Agenda

• UEFI Specification and firmware Update
• Versions of UEFI and reference code bases
• Intel® UDK2010 SR1 UEFI 2.3.1 Security Features
• UEFI development platforms
• UEFI Resources
• Opens
• Backup
UEFI 2.1 Features

- Added protocols
  - HII (several protocols)
  - Absolute pointer protocol
- New member functions or equivalent
  - Driver Supported Version (for option roms)
  - Extended Simple Text In (more function keys supported)
  - Authenticated Variables
  - Extended SCSI Pass through
  - Signal on configuration change
  - EHCI exclusive ownership
  - Firmware storage device path
  - Hot key registration support
  - Run-time services with interrupts enabled
- Clean-up: e.g. error returns, * vs ** in declarations in several protocols introduced as a result of implementation
UEFI 2.2 Significant Features

- Networking – IPv6
  - IPv6 stack corresponding to existing IPv4 stack
  - Replacement for PXE protocols which are IPv6 compliant and large network friendly
    - Now being worked through IETF
  - Support for more LAN protocols: EAP and VLAN

- Security – Driver signing
  - Added optional ability to create firmware / OS trust relationships
    - Via key exchange
  - More signature combinations
  - Good / Bad list support
  - Platform owner control of denial response
  - Pre-Boot Authentication (PBA) Framework
    - Passwords, Smart cards, Fingerprint sensors, etc.
UEFI 2.2 Other Features

- **HII**
  - Additional operators for mapping to other standards
  - Page by page security control
  - Animation updates

- **EFI_ATA_PASS_THRU Protocol**
  - Gives direct access to ATA devices

- **UEFI Driver Health**
  - Allow for a driver to fix/re-configure (e.g. rebuild RAID set)

- **ABI Updates/Clarifications**
  - Floating Point/MMX/XMM
  - 16-Byte stack alignment

- **EFI_LOAD_FILE2 Protocol**
  - Loads non-boot-option EXEs (PCI option ROMs & apps)
  - Modifies LoadImage() behavior

- **EFI_LOADED_IMAGE Protocol**
  - Associates entire device path with EXE image

- **Bug fixes in spec for rest of document**
UEFI 2.3 Features

- Two possible views
  - Special release for ARM binding
  - Fairly quick release for items including ARM binding
- Also includes
  - Boot Services protocol for firmware update
    - Mainly for Option ROMs
  - Bug fixes
- Other items on deck
  - Ubiquitous Firmware Update
UEFI Forum Updates

- **UEFI Specification**
  - Version 2.3.1, Errata A published on Sept. 7, 2011
  - Clarifications from version 2.3.1
  - Additional ECRs are work in progress

- **UEFI Self Compliance Tests (SCT)**
  - Published a UEFI Winter 2012 Plugfest Release in Feb, 2012
    - Version 2.3.1 compliance test preview
    - Investigating coverage for 2.3.1 Errata A

- **Be Ready for Windows* 8**
  - UEFI 2.3.1 support
  - UEFI drivers and applications
  - Secure boot (sign the executables)
  - Seamless boot, hybrid boot, fast boot
  - IPv6 and IPv4 network stack
  - UEFI Spring 2012 Plugfest in Taipei (May 8-10), Redmond (July 16-20th)

- **PI Specification**
  - Version 1.2 Errata C published in October 2011

2012 marks the ubiquitous adoption of UEFI on PCs
Intel® UDK2010 SR1 (UEFI 2.3.1)

- User Identity (UID) Support (UEFI 2.3.1a)
- Secure Storage Protocol
  - Enable Opal/eDrive SATA devices using the EFI_STORAGE_SECURITY_COMMAND_PROTOCOL, ATA-8 Trusted Send/Receive and IEEE1667 Silo (UEFI 2.3.1a)
- Networking Improvements
  - Errata related to Netboot6-DUID
  - Provide more DHCP4 & DHCP6 API support
  - iSCSI (ip6) open source implementation for IPv6
- Support ATA Asynchronous Block Io (UEFI 2.3.1a)
- USB 3.0 Controller Support (XHCI)
- Update Internal Forms Representation (IFR) implementation to match UEFI 2.3.1 Specification (for setup screens or configuration)
- Fast boot support (asynchronous blockIO2)

www.intel.com/UDK
Intel® UEFI Development Kit (UDK) 2010 SR1

Updated for UEFI 2.3.1+ and PI 1.2+

Enabling key OS partners for UEFI 2.3.1

Enable UEFI 2.3+ Security Features

Intel® UDK 2010 SR1 enables key UEFI features for the industry
## UEFI System Classes

