UEFI for Blue Teams

Security B-Sides Portland 2015
bsidespdx.org

2015-Oct-17

Lee Fisher
unifex <at> riseup <dot> net

"Firmware security is only meaningful if you have a mechanism to verify that your firmware hasn't been modified."

--Matthew Garrett (@mjg59)

Agenda

- Brief history/architecture of BIOS and UEFI
- Brief summary of NIST and NSA/CC/IAD BIOS security guidelines.
- Discussion of NIST's firmware-centric platform lifecyle, and attempt at integration with current hardware lifecycle management styles.
- Discussion of firmware vulnerability assessment and diagnostic tools, to help accomplish most of the previous lifecycle tasks.
- More Information on firmware.
- Calls to Action to improve organization and ecosystem.

Goals

- Learn about NIST's FW platform model, and integrate it (and FW) into existing HW/SW lifecycle models, to help with firmware security.
- Learn about available open source firmware vulnerability assessment and QA tools, and other techniques.
- Suggest information sources and other resources for learning more about firmware security.

Requirements

- You know the basics of computer security
- You know the basics of Intel hardware and firmware (IBM PC BIOS/OpROM firmware, and UEFI firmware.
  - Though we will cover some UEFI concepts, this won't be an introduction to UEFI.
- You can use a shell (bash, cmd.exe) and write shell scripts (batch files) for them.
- You can use Python to run scripts.
- Optional: Python, C, and a bit of C++ programming language skills to extend/customize tests.

Credits

- The badly-drawn graphics are mine. Any good graphics in these slides were drawn by others, attribution listed.
- For about the NIST SP 144/155 slides, I refactored some text from about 4 slides from:
  - “New BIOS Protections for Government Enterprise Clients” by
  - Bob Clemons (NSA/IAD) and Andrew Regenscheid (NIST)

Scope

- Focusing only on open source tools
  - Ignoring some powerful freeware/commercial tools (mainly Windows-centric MITRE Copernicus).
- Focusing on Intel x86/x64 systems
  - Ignoring: Intel Itanium, AMD, ARM, OpenPOWER, MIPS, etc.
- Focusing on UEFI-flavored firmware
  - Ignoring BIOS (though it's not dead yet)
  - Ignoring coreboot and U-Boot, both excellent technologies, for time constraints.

Why are we talking about this?

- Firmware-level malware ('bootkits', 'firmworms', etc.) is invisible, if you aren't looking for it – at both live hardware and virtualization levels.
- Everyone (OS vendors, OEMs, IHVs, tool ISVs, and attackers) now targets UEFI.
- Many (numbers unknown) modern OEM systems are still designed insecurely. Public exploits exist.
- Where tests exists (eg, Intel platforms), hardware reviewers should be informing consumers of insecure models, but they are not, consumers remain ignorant.
- NIST guidance aside, current System Administrator best practices doesn't seem to include firmware.

Define: firmware

- What is firmware?
  - Microcode: code to fix bugs in processor
  - TEE/BIOS/UEFI: The kind that ships with the system, not the embedded distro, and the OS uses it, baked into the system, hard to replace, extended by BIOS Option/Expansion ROMs or UEFI drivers.
  - Coreboot/U-Boot: the kind that ships with embedded Linux (or other OS), along with the OS components. Eg, OpenWRT router 'firmware'.
  - Often, embedded OS components are called 'firmware', eg Android 'firmware update'
  - USB firmware: ??

Firmware Solutions

- Modern systems come with a variety of firmware:
  - BIOS (x86 devices)
  - EFI 1.x/UEFI 2.x (Apple, Windows OEMs)
  - Coreboot
    - Chromium's variation
    - Libreboot variation
  - U-Boot (embedded devices)
  - ...Multiple vendor-specific solutions

More secure boot flavors

- Most tech from last slide also have a more secure/trusted/verified/measured mode, most specific to a HW/FW/OS technology stack:
  - TCG Measured Boot
  - UEFI Secure Boot
  - Trusted Boot (Intel TXT)
  - Solaris Verified Boot
  - Android Verified Boot
  - Chrome Verified Boot (Class A and B)
  - U-Boot Verified Boot
  - ...

Firmware (System) complexity

- Simple:
  - Single user, single process, single CPU/core, no Service Processor, single OS, local boot.
- Intermediate:
  - multi-users, multi-process, multiple CPUs/cores, multiple OSes (dual-boot), network boot
- Advanced:
  - N systems under 1 Service Processor, concurrent OSes (eg, AMIDuOS), OOB management

Reminder: scope

- Henceforth, I'll be mostly focusing on UEFI, not coreboot or U-Boot or their more secure variants. All are great, but I don't know U-Boot or coreboot well enough to talk about them yet.
- coreboot/U-Boot/other experts: please clue me in w/r/t security tools/tactics!

