Integrity Protection Analysis of OS Pre-boot

UEFI Security Sub-team
Primary Authors
T. Varugis Kurien – Microsoft
Vincent Zimmer – Intel

Version 0.1
January 17, 2007

Revision History

Revision	Revision History	Date
0.1	First draft version for SST	1/17/07

Contents

Introduction

Hardware evolution leads to increasing complexity in system board and peripheral design. The BIOS, now superseded by the Unified Extensible Firmware Interface (UEFI), [#UEFI_REF] evolved to insulate OS's from hardware specific details while allowing for hardware agility. In contrast to monolithic BIOSes, the UEFI framework allows hardware agility via software extensibility by exposing a driver model. Extensibility points can serve as a point of attack for malware – such as a recent exploit targeting ACPI [HEA] or SMM by Duflot et. al. [DEG]. This type of malware is essentially undetectable by OS hosted protection technologies, such as anti-virus since it executes well before the OS. Consequently, the security integrity foundation of the pre-OS boot environment provided by UEFI and other pre-OS extensibility must be solid, while still providing enough flexibility to support hardware agility. Consequently, we provide an integrity analysis of systems with an UEFI pre-OS environment. Specifically, this white paper defines integrity protection business goals, provides a security model within which these goals are expressed as security requirements and identifies architectural and implementation issues that cause the requirements not to be met. We expect to address these issues in the UEFI SST.

UEFI Overview

"UEFI" is an industry group that is standardizing what was the EFI1.10 specification and the Framework PEI and DXE specifications. Within UEFI, the main UEFI specification (beginning with UEFI2.0) is handled by the UEFI Specification Working Group (USWG) and the Platform Initialization (PI) Architecture PEI/DXE content are handled in the Platform Initialization Working Group (PIWG). The differences are that USWG is focused on the OSto-platform interfaces, whereas the PIWG is focused upon platform initialization. The USWG parties of interest are the OS vendors, pre-OS application writers, and independent hardware vendors who write boot ROM's for block or output console devices. The PIWG parties of focus include the platform builders (MNC's/OEMs), chipset vendors, and CPU vendors. The PIWG work can accommodate both UEFI operating systems and today's conventional/Int19h-based OS's.

PIWG-based components provide one means of producing the UEFI interfaces. What is common across both UEFI and PI Arch is the extensibility of code and interfaces. For UEFI and PI DXE, the extensibility point is a loadable driver model and for PI PEI extensibility is through firmware files with PEI Modules.

Integrity Goals

The system diagram below indicates that the pre-OS can be divided into two parts, an OEM controlled platform part for which extensibility is not necessarily a consideration (with the exception of upgrades and fixes) and a non-OEM controlled extensible portion of the pre-OS boot. The high level integrity protection goals that we have set are:

1. Since uncontrolled extensibility can damage the integrity of the platform and consequently compromise the integrity of all kernels booting on the platform, the OEM controlled platform part should be isolated from the non-OEM controlled preOS (extensible) part.

2. Since both the OEM controlled platform part and the non-OEM controlled parts can compromise the integrity of a booting OS, they must be capable of being configured by a platform administrator in a manner that there is at least one configuration of pre-OS code and data where investments made in protecting the integrity of the OS/other security kernel (like a hypervisor) are not devalued by extensibility in the non-OEM controlled pre-OS. Call the set of such states *secure configuration states.* The boot of an OS in a secure configuration state is called *secure boot* of the OS or the security kernel. Any configuration state must be security auditable – eg. by a TPM.
    a. In the event that the pre-OS security configuration state is not a secure state as defined above, then this information may be used for access control, as in divulging sealed storage keys/network access.
    b. For security bugs in either of the OEM controlled part or the non-OEM controlled pre-OS, resulting in an insecure security configuration state, the platform administrator should be able to patch the system with assurance that the platform cannot be returned to its earlier insecure configuration.
    c. For ease of administration, the security policy that describes *all* secure configuration state must be as succinct as possible – for example defined by cryptographic signing policy and not by directly naming all valid hashes of secure configuration states. DRM and network access control scenarios dictate that the current security policy of the system must be in an audit log that is available on request.

The remainder of the paper is as follows:

1. We define the integrity protection security model (compartments) and re-write the integrity goals as formal security requirements.
2. We carry out an integrity analysis using the model and point out architectural and implementation issues that lead to violations of the security requirements.
3. Possible solutions to major architectural issues are proposed.

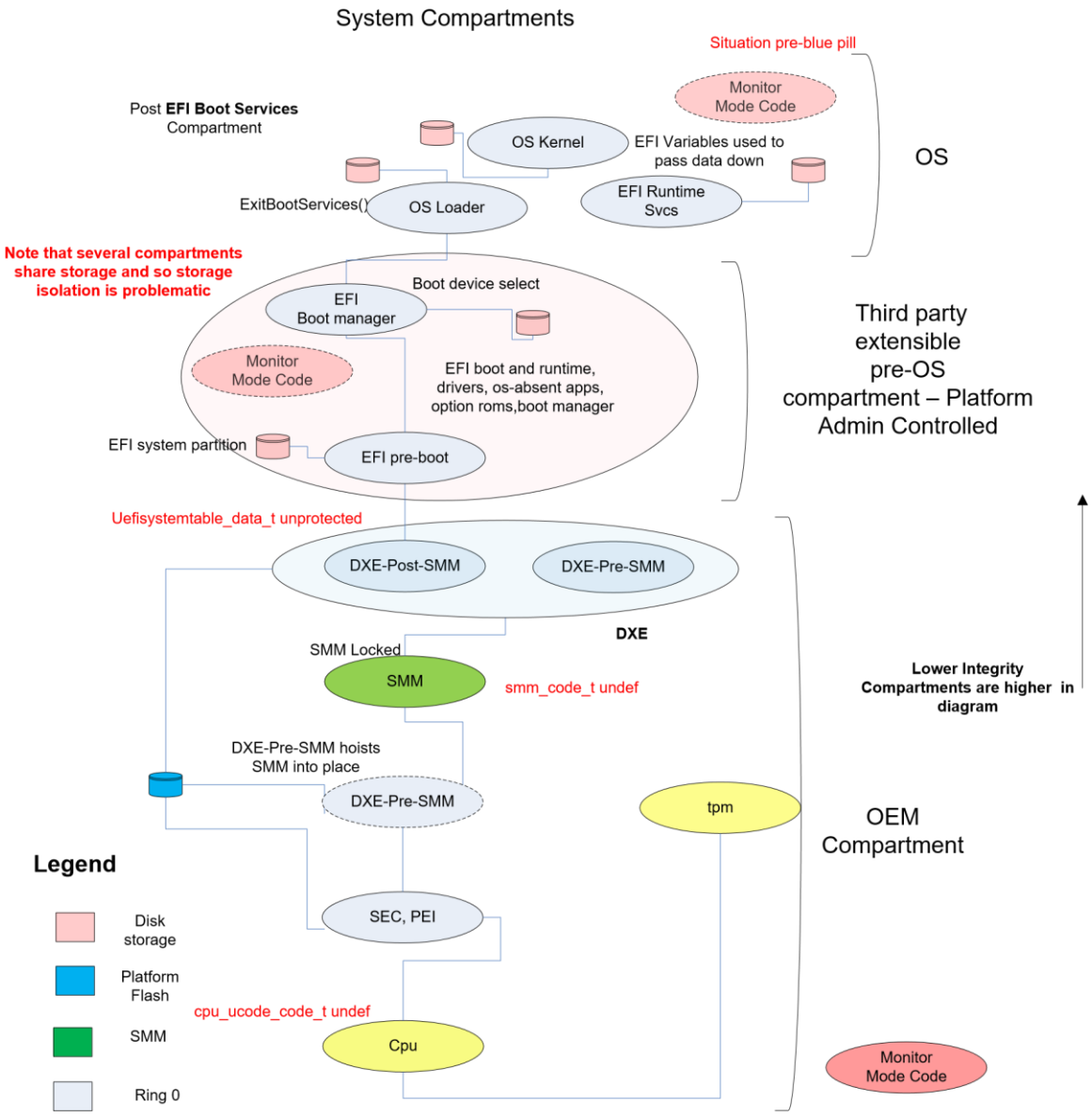


Figure 1: System compartments

Compartments

In order to perform a more formal security analysis, we use the concept of an *integrity compartment.* Formally, a compartment is an integrity protected isolation boundary holding code and data that obeys the Clark-Wilson integrity rules. An informal definition of the Clark-Wilson rules for compartments is below:

1. **Guards**: The entry of all code and data into the compartment is **guarded.** This means that

    - Install of all code and data has to be verified by a guard (such as a code signing guard). This includes initial install and upgrade of code/data. Signed volume updates provide such a mechanism and UEFI ECR 47/48 provide for Protected Variables to store signing keys used for integrity verification of code or data capsules being installed in a compartment.