### Based on Firmware I/F

#### Today

<table>
<thead>
<tr>
<th>Class</th>
<th>Legacy BIOS</th>
<th>Interface Exposed</th>
<th>UEFI CSM*</th>
<th>UEFI Switch: CSM &amp; UEFI</th>
<th>UEFI only</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>UEFI CSM*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>UEFI Switch:</td>
<td>CSM &amp; UEFI</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 2013

- Class 2 configurations (default BIOS setup)
  - A - CSM and UEFI 2.x posted during boot
  - B - UEFI 2.x only posted during boot, CSM disabled
  - C - UEFI 2.3.1 only posted during boot, CSM disabled, secure boot enabled – only signed modules executed

*A-CSM: Compatibility Support Module or Legacy BIOS written as an UEFI driver*

- A and B shipping today (2012)
- C - Required by Windows 8 logo
Windows 8 Certification – UEFI

- Requirements:
  - All Windows 8 Client systems must ship in native UEFI mode
    - Class 2 - CSM Disabled
    - Class 3 - No legacy bios Native UEFI in firmware
  - Baseline is UEFI 2.0 Windows 7 requirements
    - Secure Boot
    - New graphics requirements
    - POST time maximums
    - OEM Certification display guidance
- If Implemented
  - BitLocker network key protector
  - BitLocker Encrypted Hard Drive (eDrive) support
- Windows 8 Logo requirements (system fundamentals for BIOS)
  Must provide way of turning of secure boot in BIOS if implemented
  Must also provide a custom setup menu for pulling in certs

\(^1\)New with UEFI 2.3.1
UEFI Reference code base

- Tianocore.org EDKI reference base
  - EDK1117
  - EDK 1.0 - EDK 1.06 (last EDK 1 release)
  - UEFI 2.0 to UEFI 2.1 reference implementations

- Tianocore.org EDKII packaging
  - UEFI 2.3.1 reference code
  - Designated UDK2010 (latest version SR1)
  - All native UEFI 2.3.1 reference code
  - Hybrid versions available with both UEFI 2.3.1 but allow drivers from EDK1 to exist within (called ECP).
EDK II / Intel® UDK2010 Support

- OEM / IBVs
  - Initial support on reference platform will come from SSG
  - As Client and Server platforms transition to Intel® UDK2010 common core, the support will be as follows:
    - **First Line:** Client and Server BIOS AE Teams would be first line of OEM customer and IBV support
    - **Second Line:** SSG would be second line of support

- Other Ecosystem Players
  - SSG will plan on enabling other ecosystem players such as OSVs, ISVs, IHVs, etc.

- Client/Server Group Validation and Support
Different versions of UEFI in Marketplace
What is available today

- **Class 2a/b with UEFI hidden in setup (Most common today)**
  - BIOS derived from EDKI reference code (oldest is EDK1117)
  - UEFI 2.0/2.1 hidden in a bios setup option
  - Default boot CSM/Legacy bios
  - Does not pass UEFI SCTs
  - Missing some UEFI protocols and drivers (pxe boot, usb drivers, GOP video etc.)
  - UEFI drivers untested
  - Fancy BIOS setup screens that use graphics video mode

- **Class 1 with no UEFI**
  - Default boot CSM/Legacy bios. No UEFI drivers
  - 11/17 2007 EDK1 PI drivers for silicon setup only of cpu, chipset and memory
UEFI 2.3.1 Secure Boot
What is the heart of Trust

- The hardware root of trust includes
  - TPM
  - Flash
  - Binding of above into system
- TCG defines TPM’s functionality
  - Protected capabilities
  - Shielded locations

- Not the implementation
  - Vendors are free to differentiate the TPM implementation
  - Must still meet the protected capabilities and shielded locations requirements

Need a hardware root of trust
SRTM \(^1\) for Platform Firmware

Firmware use of TPM and Measurements

\(^1\) Static Root of Trust for Management (SRTM)
CRTM

- **What is CRTM**
  - Core root of trust for measurement
  - Detects physical presence and initiates measurements for rest of firmware bootstrap

- **Properties of CRTM**
  - Immutable, or never changed in the field
  - Appropriate cryptographic techniques need to be employed in order to update the CRTM.

