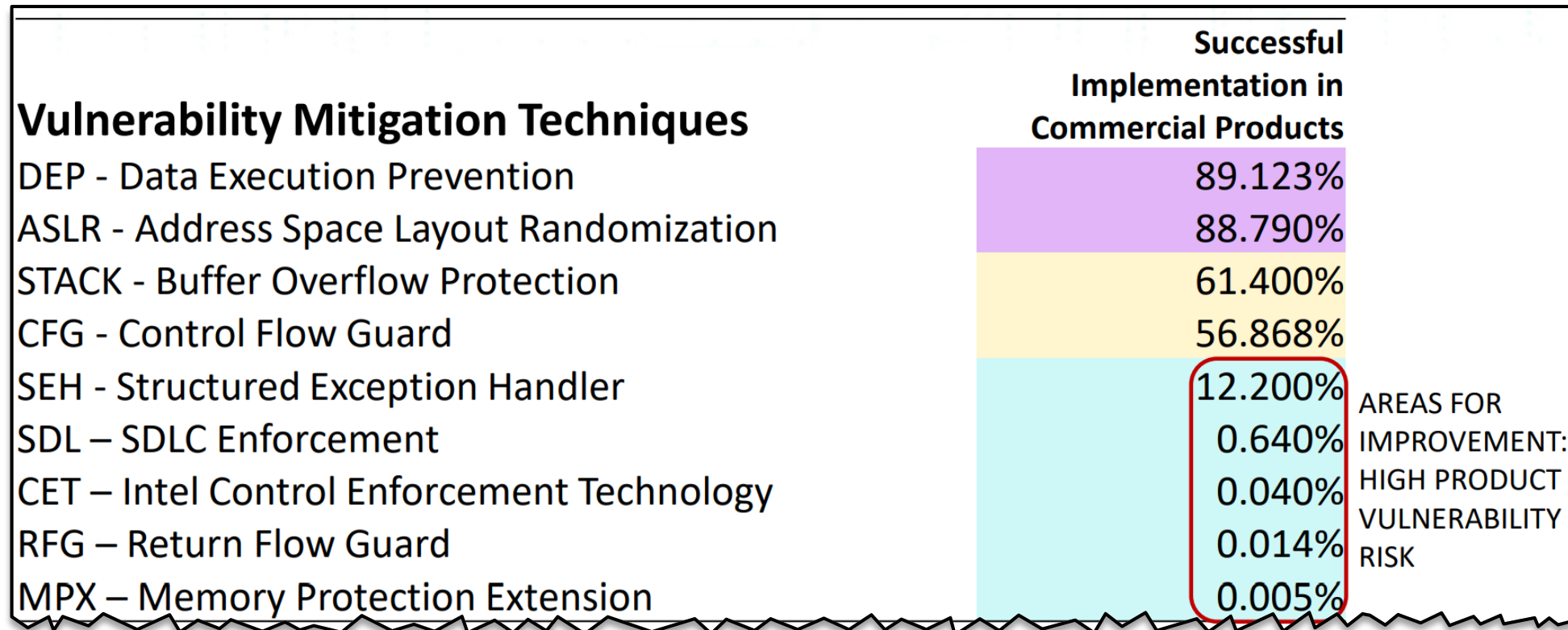
Source: NIST NVD, 3/17/21

Takeaway: This is getting worse

Source: [DHS CISA Strategy to Fix Vulnerabilities Below the OS Among Worst Offenders](#)

UEFI Memory Protections are Behind

None of these are enabled in typical UEFI firmware:



Source: [DHS CISA Strategy to Fix Vulnerabilities Below the OS Among Worst Offenders](#)

10 Years Later...



NX Protection in SMM



- In 2012 Phoenix worked with Microsoft to create and submit a proposal for NX (No-eXecute) protections in UEFI
- Modern x86 CPUs provide support for NX as a bit that can be set in PAE and IA32E page tables - called XD (eXecute Disable) in Intel Volume 3

EFI Development Kit (EDK II) documentation...

• Stack canaries

Current EDK II uses `/GS-` for MSVC and `-fno-stack-protector` for GCC. The stack guard is disabled by default. The reason is that EDK II does not link against any compiler provided stack guard.

• Limitations

The guard in Pre-EFI Initialization (PEI) phase is not supported yet, because most Intel® Architecture (IA) platforms only supports 32bit PEI and paging is not enabled. From technical perspective, we can add paging-based guard after the permanent memory is initialized in PEI. Stack guard, heap guard or NULL pointer detection can be enabled.

• DEP

The Unified Extensible Firmware Interface (UEFI) [\[www.uefi.org\]](http://www.uefi.org) specification allows "Stack may be marked as non-executable in identity mapped page tables." UEFI also defines

allocation need 12K memory. The heap guard feature will increase memory consumption and may cause memory out of resource. Especially, the System Management Mode (SMM) code runs in the limited System Management Mode RAM (SMRAM) (4M or 8M).

• ASLR

The current EDK II code does not support address space randomization.

We have observed the performance downgrade in UEFI Shell,

• NULL Pointers

Zero address is considered as an invalid address in most programs. In legacy BIOS, zero address is valid address in legacy BIOS because the 16bit interrupt vector table (IVT) is at address zero. In current UEFI firmware, zero address is always mapped.