    - Transitions of code/data into the compartment, such as RPC interfaces exposed by the compartment, must be validated by a guard in the compartment. In general, this means things like bounds checking/type safety as well as code development practices like the SDL [SDL]

    - If the compartment has UI, then the UI transition into and out of the compartment must be guarded by a Trusted Path – that is the user (usually platform administrator) has to have assurance that he is using the UI for that compartment and not some other compartment. In the case of pre-OS, the OEM compartment should not have any UI and platform administrative tasks must be absolutely minimized.

2. **Verifiable Integrity of Code and Data in the Compartment:** This means that the integrity of all code and data in the compartment must be verifiable at any time in the lifetime of the compartment. In general, this implies that the compartment needs memory (execution) isolation as well as storage isolated from other compartments. Having storage isolation implies that when a thread of execution within the compartment writes data, that the data is automatically protected by the compartment. Code signing where code signatures are checked at run time is not equivalent to storage isolation since the integrity of other data, such as security configuration settings is not checked.

3. **Admin:** The admin model for the compartment has to be fully defined in the sense that the tasks that the compartment admin can do have to be specified. Furthermore, the admin needs to be protected from other lower integrity compartments when he is performing admin actions. This means that the designer should absolutely minimize admin actions that can compromise the integrity of the compartment. For example, if the admin sets the security policy for a guard that authorizes code or data into a compartment insecurely, he should not expect that the system can be subsequently booted in a secure state even if the policy was reset to be secure. In the case of Protected Variables, the principal *platform admin* admin action was that of zeroing the platform key so as to put the system into *owner* mode – which should be done by asserting remote or local physical presence. After that, the platform admin could reset a Protected Variable. Since remote reset of a Protected Variable does not need UI, remote platform admin operations on Protected Variables are more securable than local platform administrative actions.

4. **Audit:** All integrity affecting operations including administrative operations in the compartment have to be audited. TPM Quote and Extend functions form the basis of

audit in pre-OS. The availability of the audit log is also a critical issue and requires storage isolation.

The security requirements below formalize the business requirements stated earlier.

Security Requirements

The security requirements are the following:

1. OEM-CPT-PROTECT: OEM provided components of the platform must be a compartment separated from the non-OEM provided pre-OS part of the platform. The OEM compartment is administered by an OEM administrator role and this role does not need interactive logon to the system.
2. NON-OEM-PREOS-PROTECT: Non-OEM provided (extensible) pre-OS part of the platform should be a compartment separate from the OS. This compartment is under the control of a platform admin role which may need to be entered by an interactive logon. It is recommended that physical presence be at least one way of authenticating the platform administrator.
3. NON-OEM-PREOS-SECURE-POLICY: There exists at least one set of security policy for configuring the non-OEM pre-OS compartment such that in such *secure configurations* the non-OEM pre-OS compartment does not damage the integrity of an OS booting on the system. This policy must be specified in a manner such that
    a. POLICY-MDL: The policy has minimal description length while capturing all *secure configurations* of the compartment. This is an administration requirement – for example, if the policy uses code or data hashes to specify secure configurations, a policy that uses digital signing has shorter description length and does not change as often as the former policy. This means that the platform admin does not need to administer the system often, reducing the TCO.
    b. POLICY-REVOKE: The policy must be able to revoke compartment configurations with security bugs and transition to a secure configuration based on platform administrator input. In practice, this means that if there was a code/data bug in the compartment, the policy should allow for patching the bug without intervention by the platform administrator to change the policy. For example, if patching policy was signature based then new patches could be installed without administrative intervention. On the other hand if the policy was based on hashes, the patch could not be installed without a policy change – adding the new hash(es) to the list of allowed hashes.
    c. POLICY-AUDIT: The security policies (including policy for authenticating the admin) guarding the OEM compartment as well as the non-OEM pre-OS compartment must be auditable. Note that the succinctness requirement POLICY-MDL means that the audit log will change rarely and only in case of a change in the (succinct) policy represented by the POLICY-MDL requirement.

Since memory isolation is well understood, we discuss the security requirements for storage isolation.

Storage Isolation Requirements for compartments

The diagram below is an idealized representation of storage for a compartment. Note that all read/write operations for a thread of execution within the compartment happen through a chokepoint that directs IO to a per-compartment integrity protected storage abstraction.

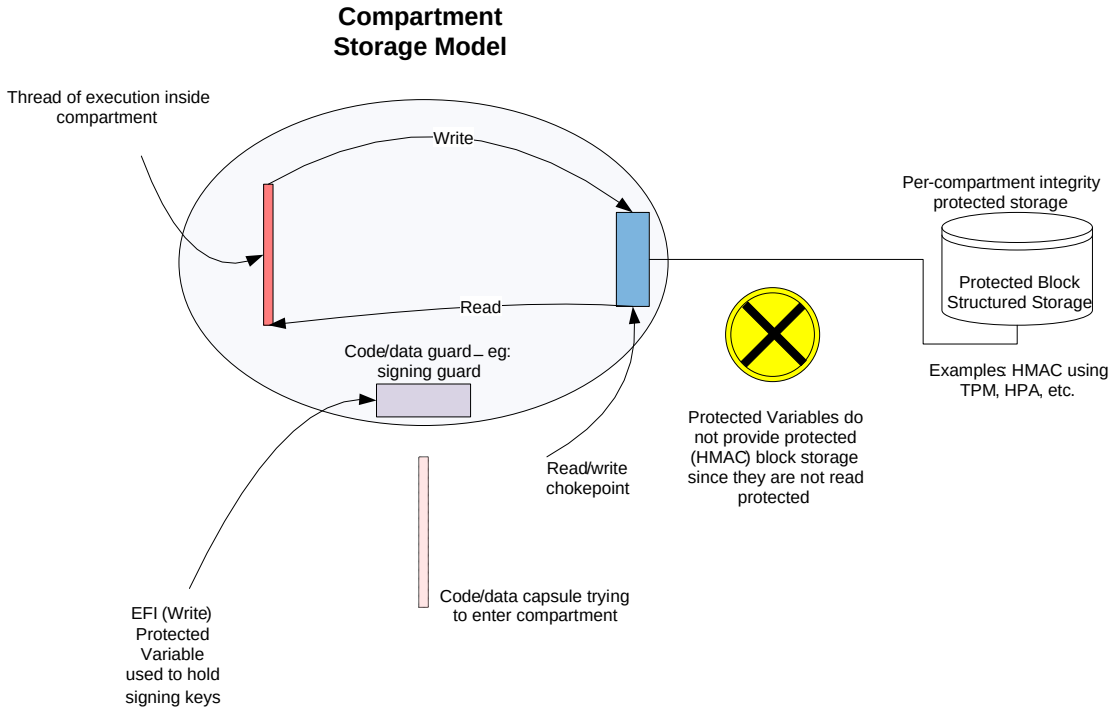


Figure 0-1

The detailed security requirements for compartment storage are below:

1. **Per-compartment storage is "access controlled for write" to the compartment (CPT-ACL):** Only the compartment should be able to write to the storage for the compartment
2. **Compartment admin should be able to write to the storage (ADMIN-ACCESS):** Since the compartment admin may need to modify policy data being consumed within the compartment, he needs to be able to access storage for the compartment. There are two access models here:
    a. **Directly modify storage:** In this case, the admin can directly modify data on the compartment storage even if the compartment is offline. This is not recommended for pre-OS compartments since it makes validating the storage hard when the compartment boots next.
    b. **Post a request for data modification:** In this case, the admin can post a request for modifying data and this request is handled and written to compartment storage when the compartment next becomes active. The UEFI Signed Capsule Update mechanism follows this model. We believe that this model is preferable for pre-OS due to low implementation complexity.
3. **Storage must be available for use (AVAIL):** It is hard to predict the security behavior of threads of execution within the compartment if storage is not available. For example, deleting configuration data for a compartment may cause the compartment to fail into a state where it is possible to arbitrarily configure it - like

deleting the Platform Key that gates transition into Owner mode in UEFI. Note that TPM Sealed Storage does not satisfy this requirement.

Note that block locking flash satisfies all the requirements above. Sealed storage satisfies CPT-ACL and ADMIN-ACL but not AVAIL – in fact deleting a sealed storage key makes the storage for a compartment permanently inaccessible.