- **For updatable CRTM**
  - A signed capsule is one implementation path.
  - Need manufacturer-approved/secure update process

_**CRTM is the firmware foundation of trust**_
UEFI/PI Architecture Boot Flow – Create/Evaluate Integrity List

- TPM Init
  - SHA1 Algo
  - Physical Presence
- Measure FV_MAIN
  - PCR Event Log
- OpROM Scan
  - Measure ROMs
  - Update PCR2, Log Event
- ASL Code
  - Legacy Boot, Measure IPL, Update PCR4, Log Events etc.

Measure Into PCR’s

Measure & Create
Measured items in UEFI

Standardized way to measure and report
Why Implement UEFI Secure Boot?

- As OS becomes more resistant to attack the threat targets the weakest element in the chain
- And 16-bit Legacy Boot is not secure!

It should be no surprise that a TDL Gang botnet climbed into the number one position in the Damballa Threat Report – Top 10 Botnets of 2010. “RudeWarlockMob” … applied effective behaviors of old viruses and kits. It combined techniques that have been effective since the days of 16-bit operating systems, like Master Boot Record (MBR) infection … with newer malware techniques.  
(from http://blog.damballa.com)

- Secure Boot based on UEFI 2.3.1 removes the Legacy Threat and provides software identity checking at every step of boot – Platform Firmware, Option Cards, and OS Bootloader
Intel® UDK2010 SR1 Security Features

- **UEFI Secure Boot**
  - UEFI variable support for UEFI Secure Boot as defined by UEFI 2.3.1a (EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS attribute with EFI_VARIABLE_AUTHENTICATION_2 and EFI_VARIABLE_AUTHENTICATION support)
  - DXE Image Verification library to support UEFI Secure Boot (UEFI 2.3.1a)
  - PK x509 Certificate Support
  - Support EFI_VARIABLE_AUTHENTICATION_2 for PK variable format (UEFI 2.3.1a)
  - Add enable/disable mechanism for UEFI Secure Boot

- **TCG Trusted Boot**
  - TCG EFI Platform Specification
  - Static root of trust starting early during BIOS/Firmware
  - Using the 1.2 TPM to measure executable code and put hash into PCIRs.
  - TCG Specs for trusted boot: (TPM 1.2, physical presence, UEFI)
    - [http://www.trustedcomputinggroup.org/developers/pc_client](http://www.trustedcomputinggroup.org/developers/pc_client)
Secure Boot – Three Components

1. Authenticated Variables

2. Driver Signing

3. System-Defined Variables
UEFI Authenticated Variables

- Uses standard UEFI Variable Functions
- Available Pre-boot and also Runtime
- Typically stored in Flash
- Variable Creator signs Variable Hash with Private Key (PKCS-7 Format)
- Signature & Variable Passed Together for Create, Replace, Extend, or Delete
- Several System-defined variables for Secure Boot

Extensible Integrity Architecture
Updating Authenticated Variable

- Support for Append added (UEFI 2.3.1) * implemented
- Counter-based authenticated variable (UEFI 2.3)
  - Uses monotonic count to against suspicious replay attack
  - Hashing algorithm - SHA256
  - Signature algorithm - RSA-2048
  - Deprecated (not used anymore)
- Time-based authenticated variable (UEFI 2.3.1) * implemented
  - Uses timestamp as rollback protection mechanism
  - Hashing algorithm - SHA256
  - Signature algorithm - X.509 certificate chains
    - Complete X.509 certificate chain
    - Intermediate certificate support (non-root certificate as trusted certificate.