For heap pool detection, we cannot enable both underflow and overflow detection in one image, because the guard page must be 4K aligned and the allocated pool is either adjacent to head guard page or tail guard page.

Current State

- Firmware is an increasingly attractive attack target.
- Firmware lacks basic memory mitigations present in other system software for decades.
- Firmware implementations vary widely in reliability and security assurance.
- Firmware is foundational to system security – the chain of trust and System Management Mode. Firmware attack vectors threaten to compromise OS security.
- It is not possible to attest to an operating system what memory protections are enabled on a system.

Investing in Firmware Security

1. Known firmware exploits are not being protected against.
2. Firmware vulnerabilities are becoming more pervasive and increasing in frequency.

We **must** do better to harden platforms against exploits of common memory-safety vulnerabilities.

Longer term, firmware protections will expand into:

Broader Windows requirements / UI

Secured-Core PCs

Defining a “Minimally Protected System”

The following UEFI memory protections should always be enabled at a minimum in DXE (more details in the appendix):

1. Null pointer detection
2. Stack guard
3. Heap guard (NX stack and stack guard page)
4. Image protection (read-only code sections and NX data sections)

Microsoft is committed to working with vendors to help address compatibility issues to enable these protections.

Partnership is needed across the industry to enable this.

Memory Mitigations and Windows

Exact details are TBD. Examples:

- Testing: Windows Logo Requirement
 - Example:
 - Memory Attribute Protocol must be present.
 - Firmware must demonstrate that it implements the memory mitigations required for a “minimally protected system” when the Memory Attributes Protocol is present.
- Transparency: Visibility in the Windows UI
 - Example
 - Firmware Security features will be listed out alongside their enablement state in the Windows Security App.
 - For example: ☒ UEFI minimum memory protections enabled
- Note: [Microsoft 3rd Party UEFI CA requirements](#) were updated in November 2022 to include NX compatibility requirements.

Memory Mitigations in edk2: Present

Today:

- Memory protections are not commonly enabled.
 - Compatibility is a leading concern.
- There's a lack of memory mitigation tools and test infrastructure.
- There's not much documentation about debugging errors.

Test Results

RW+X

Description: No memory range should have page attributes that allow RW+X
Status: Failed (15)

Set This As Filter For List

Data Sections are No-Execute

Description: Image data sections should be no-execute
Status: Success

Code Sections are Read-Only

Description: Image code sections should be read-only
Status: Success

Base Address	End Address	Page Size	# of Pages	Present	Read/Write	Execute	Privilege	UEFI Memory Type	GCD Memory Type
0x00000A0000	0x00000FFFFFFF	4k	96	Yes	Enabled	Enabled	Supervisor	None	EfiGcdMemoryTypeMemoryMappedIo
0x0006092000	0x0006092FFFF	4k	1	Yes	Enabled	Enabled	Supervisor	EfiBootServicesCode	EfiGcdMemoryTypeSystemMemory
0x0007322000	0x0007322FFFF	4k	30	Yes	Enabled	Enabled	Supervisor	EfiReservedMemoryType	EfiGcdMemoryTypeSystemMemory
0x0007AAE000	0x0007AAEFFFF	4k	1	Yes	Enabled	Enabled	Supervisor	EfiBootServicesCode	EfiGcdMemoryTypeSystemMemory
0x0007EC3000	0x0007EF3FFFF	4k	49	Yes	Enabled	Enabled	Supervisor	EfiBootServicesCode	EfiGcdMemoryTypeSystemMemory
0x0008000000	0x0007FFFFFFF	2m	960	Yes	Enabled	Enabled	Supervisor	None	EfiGcdMemoryTypeNonExistent
0x0008000000	0x000FBFFFFFFF	2m	992	Yes	Enabled	Enabled	Supervisor	None	EfiGcdMemoryTypeMemoryMappedIo

OvmfPkg w/ edk2 Memory
Protection PCDs Enabled

Memory Mitigations in edk2: Future

Closing the Gap to Realize a “Minimally Protected System”:

1. Enable “minimal protections” by default in edk2.
2. Define “protection profiles” that offer platforms well-defined regularly tested sets of protections that trade off security for compatibility.
3. Push the industry to produce compatible firmware through requirements and testing.
4. Make tools to audit and verify memory mitigations available in edk2.
 - For example:
 1. Binary image verification tools
 2. Boot audit tests
 3. Integration and unit tests
5. Clearly document how to debug common memory protection violations in TianoCore hosted documentation. Make error messages useful.
6. Simplify application of memory protections during boot by consolidating responsibilities.