The next section discusses the method that we use to analyze compartment protections by looking for inter-compartment information flows that fail the Clark-Wilson rules.

Compartment Protection Analysis Overview

Figure 1 is a pictorial representation of components in pre-OS and OS compartments. For example, the OEM compartment contains the PEI, DXE and SMM modules. Think of these modules as security types in a type system. Therefore, the OEM compartment contains the PEI, DXE and SMM security types. The non-OEM pre-OS compartment contains security types representing UEFI drivers and applications, runtime services etc. The Clark-Wilson analysis looks for *unguarded writes* from security types in the non-OEM pre-OS compartment to security types in the OEM compartment in order to see if the OEM-CPTPROTECT requirement is satisfied. A similar analysis is carried out for finding unguarded writes from security types in the OS compartment to the non-OEM pre-OS so as to see if the NON-OEM-PREOS-PROTECT requirement is satisfied. Note that by transitivity, an unguarded write from the OS compartment to the non-OEM pre-OS and thence to the OEM compartment results in the OS being able to affect the integrity of the OEM compartment.

Looking more closely at Figure 1, some obvious violations of the security requirements are below:

1. SMM security data type is undefined: The SMM code data type in the OEM compartment is undefined since there is no defined integrity verification procedure for the code in the SMM. This means that the set of OEM modules do not form a compartment. Code signing of code in the SMM using signed updates from the OEM fixes this problem.
2. The DXE provided UEFI System Table type in the OEM compartment is not memory isolated from UEFI drivers in the pre-OS non-OEM compartment. This is a problem that requires architectural consideration.
3. None of the security data types in the non-OEM pre-OS are storage isolated from the security data types in the OS since they share disk storage on the UEFI system partition.

By transitivity, there are write flows from the OS to modules in the OEM modules collection. The faults above are architectural in nature.

Some design motifs for compartments in EFI are shown in the table below:

Compartment Design Motifs

Compartment Requirement	Design Motifs	Comments	Implementation atoms
Identification and protection of code entering compartment	Code signing verified at install time (signed capsule update)/run time (ECR 48)	Install time protection requires protected storage for compartment. Install time (fail early) provides better overall experience and security	Code signing + block locking flash
Identification and protection of data in a compartment	As part of install process and needs compartment protected storage	All programs within the compartment should have data protection by design. **	Blocking locking flash, HPA
Inter compartment memory separation	Temporal separation of compartments, memory protection domains	Install of code and data into compartment should be unprivileged	Temporal separation, SMM hosting
Protection of compartment admin	Minimize admin tasks– ideally a one time "take ownership" task	Want a "no UI " but remoteable admin	TCG ownership protocol (local and remote)
Protecting compartment boundary against other compartments	Identify inter- compartment call points (incl. call interfaces) and need to figure out how to protect	Inter compartment IPC points need armoring, inter compartment data transitions need armoring (including data handoff like the HOB)	Type safety, SDL as part of dev process
Audit	Use tpm to quote security ops for compartments	Not guaranteed available audit log	Tpm, block locking flash, 21 HPA

Table 1 – Design motifs

The next section of the white paper does a detailed analysis of information flows across the putative compartments when a system goes through power transitions (S4/S5 and S3).

Security Analysis

The analysis method consists of identifying security relevant data types in the 3 putative compartments, namely OEM pre-OS, non-OEM pre-OS and OS and using Clark-Wilson information flow rules to check to see if there are unguarded writes from security types in one compartment to security types in a lower integrity compartment. Before we go further, we need to introduce some notation:

Notation

1. *foo_t*: Denotes a security type named *foo.* Eg: *smm_t* represents the SMM type and can be used to denote either a code or data type (below).
    a. *foo_code_t*: Represents a code object type named *foo.*
    b. *foo_data_t:* Represents a data object named *foo.* For example, *efi_system_table_data_t.* We make the distinction since in general, code objects tend to be protected differently from data objects – eg: by digital signatures.

2. *foo_r:* Denotes a security role named *foo.* For example, *platform_admin_r* represents the platform admin and *oem_admin_r* is a role representing the OEM. The security requirements dictate that the pre-OS OEM compartment is under the control of the *oem_admin_r* and the pre-OS non-OEM compartment is supposed to be under the control of the *platform_admin_r.*

Integrity Analysis

In order to put the security analysis into practice, a representative system based upon UEFI Platform Initialization (PI) Architecture [PI] elements will be decomposed using the above listed types and compartment taxonomy.

Strictly speaking, the UEFI Main Specification does not have to be built on PI elements, such as a firmware substrate containing the Driver Execution Environment (DXE) and Pre-EFI Initialization (PEI). DXE and PEI are used in this discussion, though, since they are publicly described implementations of EFI based pre-OS environments. The analysis conducted below shows how to reason about the integrity of pre-boot using the compartment model.