Protected Variables that can be Securely Updated
Secure Boot – Three Components

1. Authenticated Variables

2. Driver Signing

3. System-Defined Variables
UEFI Driver Signing

- UEFI Driver Signing Utilizes Microsoft* Authenticode* Technology to sign UEFI executables

In Secure Boot, signatures should be checked:
1. UEFI Drivers loaded from PCI-Express Cards
2. Drivers loaded from mass storage
3. Pre-boot EFI Shell Applications, f/w updaters
4. OS UEFI Boot-loaders

UEFI Signing is not applied to
1. Drivers in the Factory BIOS
2. Legacy BIOS components (also known as CSM)
3. CSM must be disabled in boot for system to be secure (UEFI boot)
   - CSM can be enabled in setup for non-UEFI boot options
   - Shell is not considered a secure boot option

Enhanced by UEFI 2.3.1
Secure Boot – Three Components

1. Authenticated Variables

2. Driver Signing

3. System Defined Variables
## Secure Boot Authenticated Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PK</strong></td>
<td>Platform Key – Root key set to enable Secure Boot</td>
</tr>
</tbody>
</table>
| **KEK**  | Key Exchange Key  
List of Cert. Owners with db, dbx update privilege |
| **db**   | List of Allowed Driver or App. Signers (or hashes) |
| **dbx**  | List of Revoked Signers (or hashes) |
| **SetupMode** | 1 = in Setup Mode, 0 = PK is Set (User Mode) |
| **SecureBoot** | 1 = Secure Boot in force |

**Notes:**
- Owner of cert. in KEK can update db, dbx
- Owner of cert. in PK can update KEK

UEFI Defines System Databases for Secure Boot
Secure Boot – Three Components

1. Authenticated Variables
2. Driver Signing
3. System Defined Variables

= UEFI 2.3.1 SECURE BOOT
Secure Boot Begins @ the Factory

Pre-production → Production → User

Certificate Generating Station @ OEM

1. Initial Security Load

2. Initial Security Load is installed onto each computer at the factory, enabling Secure Boot.
   - 1) Initial db and dbx
   - 2) KEK with allowed updaters
   - 3) Platform Key (PK)

3. After delivery, the OEM or OSV can update with new certificates or revoked certificates (dbx)

OEM collects certificates provided by OSVs, Partners, and OEM’s own keys.
“DB Generator” creates the Initial Security Load for new computers.

OEM Responsible for Initializing Secure Boot and can allow user to disable Secure Boot or add KEK, PK, DB in setup
Secure Boot Protects the User

User attempts to boot a compromised system

OS Boot-loader image checked against pre-loaded database

Root-kit fails checks, user protected by Secure Boot

Secure Boot Tests Signatures to Reject Potential Threats
UEFI secure boot tools

- IA32 and X64 OVMF  (run UEFI in an emulator on your build system)
  - Run UEFI NT32 boot under OVMF add use secure boot setup menu
  - Add `D SECURE_BOOT_ENABLE` IN Nt32pkg to enable

- Secure boot
  - Signing UEFI applications and Drivers for UEFI secure boot
  - Security package on Tianocore.org
  - Signtool Microsoft toolchain
    - Signtool (sign an executable image, Makecert (create private key), Pvk2Pfx convert .pvk files to .pfx)
Tunnel Mountain Intel DQTM57 UEFI 2.3.1 platform

Intel® UDK 2010 Compatible, supports UEFI 2.3.1
Pre-assembled systems available at HDNW, visit
http://www.Tunnelmountain.net
tomk@hdnw.com, (425) 943-5515 ext 42234. Use product name “Tunnel Mountain” when ordering

Comes with class 2 CSM and UEFI enabled firmware
Download site has Class 3 UEFI only firmware (nocsm)
Comes with serial port for debug
Can be ordered with optional ITP connector rand socketed SPI flash - AC-SPEC4480
*Romley Server UEFI 2.3.1 to be available Q2 ’12
*Sandybridge Desktop DQ67SW Q2 ’12
* - not available yet in validation test

Release 10 support secure boot and setup menus for adding
Certs to the firmware

Visit http://developer.intel.com/technology/efi/uefi-ihv.htm for the latest information and platform BIOS downloads
Spring 2012 UEFI Plugfest in Taipei, Taiwan

- Next UEFI plugfest July 16-20th Redmond WA (Microsoft campus)
- Visit [www.uefi.org](http://www.uefi.org) for event details and registration
- Next opportunity to see upcoming UEFI systems and to test them.
UEFI Resources

- Intel UEFI sites
  - www.intel.com/technology/UEFI
  - Disk utilities, IDF presentations,
  - www.intel.com/udk2010  (To be launched)