Memory Protections in Production

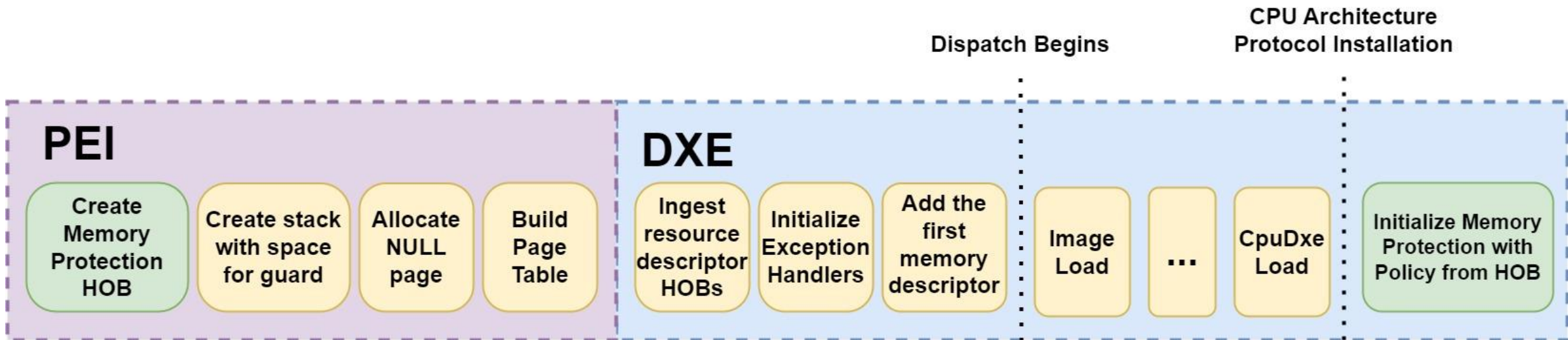
A Note on This Section

- The following section summarizes the changes Project Mu has made to improve memory protections.
- A more in-depth technical document and a branch in edk-staging containing the memory protection changes described here will be made available after this meeting.
- This is not an ideal solution -- more fundamental plumbing should be done in DXE (particularly DxeCore) for a more robust solution.
- We'd like to make these DXE improvements in collaboration with the community.
- Some ideal changes will be discussed in the next section. Work on these changes will be done in the edk2-staging branch.

Problem: Memory Protections are Fixed at Build

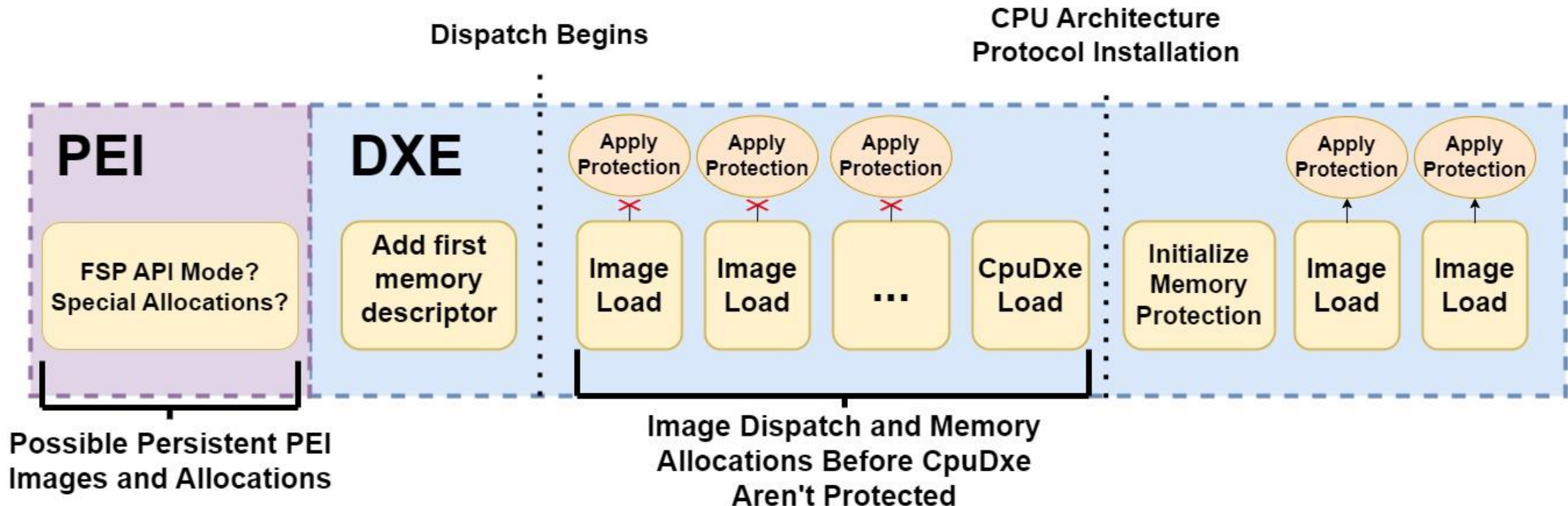
Solution: Memory protection configuration set via a HOB

- Enables the system to adjust memory protection settings at runtime
- Libraries are provided to populate a global variable containing the protection policy (similar to UefiBootServicesLib)
- DXE and MM use separate HOB entries and settings profiles