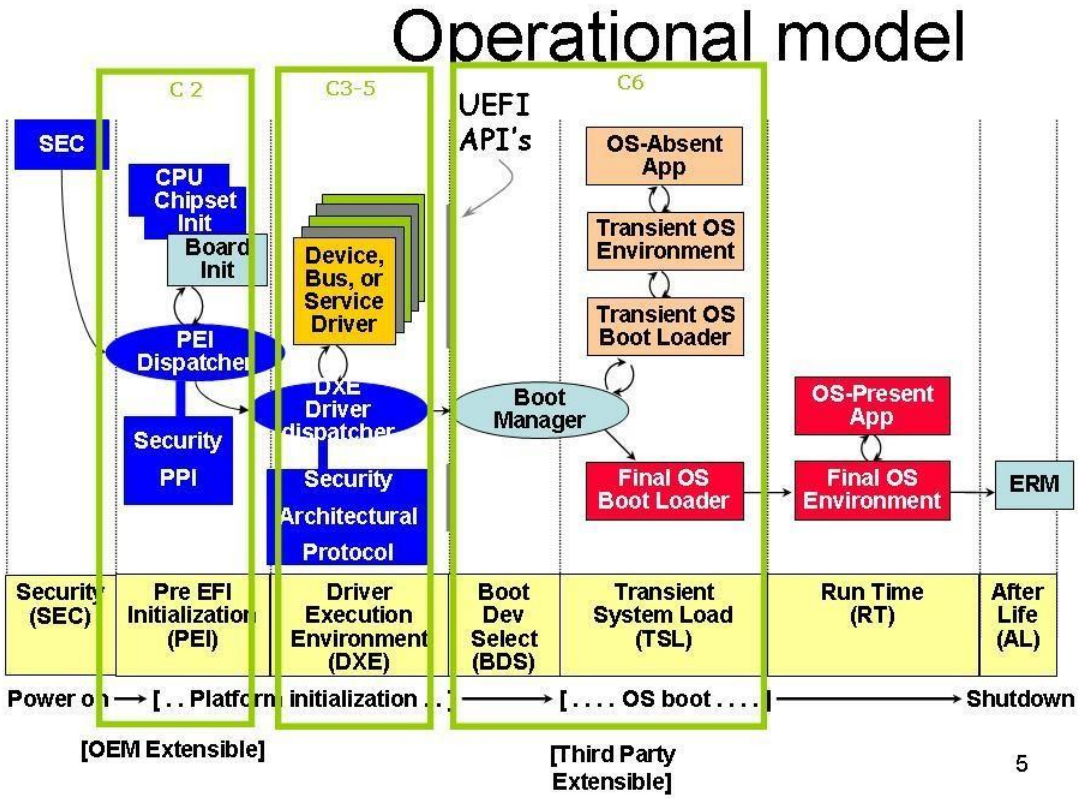


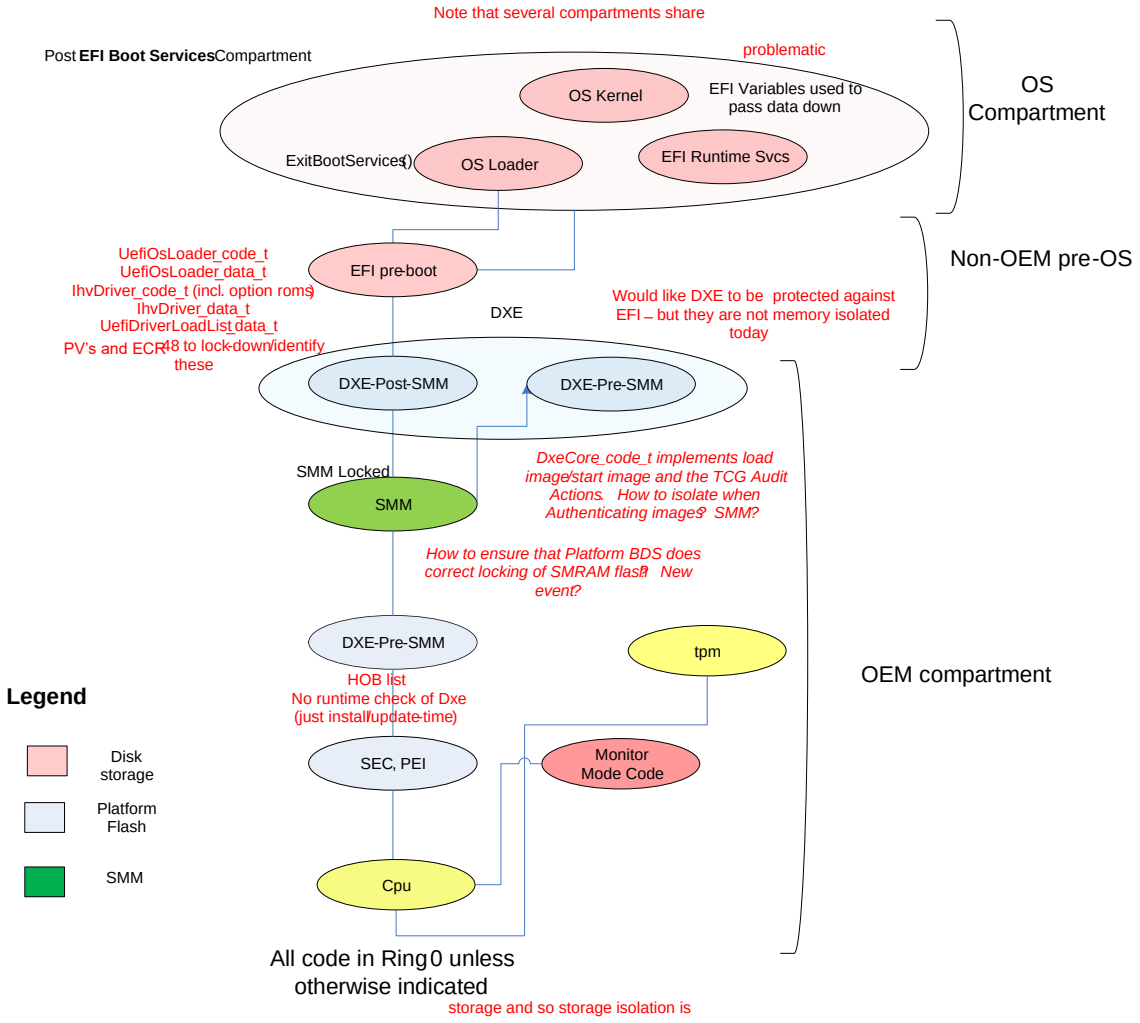
Figure 3: System compartments

The modeling methodology is to take the generic compartment figures from earlier, name them according to the PI Architecture and UEFI elements, and ascribe some prototypical code types, data types, and roles to each compartment. The types will be colored in RED if there is a Clark-Wilson integrity violation noted. Mitigation of those violations will either be via ECR47, 48 or future work items.

A representative list of components will be provided in the appendix.

Integrity Analysis Case 1 – Cold Boot

This flow is the traditional machine restart from a mechanical power-on. This flow does not include the addition of ECR47,48 but is instead intended to describe the canonical flow.



Normal boot

Figure 4: Cold boot/S5

Compartment /protection	Type Name & issue	Proposed fix	Comment

C3-C5 / OEMCPTPROTECT	The UefiSystemTable_data_t is created by code in C1-5 (SEC through post-SMM DXE), but it exposed to OEMextensible code in C6 coequally in ring0. Services like LoadImage(), AllocatePool(), and other Boot and Runtime services are exposed in the System Table data table for possible corruption and usurpation.	Possible mitigation would be to run the 3rd party extensions in EFI Byte Code (EBC), a simple instruction set that is managed by the EbcInterpreter_code_t. EbcInterpreter_code_t can perform software-fault isolation (SFI) wherein the interpreter will ensure that the UefiSystemTable_data_t is not corrupted. Another option would be to de-privilege the the 3rd party extensions into Ring 3 (namely C6), keeping C1-5 in Ring0. Alternately, moving the C1-5 into Ring -1, via technology such as a virtualization hardware support, may allow for protecting these data structures and other code/data pages that produce the boot services. Another option would be to require that all 3rd party extensions are signed and authorized.	Some of these solutions would read on material in the UEFI specification, so become in-scope for the USWG. Other items may read on platform initialization flows, namely changes in PEI and/or DXE components, so they would read on the platform initialization working group.
NON-OEM-PREOSPROTECT	UefiOsLoader_code_t, UefiOsLoader_data_t, UefiDriver_code_t, UefiDriver_data_t	Authenticated Variables for code/data	

Table 2 – Issues w/ types on a normal boot

Integrity Analysis Case 2 – S3 Resume

S3 resume is a power-management state wherein memory contents are preserved in a self-refresh manner with the DRAM complex but the main CPU is powered down. More information on S3 can be found in the ACPI3.0 specification at [ACPISpec]. The S3 resume is a simple flow in that some implementations of PEI Architecture-

based firmware only execute the SEC and PEI phases in order to restore the machine state. As such, this flow exemplifies the value of the modeling.

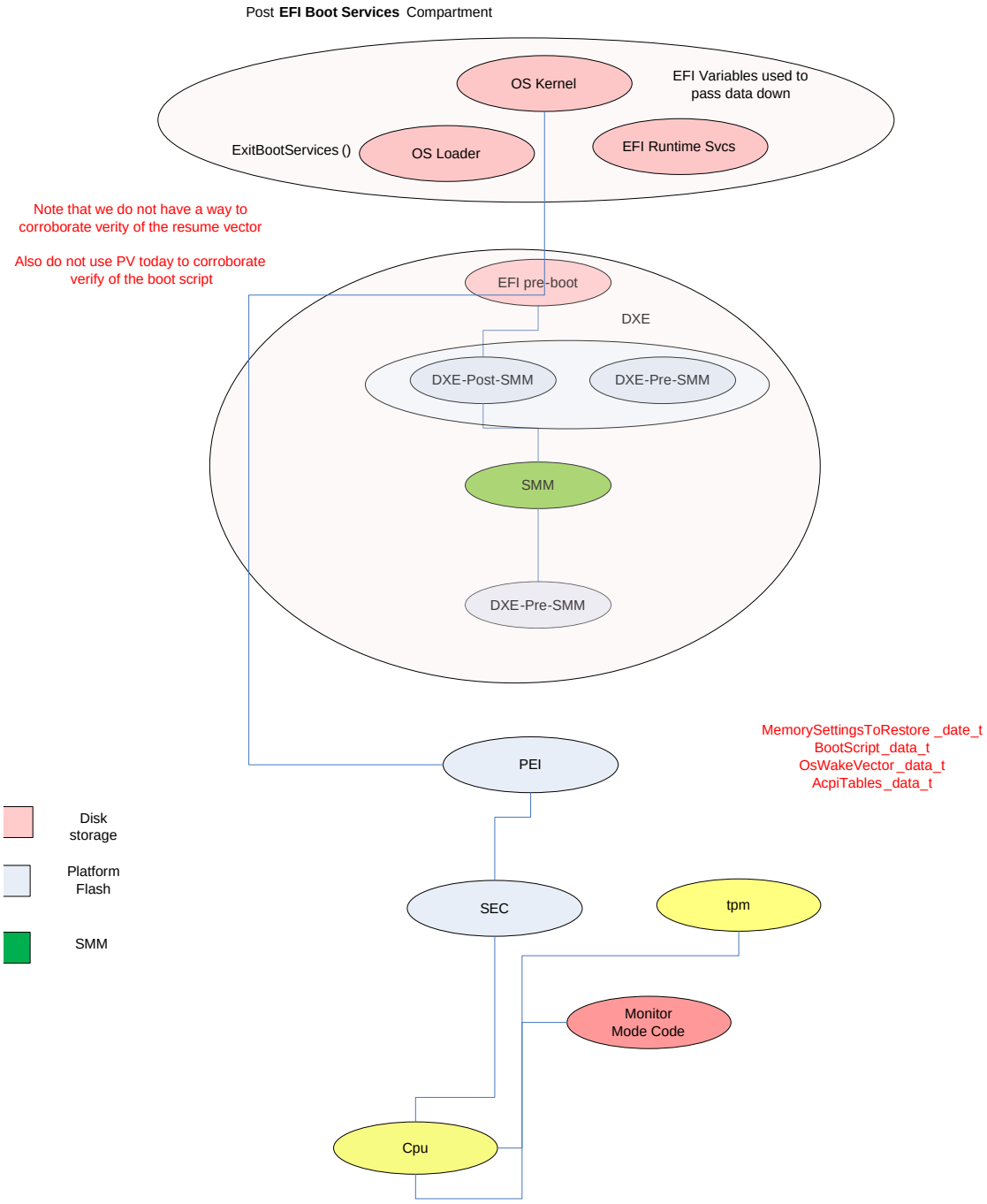


Figure 5: S3 restart

Possible mitigation is to use a Protected Variable for the storage.