- Ubuntu resources
  https://help.ubuntu.com/community/UEFIBooting
Backup
UEFI Secure Boot VS TCG Trusted Boot

UEFI authenticate OS loader (pub key and policy)
Check signature of before loading

• UEFI Secure boot will stop platform boot if signature not valid (OEM to provide remediation capability)
  • UEFI will require remediation mechanisms if boot fails

TCM/TCM will measure OS loader into PCR (Platform Configuration Register)

• TCG Trusted boot will never fail
• Up to other SW to make security decision using attestation
UEFI Secure Boot Component:

- **UEFI Driver (option rom) Signing**
  - The system provider may decide to authenticate either the origin of the executable or its integrity.

- **Authenticated UEFI Variable**
  - It provides a way to protect the critical variable being modified by malicious software.

- **Firmware/OS Key**
  - We can create a trust relationship between the platform owner, the platform firmware, and an operating system.
UEFI Driver Signing

- **Why?** - Origin & Integrity
- **How?** - Authenticode PE

PE Header
- Certificate Directory
- Section 1
  - ...... Section N
- Type
- Attribute Certificate Table

ContentInfo
- PE file hash

Certificate
- X.509 Certificate

SignInfo
- Signed hash of ContentInfo

PKCS#7 + Authenticode Ext
UEFI Authenticated Variable

- **Why?** - Integrity (no confidentiality)
- **How?** - Time Based

Authenticated Variable

- **Input Variable**
  - Data Authentication
    - Time Stamp
    - Type
    - Certificate
  - Data Content

PKCS#7
- ContentInfo
  - N/A
- Certificate
  - X.509 Certificate
- SignInfo
  - Signed hash of
    - VariableName + VariableGuid + Attributes + TimeStamp + DataContent
Firmware/OS Key

- **Why?** - How can firmware know if certificate is valid?
- **How?** - Firmware/OS Key

(Signature Database)

- Certificate
  - X.509 Certificate

- UEFI Signature List
  - Type

- UEFI Signature Data
  - Owner
  - Signature

- UEFI Signature Data
  - Type

- UEFI Signature Data
Put them altogether: UEFI Secure Boot
Put them altogether: UEFI Secure Boot

1. Enroll

Authenticated Variable
- PK
- KEK

2A. Signed Image Discover
- Certificate + SignInfo
- OsLoader.efi

2B. Signature Verification

Variable
- db
- dbx

2C. Signed Image Load
- Certificate + SignInfo
- OpRom.efi

DXE FV
- Image Verify
UEFI Secure Boot Database Review

- **PK**: Update Enable
- **KEK**: KEK
- **db**: Update Enable
- **dbx**: Update Enable

*If Signed by key in db, driver or loader can Run!*

*If Signed by key in dbx, driver/loader forbidden!*
Public vs. Private Keys

- A pair of keys, one public, one private, are created
- Private keys stay secure at Partner or in the OEM’s Security Office
- Private keys are used to ‘sign’ objects
- Only Public keys loaded into the Platform
- Public keys are used to check signatures
- GPL3 does not allow private keys

Private Keys Must be Stored Securely!
Who “Owns” The System Security Keys?

- **PK** – Key pair is created by Platform Manufacturer
  Typically one PK pair used for a model or model Line

- **KEK** – Key supplied by OS Partner,
  Optional: Include 2\textsuperscript{nd} key created by OEM

- **db** – OS Partner supplies Key,
  CA Partner supplies Key,
  Optional: OEM App Signing Key

- **dbx** – list of revoked keys
  - Signing authority issues revoked keys

*Signature Tests using db Keys Block Rogue S/W!*
OEM Key Administration

- Keys are installed for testing with target OS
- Keys are installed in the factory before shipping
- BIOS setup screens allows for population of keys by user in Administrative mode (i.e., from USB stick, internet etc.)

**Preparation Tasks**
1. Gather public keys from partners
2. Generate PK for model
3. Make a package of initial key load
4. Occasional maintenance of forbidden list

**Repetitive Tasks**
1. Factory will boot and install the initial key load

Who administers keys in Linux community?
tianocore.org source location
where to get secure boot systems
where to get uefi systems