Problem: Initialization Leaves Protection Gaps

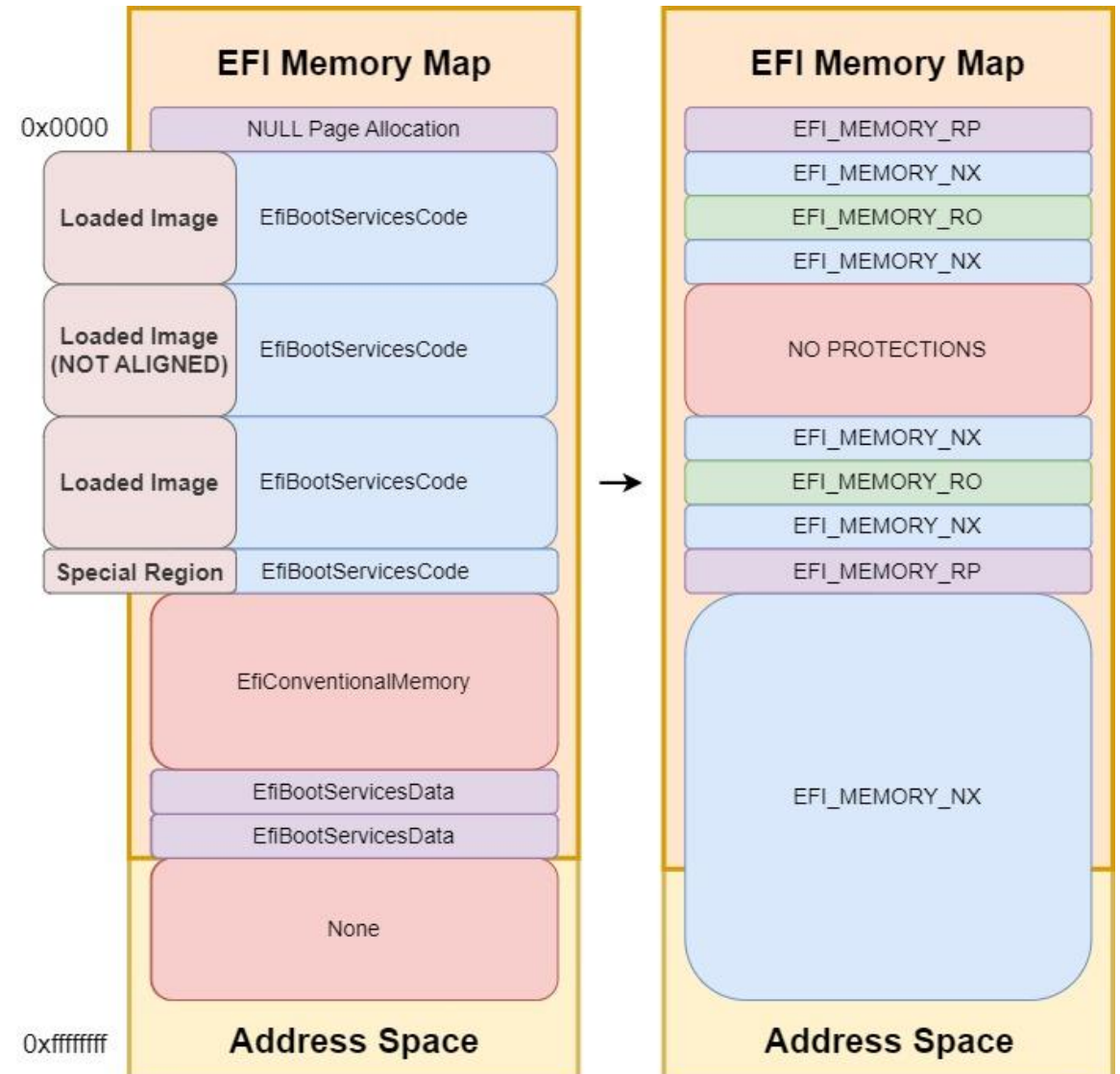
Cannot initialize memory protection policy to memory regions based on type alone because allocations are made before CpuDxe and there may be persistent memory allocations from PEI



Problem: Initialization Leaves Protection Gaps

Solution: Augment Memory Protection Initialization

- Allow loaded images to be non-protected and enable blocking their load if policy dictates
- Allow memory allocated before CPU Arch Protocol to have protections which deviate from policy (Special Regions)
- Protect the entire address space instead of just what's in the EFI memory map
- Enable setting NX policy on code memory types

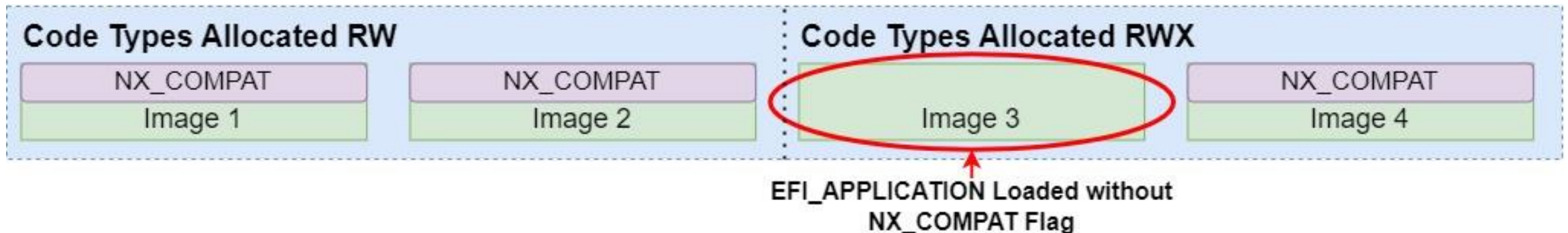


Problem: OPROMS may not be Compatible

Solution: Use NX_COMPAT PE/COFF Flag to Designate Compatibility

- Indicates OPRON expects code buffers to be allocated RW- and will use the Memory Attribute Protocol to update code buffer attributes to R-X
- This is a temporary compatibility option which will be revoked in the future
- If an image of subsystem type EFI_APPLICATION is loaded without the flag, code memory types will be allocated as RWX for the remainder of boot