Compartment /protection	Type Name & issue	Proposed fix	Comment
OEM-CPTPROTECT	OEM-CPT-PROTECT – has issues with the MemorySettingsToRestore_data_t and the BootScript_data_t. Today, vendors may choose to use storage not ACL'd to the OEM compartment.	Possible mitigation would be to use authenticated variables for storage of the data types.	This is platform implementation flow, not an exposed UEFI behavior. Shows value of the modeling.
NON-OEM-PREOSPROTECT	OsWake_vector_data_t can be modified by OS runtime (C6).	Firmware to record value of in an authenticated variable.	This analysis explicitly avoiding discussion of ACPI vulnerabilities.

Table 3 – Issues with types on an S3 restart

Integrity Analysis Case 3 – Capsule Update

This is an important flow to model since it is one means by which to effect the OSfirmware key exchange desribed in ECR48.

From an admin model perspective, the update of the SEC – DXE compartments is managed by the OemAdmin_r. Typically systems support field upgrades of the base firmware that meets the original equipment manufacturer's criteria. This OemAdmin_r-based update helps to maintain OEM-CPT-PROTECT.

ECR47/48, though, are aimed at managing the NON-OEM-PREOS-SECURE-POLICY and would be a PlatformAdmin_r, which is analogous (or may be the same as) the TPM owner.

For CW-VALIDATE-INTEGRITY, we have an install-time check.
For the Capsule update, there is an IVP in that allows code into the OEM-extensible compartments, namely C1 (SEC) through C5 (post-SMM DXE) is the CapsuleUpdateAndVerifyPeim_code_t.

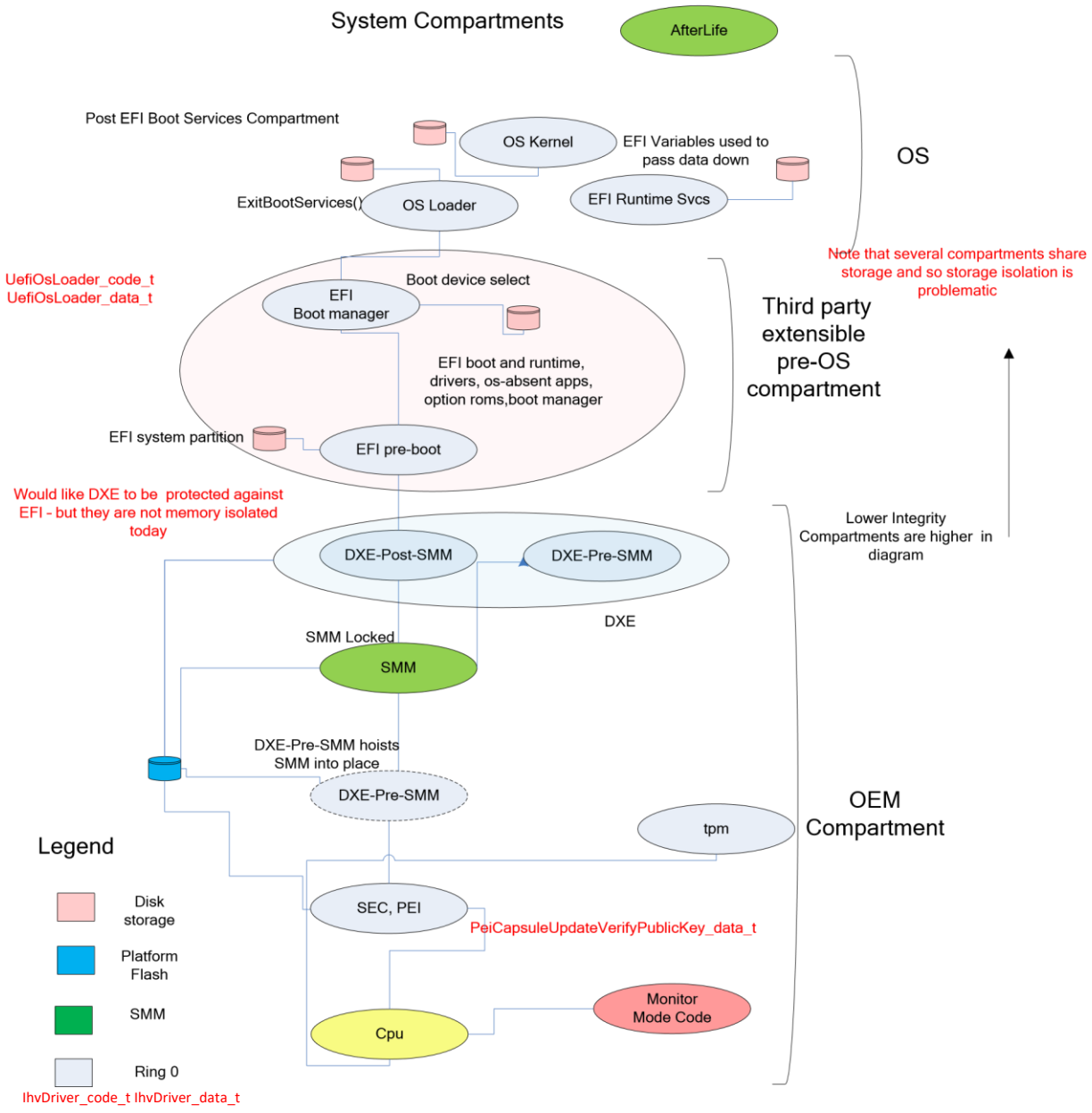

System Compartments
AfterLife
Post EFI Boot Services Compartment
OS Kernel
EFI Variables used to pass data down
OS
ExitBootServices()
OS Loader
EFI Runtime Svcs
Note that several compartments share storage and so storage isolation is problematic
UefiOsLoader_code_t
UefiOsLoader_data_t
Boot device select
EFI Boot manager
Third party extensible pre-OS compartment
EFI boot and runtime, drivers, os-absent apps, option roms,boot manager
EFI system partition
EFI pre-boot
Would like DXE to be protected against EFI - but they are not memory isolated today
Lower Integrity Compartments are higher in diagram
DXE-Post-SMM
DXE-Pre-SMM
DXE
SMM Locked
SMM
DXE-Pre-SMM hoists SMM into place
DXE-Pre-SMM
tpm
OEM Compartment
Legend
Disk storage
Platform Flash
SMM
Ring 0
SEC, PEI
PeiCapsuleUpdateVerifyPublicKey_data_t
Monitor Mode Code
Cpu
IhvDriver_code_t IhvDriver_data_t

Figure 6: Capsule Update

Compartment /protection	Type Name & issue	Proposed fix	Comment
OEM-CPTPROTECT	OEM-CPT-PROTECT – has issues with the PeiCapsuleUpdateVerifyPublicKey_data_t. Today stored in flash and use block-locking on the SPI flash or FWH to protection. Still vulnerable to flash-swap/mod-chip attack.	Seal the value agains the TPM or store in TPM-Nvdata.	

Table 4 – Issues with types on a capsule update

Conclusions

This analysis has shown that the evolution of new capabilities for platform system software, especially under the guise of security, requires a broader approach. This approach needs a more formal context, such as a security model, through which the goals of the various business interests can be expressed. As the UEFI SST, we should drive these security requirements through sister working groups in UEFI and/or PIWG in order to ensure that the requirements are met.

Appendix 1: Integrity Protection Models

This section provides an introduction to the Biba security model and the Clark-Wilson information flow rules that we'l be using in the rest of the paper. The intent is to educate readers who are not intimately familiar with these models.

Biba Model

The Biba model is built around the notion that there is a set of (ordered) integrity levels and sets up two rules:

1. No-Write-Up – this means that lower integrity subjects (process/thread) can't write over higher integrity objects to destroy their integrity.
2. No-Read-Down – this means that high integrity subjects cannot read low integrity data. Cases where a high subject *executes-data* of a low integrity subject is also considered *read-down* for the purposes of this paper.

The problem with the Biba model is that the rules are very stringent. For example, reading data off the network for upgrade can lead to a violation of the rules. Instead of this, the Clark-Wilson rules relax Biba rules in a controlled manner – (CW-GUARD in Table 1) by saying that higher integrity subjects can read lower integrity data but must do so through the services of a guard – a data validation mechanism.

Clark-Wilson Information Flow Rules

In contrast to the Biba model, the Clark-Wilson information flow rules specify restrictions on information flows between pairs of integrity levels such as the OEM compartment and the non-OEM pre-OS compartment. This is not a limitation because information flows are transitive.

Definitions

1. **Constrained Data Items**: The Clark-Wilson information flow rules speak to information flow between a pair integrity levels, since one integrity level is of higher integrity than another, Clark-Wilson uses the term "Constrained Data Item" (CDI) to refer to objects or subjects (processes/threads) of the higher integrity level and "Unconstrained Data Item" (UDI) to refer to objects of the lower integrity level. Since there are multiple integrity levels the term "Constrained/Unconstrained" should be read relative to two different integrity levels. For example, data and code in the OEM compartment is a CDI relative to the non-OEM pre-OS compartment or the OS compartment.
2. **Integrity Validation Procedure** (IVP) or **Integrity Verification Guard**: An IVP/Guard checks to see if a CDI has the specified degree of integrity. Some examples of IVP are below.

a) **Code Integrity maintenance Guard:** An example IVP that checks the integrity of code is digital signing of the code. The set of valid root keys is policy data for the guard and this policy may be edited by the admin.
b) **Data Integrity maintenance Guard:** Data HMAC in transit, a guard that protects against direct modification of a physical resource for data – such as dropping direct writes to a volume with an active file system on it.

IVPs are guards for information flows between differing integrity levels. Guards allow for the controlled relaxation of the Biba No-Write-Up/No-Read-Down rules across integrity levels of the system. The main reason why guards are important is that there is no protection within an integrity level. The guard provides separation across integrity levels and so it is the guard that has to be validated extensively in the analysis of the system. On the other hand the rest of the code running in the high level does not have to be validated, since the guard is the only way by which low information can enter a higher integrity level. Therefore, verification and penetration test should focus on the guard.

3. **Subjects**: Subjects are active entities with an associated security context like processes/threads/virtual machines etc. A user authenticates as a subject by providing a credential. The subject security context in Windows consists of a list of SIDs and Windows Privileges.
4. **Transformation Procedures** (TP): A Transformation Procedure is an *integrity preserving* mechanism by which a subject transforms data in the system (read/write/copy/delete/….) which are sequences of atomic actions in the system. Transformation Procedures are used to implement IVP. Since a TP is integrity preserving, applying a TP to a CDI will preserve the integrity of the CDI.

Clark-Wilson Information Flow Rules

Rule	Rule Specification	What does it mean?

CW-VALIDATEINTEGRITY INTEGRITY MUST BE VALIDATABLE	IVP's must be available on the system for validating the integrity of any CDI.	Any piece of code/data that claims to be at some integrity level must be verified to have it before being consumed at that level. For example, kernel mode code signing is an implementation of this rule for integrity validation of kernel mode code. There are two ways to ensure that a CDI (integrity protected item) remains integrity protected: a) Access Control – make sure that the item cannot be modified at all by items of lower integrity. Set an ACL on the item – for example, the data for the OEM compartment can be stored in block-locked flash which is not writable by modules in the nonOEM pre-OS. b) Integrity protect the item, perhaps by hashing or signing or sealing. Generally, (a) is preferable to (b).

CW-CHOKEPOINT MUST HAVE CHOKEPOINTS FOR INTEGRITY PROTECTED DATA	A CDI can be changed only by a TP.	The only way by which CDI can be modified maintains the integrity of the CDI since only TP are used to edit the CDI and TP maintain the integrity of CDI. If there are ways by which the CDI can be modified, without using a TP then we cannot assure the integrity protection of the high integrity object. For example, if one can install code into the OEM compartment without going through the signed capsule update chokepoint then this rule is violated.
CW-ACCESS-CONTROLINTEGRITYDATA MUST RESTRICT WHO CAN EDIT INTEGRITY PROTECTED DATA	Subjects can only initiate certain TPs on certain CDI – there is a table (*s, t, d*) that specifies the set of TP, *t,* that a subject *s* can perform on a datum *d*	Recall that editing high-integrity data must be done via a TP due to CW-CHOKEPOINT. Since the TP may change the integrity of the system, such as changing the Platform Key, we must restrict the set of subjects (users) that can invoke the TP used to reset the Platform Key. For example, access to the TP used to reset the Platform Key should be gated on being in the *platform_admin* role.

CW-GUARD INTEGRITY UPGRADE DONE THROUGH GUARDS	Guards are TP that act on UDI and produce CDI as output. That is, low integrity data/code can be promoted to high integrity (after it is verified by a guard) using a special TP – the "un-tainting" guard.	Guards are used to validate UDI transitioning across a security boundary. For example when a CDI piece of code accepts data from an untrusted network (UDI), then the data transitions across the UDI->CDI boundary. This data needs to be validated and guards do this. Guards relax the no-read down rule of Biba integrity, so that read-downs are allowed via an integrity guard. The guard is an airlock (a one-way data flow) from a lower integrity level to a higher level. Guards can guard entry into the admin role as well. For example, if the capsule update mechanism is the only mechanism to upgrade the OEM compartment, the signature checking guard prevents install of unsigned capsule updates. **It is very important that the guard must have integrity at least as high as the level that it is protecting. For example, the signing check should be done by code inside the OEM compartment.**
CW-AUDIT MUST AUDIT INTEGRITY AFFECTING OPERATIONS	Each TP application must be audited since TP can affect the integrity of code/data	An example of this being violated in the system today is dification of the audit log post-hoc. Note that TPM Quote and Extend do not provide a protected audit log since the audit log may be deleted in its entirety.
CW-ADMINAUTH MUST AUTHENTICATE ADMINS	System must authenticate subjects attempting to perform a TP	By CW-ACCESS-CONTROL-INTEGRITY-DATA, only certain subjects like the *platform admin* can instantiate TP on protected objects. What this requirement says is that these subjects must be authenticated. We suggest authenticating the *platform admin* based on physical presence.
CW-SECADMIN MUST HAVE SECURITY ADMINS	Only special subjects (security administrators) must be able to change any authorization related lists such as the map (*s, t, d*). Generally this leads to the concept of a per-layer admin.	Since the administrator now executes code at every level, it is very hard to restrict information flow across levels because the administrator provides a natural conduit for such flows. Having the *platform admin* and *oem admin* roles satisfies this requirement in our case.

CW-INTEGRITYPRESERVED TRUSTED PROCEDURES PRESERVE INTEGRITY	Application of a TP to a CDI must preserve integrity of the CDI – the set of CDI are closed under TP actions	This means that a TP does not destroy the integrity of a CDI. How the TP preserves the integrity of a CDI depends, but if a procedure claims to be a TP, then it must be verifiable that the TP maintains the integrity of a CDI it acts on.

Table 5

We stated earlier that the C-W information flow rules were to be used to drive the Integrity Analysis Program.

Integrity Protection Affecting Information Flows

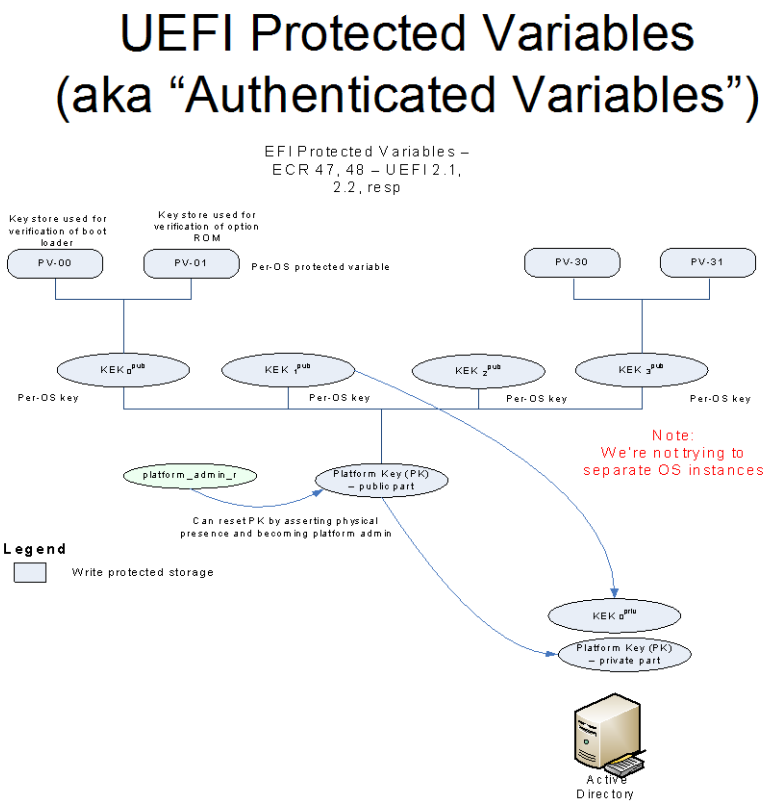
The C-W and Biba rules are rules that say what flows of information can result in integrity compromise. In the rest of this document we divide information flows into the following classes:

1. Write-like information flows: Write like information flows are flows from a starting integrity type to an ending integrity type by which the starting integrity type can write chosen data to an ending integrity type. The C-W and Biba rules seek to ban write like information flows where the starting integrity type is of lower integrity than the ending integrity type.
2. Read-like information flows: Read like information flows are flows from a starting integrity type to an ending integrity type by which the ending integrity type can read chosen data low integrity data from a starting integrity type. The C-W and Biba rules seek to ban un-guarded read like information flows where the starting integrity type is of lower integrity than the ending integrity type.