Windows Bootloader is built with NX_COMPAT



Paging Audit with Described Changes

Test Results

RW+X

Description: No memory range should have page attributes that allow read, write, and execute

Status: Success

Data Sections are No-Execute











Description: Image data sections should be no-execute

Status: Success

Code Sections are Read-Only

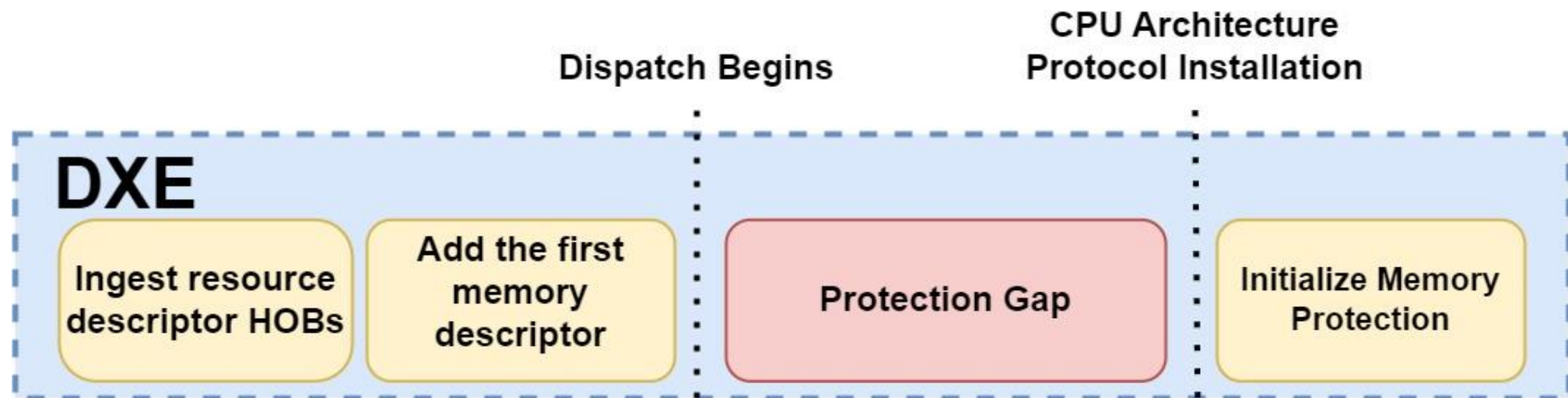
Description: Image code sections should be read-only

Status: Success

Base Address 	End Address 	Page Size 	# of Pages 	Present 	Read/Write 	Execute 	Privilege 	UEFI Memory Type 	GCD Memory Type 
0x0000088000	0x000009FFFF	4k	24	Yes	Enabled	Disabled	Supervisor	EfiConventionalMemory	EfiGcdMemoryTypeSystemMemory
0x00000A0000	0x00000FFFFFFF	4k	96	Yes	Enabled	Disabled	Supervisor	None	EfiGcdMemoryTypeMemoryMappedIo
0x0000100000	0x00001FFFFFFF	4k	256	Yes	Enabled	Disabled	Supervisor	EfiConventionalMemory	EfiGcdMemoryTypeSystemMemory
0x0000200000	0x00007FFFFFFF	2m	3	Yes	Enabled	Disabled	Supervisor	EfiConventionalMemory	EfiGcdMemoryTypeSystemMemory
0x0000800000	0x0000805FFF	2m	1 (p)	Yes	Enabled	Disabled	Supervisor	EfiConventionalMemory	EfiGcdMemoryTypeSystemMemory
0x0000806000	0x0000806FFF	2m	1 (p)	Yes	Enabled	Disabled	Supervisor	EfiBootServicesData	EfiGcdMemoryTypeSystemMemory
0x0000807000	0x000080AFFF	2m	1 (p)	Yes	Enabled	Disabled	Supervisor	EfiConventionalMemory	EfiGcdMemoryTypeSystemMemory
0x000080B000	0x000080BFFF	2m	1 (p)	Yes	Enabled	Disabled	Supervisor	EfiBootServicesData	EfiGcdMemoryTypeSystemMemory

Challenges Securing Platforms

- Many table manipulation interfaces (CPU Arch Protocol, DXE Services, Memory Attribute Protocol, CpuPageTableLib, ArmMmuLib)
- DXE protections currently rely on PEI logic
- A protection gap between memory Init and CPU Arch Protocol Install



Tools and Tests

Tools and Tests

Memory Protection Test App [\[link\]](#):

- Tests page guards, pool guards, stack guard, NX protection, NULL detection.
- Test can be run by violating expected protections and either clearing the fault or performing a reset.
- Test can be run by checking allocated buffers using the Memory Attribute Protocol.

Memory Attribute Protocol Test App [\[link\]](#):

- Tests the Memory Attribute Protocol functionality.
- Tests for some bugs found when adding NX compatibility to the Windows Bootloader.

PE/COFF Image Validation [\[link\]](#):

- Tests PE images against a set of tests and associated requirements.
- This can help confirm that NX_COMPAT is set, sections are W^X, aligned, etc.