Our analysis tries to find such flows across the compartments defined in the security requirements.

Appendix 2: UEFI ECR 47, 48 background

This section provides a background on the proposed ECR 47,48 that describe protected variables and the OS-firmware key-exchange/Owner mode, resp.



This ECR introduces “Authenticated” or “Protected Variables” (PV’s). Specificaly, the ECR adds the capability for a platform owner to ensure that variables are only updated in an owner-authorized fashion. This authorization occurs via authentication of the caller

This ECR is a request to add support for authenticated variables to UEFI. This change adds a new behavior to the UEFI Variable interface service set. The change allows for backward compatibility and is deployed via a new attribute with associated metadata.

<Tim on 48>

Appendix 3: Sample code and data types

This section provides some sample code and data types used in the above analysis.

Compartment 1	Type Name + description in compartment	Type Authentication and protection	Audit for compartment

SEC – ideally only OEM extensible	SecCore_code_t - transitions machine from 16-bit real to 32-bit protected mode via ROM'd GDT, hands control to the PeiCore_code_t with address of the boot firmware volume that contains the PeiCore_code_t and other PeiModuleXXX_code_t's	Non-updatable flash or installtime check (sign). Flash protection	Optionally measured into PCR[0].
	CpuUcodePatch_data_t	Intel CPU's have an IVP; patch signed. SEC patches Bootstrap processor (no MP until Compartment 3)	Optionally measured into PCR[1].

Table 6

Compartment 2	**Type Name + description in compartment**	**Type Authentication and protection**	**Audit for compartment**
PEI – ideally only OEM extensible	PeiCore_code_t - discovers PEI modules, maintains the boot mode, orchestrates installation and discovery of PPI's	Install-time check (signed capsule update)	Optionally measured into PCR[0].
	DxeIpl_code_t - responsible for discovering, decompressing, PE load/relocate and pass control to the DxeCore_code_t	Install-time check (signed capsule update). Flash protection (readonly)	Optionally measured into PCR[0].
	OsWake_vector_data_t - location in static ACPI table that the S3ResumePeim_code_t uses to pass control to OS upon S3	Install-time check (signed capsule update). Flash protection (readonly)	Measured into PCR[1]
	Capsule_Fv_data_t - Data in memory during a capsule update. Created by OS runtime agent prior to the Update Capsule UEFI RT call	Signed	Measured into PCR[0]
	HobMemory_location_data_t - Hand-offBlock (HOB) created by the MemoryInitializationPeim_code_t to describe the initialized memory set	In memory. No checksum or integrity metric	Not measured today. PCR[1] may be right place?
	HobFirmwareVolume_data_t - created by the PlatformPeim_code_t in order to inform DxeIpl_code_t where to find the DxeCore_code_t	In memory. No checksum or integrity metric	Measured into PCR[1]

	S3ResumePeim_code_t - orchestrate the restoration of machine state via BootScript_data_t and other programmatic actions (such as restoratoin of MTRR's & SMBUS on all CPU's)	Install-time check (signed capsule update). Flash protection (readonly)	Optionally measured into PCR[0].
	BootMode_data_t - created by the PlatformPeim_code_t to tell system if Flash update, recovery, normal boot, boot w/ no configuration changes, etc.	Managed by PeiCore_code_t	
	MemoryInitializationPeim_code_t - provided by the chipset vendor. Initialized the memory (DDR2, DDR3, etc). Uses Spd_data_t on DIMM's to get geometry	Install-time check (signed capsule update). Flash protection (readonly)	Optionally measured into PCR[0].
	Spd_data_t - Serial presence detect data types on the Dual-Inline memory modules (DIMM's) that reports properties of the memory hardware	No integrity metric	
	MemorySettingsToRestore_data_t - created by the DXE code to allow for a noconfiguration-change boot to simply restore settings.	Implementations could use Protected Variables (didn't prior to UEFI2.1)	
	IchSupportPeim_code_t - abstract capabilities of the south bridge, such as time service, SMBUS, Boot - mode (Sx state on restart)	Install-time check (signed capsule update). Flash protection (readonly)	Optionally measured into PCR[0].
	PlatformPeim_code_t - detects BootMode_data_t, publishes FirmwareVolumeHob_data_t for the main firmware volume, etc	Install-time check (signed capsule update). Flash protection (readonly)	Optionally measured into PCR[0].
	CapsuleUpdateAndVerifyPeim_code_t - PEIM responsible for locating the capsule contents (possibly scattered in phys memory since created during OS runtime) location via the CapsuleHeaderPointerVariable_data_t, creating a contiguous CapsuleUpdateFirmwareVolume_data_t, and authenticating the contents of latter w/ the PeiCapsuleUpdateVerifyPublicKey_data_t Creates a HobFirmwareVolume_data_t with the CapsuleUpdateFirmwareVolume_data_t if authenticated	Install-time check (signed capsule update). Flash protection (readonly)	Optionally measured into PCR[0].
	CapsuleHeaderPointerVariable_data_t - UEFI variable that points to the head of the scatter-gather list of the OS-created capsule.	In UEFI variable. Could be protected	

	Not access controlled since corruption of this is only a DoS (worse-case)-> makes a Capsule Update fail authentication	variable.	
	ReadOnlyVariablePeim_code_t - provides read-only access to UEFI variables	Install-time check (signed capsule update). Flash protection (readonly)	Optionally measured into PCR[0].
	TpmPlatformPeim_code_t - Detect physical presence PhysicalPresenceHob_data_t	Install-time check (signed capsule update). Flash protection (readonly)	Optionally measured into PCR[0].
	Tpm12Peim_code_t - TPM Initialization, Audit invoked by DxeIpl_code_t in order to measure the FvMainInFlash_data_t	Install-time check (signed capsule update). Flash protection (readonly)	Optionally measured into PCR[0].
	HobMeasurementInfo_data_t - Allow for Tpm12Dxe_code_t to create the TCGprescribe 'TCPA' Event log in memory	In memory. No checksum or integrity metric	
	BootScript_data_t - UEFI variable, memory, or other store that contains information on how to restore the settings of the various platform components upon an S3. Consumed by the S3ResumePeim_code_t	Could be a protected variable. Today, just regular UFEI variable or inmemory (ACPI NVS) data structure	
	PeiCapsuleUpdateVerifyPublicKey_data_t - Manufacturing-time provisioned and installtime updated platform key used to verify signed capsule updates. NOT stored in public UEFI variable store	Install-time check (signed capsule update). Flash protection (readonly). TPM NVData. Fuses.	Would not measure. Usu. privacy sensitive data

Table 7

Compartment 3	**Type Name + description in compartment**	**Type Authentication and protection**	**Audit for compartment**