Tools and Tests

DXE Paging Audit [\[link\]](#):

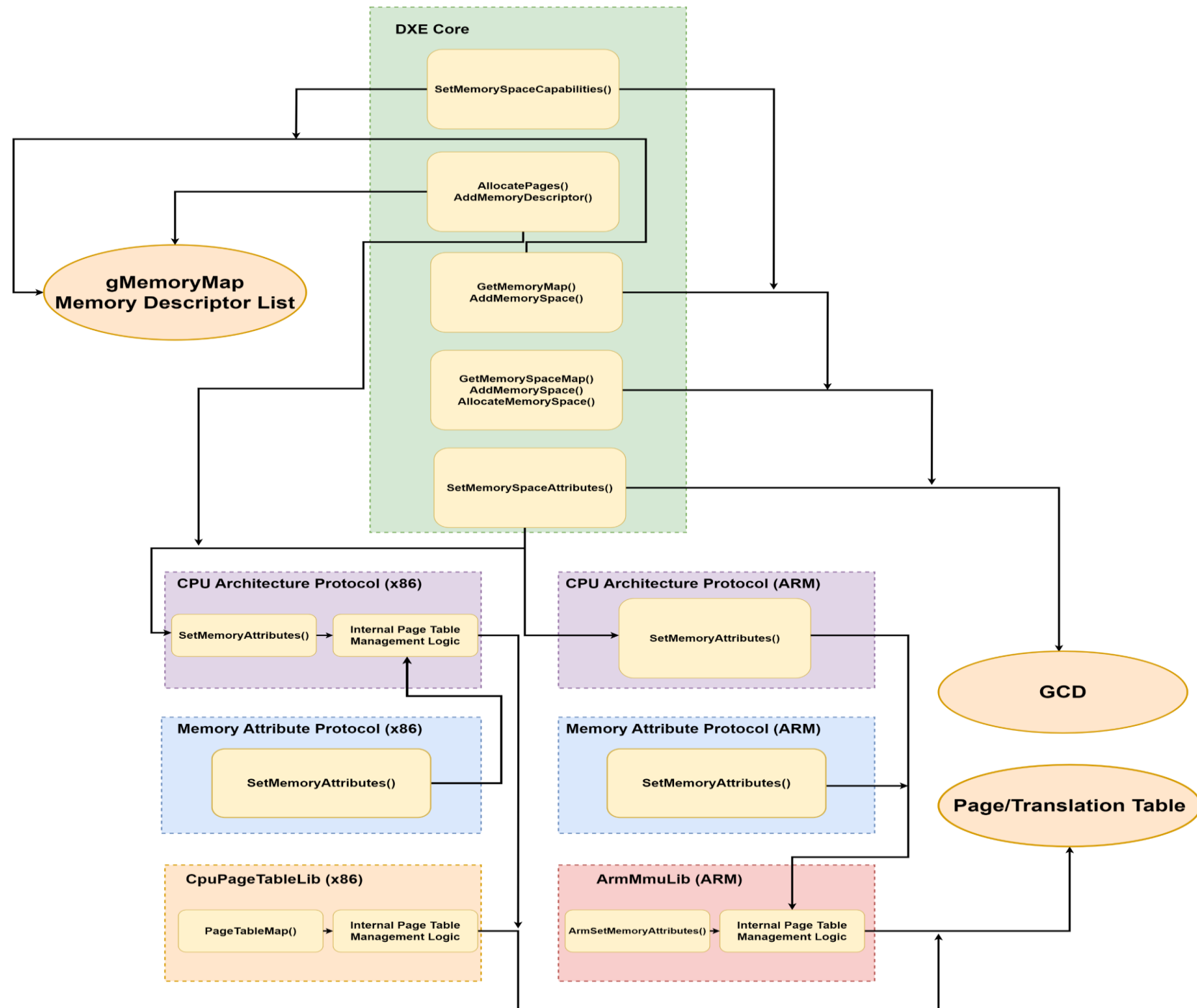
- Collects the page table, stack information, EFI and GCD memory maps, loaded images, and processor specific info to generate a human-readable snapshot of memory at the time of the audit.

0x007EBDA000	0x007EBDAFFF	4k	1	No	Disabled	Enabled	User	EfiACPIMemoryNVS	EfiGcdMemoryTypeSystemMemory	Not Tracked	GuardPage	Nothing Found
0x007EBDB000	0x007EBFDFFF	4k	35	Yes	Enabled	Disabled	Supervisor	EfiACPIMemoryNVS	EfiGcdMemoryTypeSystemMemory	Not Tracked	None	Nothing Found
0x007EBFE000	0x007EBFFFFF	4k	2	Yes	Enabled	Disabled	Supervisor	EfiBootServicesData	EfiGcdMemoryTypeSystemMemory	Not Tracked	None	Nothing Found
0x007EC00000	0x007EDFFFFF	2m	1	Yes	Enabled	Disabled	Supervisor	EfiBootServicesData	EfiGcdMemoryTypeSystemMemory	Not Tracked	None	Nothing Found
0x007EE00000	0x007EED6FFF	4k	215	Yes	Enabled	Disabled	Supervisor	EfiConventionalMemory	EfiGcdMemoryTypeSystemMemory	Not Tracked	None	Nothing Found
0x007EED7000	0x007EED7FFF	4k	1	No	Enabled	Disabled	Supervisor	EfiBootServicesData	EfiGcdMemoryTypeSystemMemory	Not Tracked	BSP Stack Guard	Nothing Found
0x007EED8000	0x007EEF6FFF	4k	31	Yes	Enabled	Disabled	Supervisor	EfiBootServicesData	EfiGcdMemoryTypeSystemMemory	Not Tracked	BSP Stack	Nothing Found
0x007EEF7000	0x007EEF7FFF	4k	1	Yes	Enabled	Disabled	Supervisor	EfiBootServicesCode	EfiGcdMemoryTypeSystemMemory	DATA	None	DxeCore.pdb
0x007EEF8000	0x007EF18FFF	4k	33	Yes	Disabled	Enabled	Supervisor	EfiBootServicesCode	EfiGcdMemoryTypeSystemMemory	CODE	None	DxeCore.pdb
0x007EF19000	0x007EF2EFFF	4k	22	Yes	Enabled	Disabled	Supervisor	EfiBootServicesCode	EfiGcdMemoryTypeSystemMemory	DATA	None	DxeCore.pdb