Pre-SMM DXE Phase – ideally only OEM extensible	DxeCore_code_t - platform independent code that is responsible for published the DXE services and UEFI2.x memory-only services, Initial UEFI System Table – UefiSystemTable_data_t Provides a reference to the HOB List in the UEFI System Configuration Table UefiSystemTableConfigurationEntry_data_t	Install-time check (signed capsule update). Flash protection (block-locking). Typically not stored in bootblock but instead in major block of flash.	Measure into PCR[0]
	UefiSystemTable_data_t**Error! Not a valid bookmark self-reference.** - memory structure w/ the UEFI 2.0 boot service and runtime service entries		Measure into PCR[0]
	UefiSystemTableConfigurationEntry_data_t - list of alternate industry tables, such a SAL System Table, ACPI 1.0b/2.0/3.0 tables, SMBIOS 2.5 table, and Dxe Services Table		Measure into PCR[1]
	AcpiTables_data_t - static and dynamic ACPI entries		Measure into PCR[1]
	SmbiosTables_data_t - DMTF SMBIOS static tables		Measure into PCR[1]
	AprioriFile_data_t - lists the drivers that must be dispatched immediately after execution of DxeCore_code_t		Measure into PCR[0]
	DxeCpuDriver_code_t - The Architectural protocols that provide support for the hardware-dependent UEFI services (e.g., Stall, Monotonic Counter) Initialize al of the CPU's and provides the MP services that will be needed by C3 in order to setup SMM on al CPU's		Measure into PCR[0]
	CpuUcodePatch_data_t. Applied to application processors as part of multiprocessor (MP) initialization	Intel CPU's have an IVP; patch signed.	Optionally measured into PCR[1].
	DxeSecurityDriver_code_t - Provides an instance of the Security Architectural Protocol if support policy-based dispatch and/or authenticate the capsule FV contents in DXE		Measure into PCR[0]

	DxeTpmDriver_code_t - Takes the initial measurement HOB from PEI and creates the 'TCPA' event log and creates the measurements of UEFI drivers, GPT, etc pursuant to the EFI TCG specification	Install-time check (signed capsule update). Flash protection (block-locking). Typically not stored in bootblock but instead in major block of flash.	Measure into PCR[0]

Table 8

Compartment 4	**Type Name + description in compartment**	**Type Authentication and protection**	**Audit for compartment**
SMM DXE Phase – ideally only OEM extensible	DxeSmmCore_code_t **Error! Bookmark not defined.** - Takes the SmramHob_data_t and copies instance of the SmmCore into SMRAM Locks SMRAM (i.e., sets D_LOCK) once the	Boot-service code to hoist agents into SMRAM	Measure into PCR[0]
	DXE SMM Drivers are all loaded. Uses PeCoffLoaderEx service in order to load and relocate DXE SMM Drivers into their DRAM location Publishes the EFI_SMM_BASE_PROTOCOL upon which all DXE SMM drivers have a dependency expression		
	SmmCore_code_t - Provides a minimal set of services scoped in SMM, including SMRAM allocation, MP API within SMM, etc Invokes by al CPU's upon receipt of SMI. Rendez's CPU's and runs single-threaded through the handlers		Measure into PCR[0]
	DxeIchDispatcherSmmDriver_code_t - Handles ICH-specific hardware events (GPE trap, TCO time, USB trap, software SMI) and invokes appropriate child handler	SMRAM protection via chipset/cpu	Measure into PCR[0]
	DxeSmmUefiVariableDriver_code_t - Handles actual binary storage of the UEFI variables within SMM. Use synchronous SMI from UEFI runtime "Get/Set" variable API in order to access these services This driver would be updated to support UEFI USWG ECR47 "Authenticated/Protected Variables" for the signed UEFI variables	SMRAM protection via chipset/cpu	Measure into PCR[0]

Table 9

Compartment 5	Type Name + description in compartment	Type Authentication and protection	Audit for compartment
Post-SMM DXE Phase – ideally only OEM extensible	DxeBds_code_t - Implement the platform specific boot-policy Invoke the memory test and put indicator on screen to show progress. Use DXE GCD services to graduate memory from unitialized to "available" in the UEFI memory map		Measure into PCR[0]
	DxeHii_code_t - Support glyphs, fonts	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]
	DxeIchInitialization_code_t - Initialize the SATA channel, etc Initialize the USB, other I/O devices	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]
	DxeMemoryTest_code_t - Invoked by BDS to initialize rest of memory. Untested memory from PEI in HOB that marks it as not available. This driver tests memory and if OK, uses DXE GCD to add memory to the memory map.	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]
	DxePciHostBridge_code_t - describes the programming of the chipset for different I/O segments	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]
	DxePciResourceAllocator_code_c - enumerates the bus resources and requests from the host bridge driver a programming aperature range	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]

	DxePciOverride_code_t - support for nonstandard devices	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]
	DxeIdeBusDriver_code_t - initializes the SATA/PATA/IDE controller, typically in the ICH	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]
	DxeUsbHostController_code_t - Handles the USB host controller, typically in the ICH	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]
	DxeUsbBot_code_t - supports USB blockoriented transfers/devices. Similar driver for CBI, etc	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]
	DxeFatFilesystemDriver_t - Implements the FAT12/16/32 support in UEFI	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but	Measure into PCR[0]
		instead in major block of flash.	
	DxeFlashDriver_code_t - supports read/write access to flash for cases where variables not implemented in SMM	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]

	BiosRuntime_code_t - 16-bit BIOS runtime for loading legacy OS through Int19h/mbr	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash. Stored in read-only f000-segment (not locked r/o control, though, like flash)	Measure into PCR[0]
	CsmDriver_code_t - Compatibility Support module. Supports loading 16-bit BIOS runtime BiosRuntime_code_t into 0xE/F0000p through 0xFFFFFp.	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]
	EbcInterpreter_code_t – Provide an interpreter for the EFI applications and drivers with a subystem type of EBC	Install-time check (signed capsule update). Flash protection (blocklocking). Typically not stored in boot-block but instead in major block of flash.	Measure into PCR[0]

Table 20

Compartment 6	**Type Name + description in compartment**	**Type Authentication and protection**	**Audit for compartment**
UEFI boot services 3rd party extensible	UefiOsLoader_code_t - Loaded across the network or stored on the EfiSystemPartition_data_t. Not used for drivers in HBA/PCI cards. Those are loaded when discovered during the PostSMM DXE compartment.	Stored on disk typically	Measure into PCR[4]
	UefiOsLoader_data_t - information about policy (e.g., Vista BCD, Elilo configuratino file, etc)	Stored on disk typically	Measure into PCR[5]
	IhvDriver_code_t - Independent Hardware Vendor UEFI driver code image. It's a PE/COFF executable loaded from platform firmware, a command line interface, a host-	Stored on disk or hostadaptoer card ROM	Measure into PCR[2]
	bus adapter (HBA) PCI device, and/or the UefiDriverLoadList_data_t		
	UefiBootOsBootVariables_data_t - list of the OS loaders to invoke from local media or network	Stored as generic UEFI variable, not a PV today	Measure into PCR[5]

	UefiDriverLoadList_data_t - UEFI variable that stores the list of device paths for onmedia UEFI drivers	Stored as generic UEFI variable, not a PV today	Measure into PCR[5]
	IhvDriver_data_t - IHV driver data that may be stored in UEFI variable		Measure into PCR[3]
	3rdPartyApplication_code_t	Stored on disk typically	Measure into PCR[4]
	3rdPartyApplication_data_t		Measure into PCR[5]
	EfiSystemPartition_data_t - one of possibly several partitoins that stores the UefiOsLoader_code_t's, etc		
	Gpt_data_t - GUID'd partition table.	disk	Measure into PCR[5]
	Mbr_code_t - master boot record	disk	Measure into PCR[5]
	Mbr_data_t - master boot record	disk	Measure into PCR[5]

Table 31

Compartment 7	Type Name + description in compartment	Type Authentication and protection	Audit for compartment
OS Runtime – OS-specific extensibility	OsKernel_code_t		OS specific
	OsKernel_data_t		OS specific
	SmmCore_code_t – from before	SMRAM protection	Done during load
	SmmDrivers_code_t's - from before	SMRAM protection	Done during load
	UefiRuntime_code_t - portions of the UefiSystemTable_data_t and their implementations that provide the UEFI2.x runtime services	Co-located in Ring0 w/ OS types	

	Smbios & ACPI_data_t - from earlier	Co-located in Ring0 w/ OS types	

Table 42

Compartment 7	Type Name + description in compartment	Type Authentication and protection	Audit for compartment
Afterlife – ideally only OEMextensible	SmmAfterLifeDriver_code_t	Install-time check via signed update, runtime isolation via SMRAM protection	Measured into PCR[0]
	SmmAfterLifeNetConsoleDriver_code_t	Install-time check via signed update, runtime isolation via SMRAM protection	Measured into PCR[0]

Table 53

References

[ACPI] http://www.acpi/info
[HEA] http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06Heasman.pdf
[DEG] http://www.ssi.gouv.fr/fr/sciences/fichiers/lti/cansecwest2006-duflotpaper.pdf
[TCG] www.trutsedcomputinggroup.org
[UEFI] www.uefi.org
[PI] http://www.uefi.org/specs/platform_agreement
[SDL] http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnsecure/html/sdl.asp