Future Work

Current State

- 5 different places to get/set attributes
- Not all synced which creates a torn state



FlattenDxeCore

Main Goals:

1. Close the gap of memory protection before CpuDxe is ready
2. Streamline the page table setup from the core implementation
3. Simplify the memory protection interfaces
4. Have DXE Core drive all memory attributes with the GCD as the source of truth

FlattenDxeCore proposal

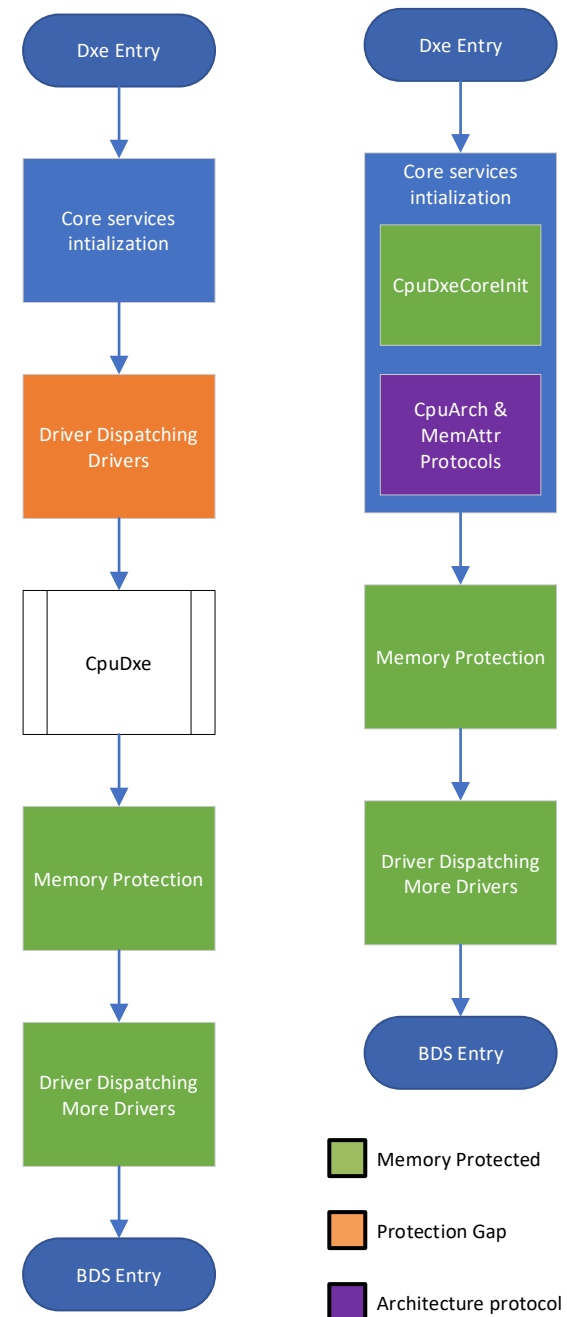
Proposal:

1. Converting current CpuDxe into DXE_CORE libraries

- Flatten driver specific dependencies (ArmGic, etc.)
- Host CPU Arch and Mem Attr protocols
- Memory attribute update go through GCD updates

2. Centralize the CpuDxe initialization implementation (more in next page)

- Create *GcdMemorySpaceLib* for common GCD operations
- Implement *CpuDxeCoreInit* common routine
- Define interfaces and port arch specific operations



“Combining” CpuDxe(s)

Proposal:

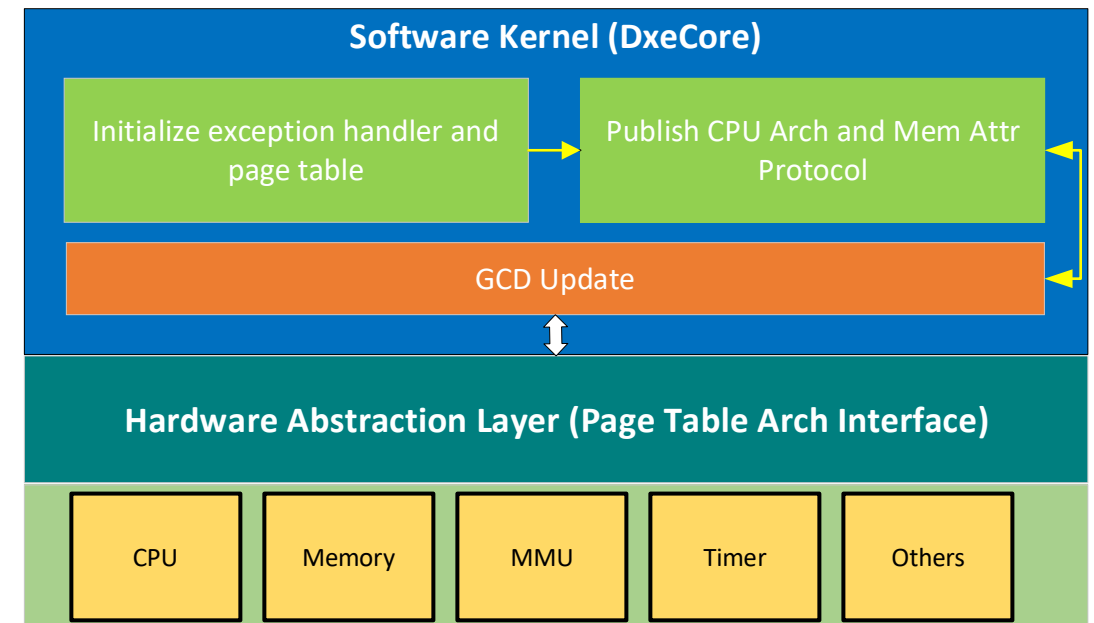
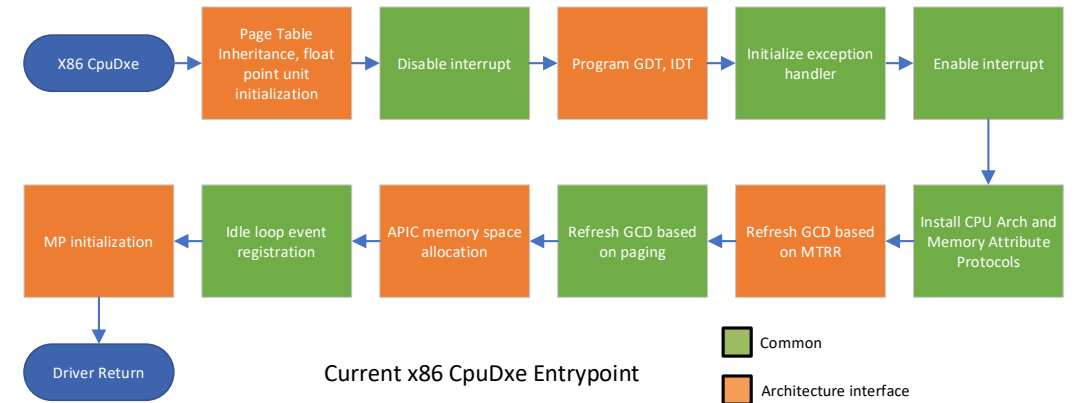
1. Restructure CpuDxe to a common routine
2. Define CpuDxeCoreFeatureLib for arch specific functions
3. Centralize CPU arch and memory attribute protocols to be hosted in DXE Core
4. Update CPU arch and memory attribute protocol to go through GCD update routine

Pros:

- Common routine for essential logic (exception handler and GCD to paging synchronization)
- Streamlined memory attribute call with GCD update

Cons:

- Extra work and slightly altered flow

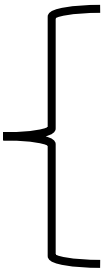


Appendix

Minimal Protection Definitions

Protection Name	Definition
Null Pointer Detection	<ul style="list-style-type: none">• EFI_MEMORY_RP applied to the 4K page at the NULL address
Stack Guard	<ul style="list-style-type: none">• EFI_MEMORY_RP applied to the base of the stack to catch overflow• EFI_MEMORY_XP applied to the entire stack to prevent execution
Page Guard	<ul style="list-style-type: none">• Allocated pages flanked by pages marked with EFI_MEMORY_RP
Pool Guard	<ul style="list-style-type: none">• Final byte of an allocated pool is aligned to the first byte of a 4K page marked as EFI_MEMORY_RP
Image Protection	<ul style="list-style-type: none">• Data and code sections are separated• Data sections have the attribute EFI_MEMORY_XP and code sections have EFI_MEMORY_RO

Heap Guard



More “Detailed System Expectations”

Microsoft requests that all UEFI binary images (e.g. UEFI drivers such as GOP, UEFI applications such as manufacturing tools, etc.) meet these requirements for compatibility in firmware with memory protections enabled:

1. Do not read/write to NULL/Page 0
2. Separate image code and data sections
3. Page align image sections (4KB)
4. Allocate data into data memory types and code into code memory types
5. Do not load code images from code distributed as UEFI binaries
 - If this is necessary, we would be interested in discussing how to best handle code memory buffers allocated by the binary module code
6. Stay within the boundaries of allocated memory buffers
7. Avoid stack overflow and underflow
8. Do not execute from the stack
9. Set the /NXCOMPAT DLL characteristic
10. Do not expect all memory to be mapped (memory could be identity mapped but sparse)