BIOS

- Created for the IBM PC, as a boot loader for the IBM PC and a systems library for IBM PC-DOS, then MS-DOS OEMs begin cloning the IBM PC BIOS...
- MBR: boot loader is not a file, but embedded in MBR structure, file must be next to MBR
- BIOS can't traverse FAT or other file systems
- Intel 8086 Real Mode-only (segment:offset)
- BIOS is an x86 blob, no driver standard
- IHV extend via OpROMs, no driver standard
- Assembly, closed source, except early IBM tech ref manual; splinted ecosystem (each BIOS clone)
- Various size and other limits. NO SECURITY FEATURES.
- Firmware updates varied by vendor

Rings of Protection

- Early Intel processor security model (MS-DOS-era, aka no security):
  - Ring 3 (outermost): user mode, apps
  - Rings 2-1: often not used.
  - Ring 0 (innermost): kernel mode, drivers/OS kernel
- Simplistic model, 3 (aka userspace) and 0 (aka kernespace), ignoring nuances of physical and virtual firmware and silicon levels.
- Invisible Things Lab (ITL) and others have proposed adding new negative rings – below ring 0 – to clarify this (next slide).

Negative (Subzero) Rings

- Ring -1: VM, hypervisor/CPU (Intel VMX, AMD SVM)
- Ring -2: Intel SMBIOS, SMM
- Ring -3: Firmware (BIOS, UEFI, coreboot, U-Boot, ...)
- Ring -4: hardware, silicon exploits
- Note how firmware (-3) unites all of these rings: it is the gateway to access SMM (-2) and HW (-4), and also works on virtualized systems (-1), and can be accessed via OSes/drivers (0) and apps (3).

NIST BIOS Guidelines

- In response to BIOS insecurities, NIST has released 3 documents giving BIOS security guidance to BIOS vendors, as well as users (enterprise sysadmins):
- 2011: SP800-147:
  BIOS Protection Guidelines
- 2011: SP800-155:
  BIOS Integrity Measurement Guidelines (Draft)
- 2014: SP800-147B:
  BIOS Protection Guidelines for Servers

SP800-147

- Defines best practices for various phases: Provisioning, Platform Deployment, Operations and Maintenance, Recovery, Disposition.
- Defines authenticated BIOS update mechanism, and optional secure local update mechanism.
- Defines Integrity protection and non-bypassability features.

SP800-147B

- Expands 147's model from 'PC' (Basic Server) to include more complex servers (Managed Server, Blade Server), with dedicated management channels, possibly a Service Processor.
- Service Processor as Root of Trust
- Non-Bypassability of BIOS Protections by Service Processor
- Adds authenticated remote online firmware update mechanisms, in addition to local updates.

SP800-155

- Provides the hardware support to Roots of Trust for BIOS integrity measurements.
- Enables endpoints to measure the integrity of all firmware components and configuration data components.
- Securely transmits measurements of BIOS integrity from endpoints to the Measurement Assessment Authority.
- Use of Trustworthy Computing security to augment firmware security, locally (TPM) and remotely (TNC).

NIST 147 recommendations

- See section 3.2
- Recommended Practices for BIOS Management
  - Update systems to 800-147-compliant BIOSes, if available for your systems.
  - Keep track of deployed versions
  - Use the authenticated update mechanism to update BIOS
  - Monitor the BIOS for deviations and remedy if necessary

NIST 155 recommendations

- New computer purchases should include an 800-147-compliant BIOS
- Update to 800-147-compliant BIOSes if available
- If not available, consider the criticality of the unprotected systems
- New computer and IT infrastructure purchases should support BIOS measurement
- Make BIOS management part of platform lifecycle

NIST 155 goals

- Goal: detect unauthorized changes so administrators can remedy
- Means:
  - Roots of Trust to measure, store, and report to Measurement Assessment Authority (MAA)
  - MAA verifies measurements
  - MAA can instruct IT components (e.g., managed switch) to respond accordingly

NIST 155 use cases

- Basic Measurements Reporting
  - Notify administrators of changes
- Comply-to-Connect
  - Network access control
  - TNC framework
- Continuous Monitoring

NIST: 3-pronged approach

- SP800-147: Protects systems from unauthorized BIOS modification by defining a secure, non-bypassable authenticated update mechanism.
- SP800-147b: Extends SP800-147 system model, from simple PC to more sophisticated servers with BMC and OOB update mechanisms.
- SP800-155: Outlines a framework for a secure BIOS integrity measurement and reporting chain for client systems, to detect unauthorized modification of System BIOS and configuration using secure measurement and reporting mechanisms.

NIST updates

- See blog post[1] for interview with Andrew of NIST. He clarifies the 'golden master' part of the Provisioning phase, countersigning, and a few other things.
- firmwaresecurity.com/2015/10/13/interview-with-andrew-regenscheid-of-nist/

NSA – BIOS Protection Profile

- NIST isn't the only org issuing firmware security guidance.
- IAD (Information Assurance Directorate): BIOS Update Protection Profile, 1.0, 2013
- Common Criteria's Standard Protection Profile (PP) for PC client devices BIOS firmware (including UEFI).
- “Addresses the primary threat that an adversary will modify or replace the BIOS on a PC client device and compromise the PC client environment in a persistent way.”
- AFAICT, no Validated Products with this PP. :-(
- www.niap-ccevs.org/pp/pp.cfm

Linux Foundation: IT Policies

- Linux Foundation recently started offering guidance on securing systems, which includes UEFI, SecureBoot, TPM, BIOS password, and other firmware-level security measures. Their current advise only targets Linux workstations, but would be a starting point for other devices (from embedded/IoT devices to servers)
- This guidance needs to be integrated with Linux workstations in below model section...
- https://github.com/lfit/itpol
- http://linuxfoundation.org

SECTION: UEFI

- Next few slides: background on UEFI technology

What is UEFI?

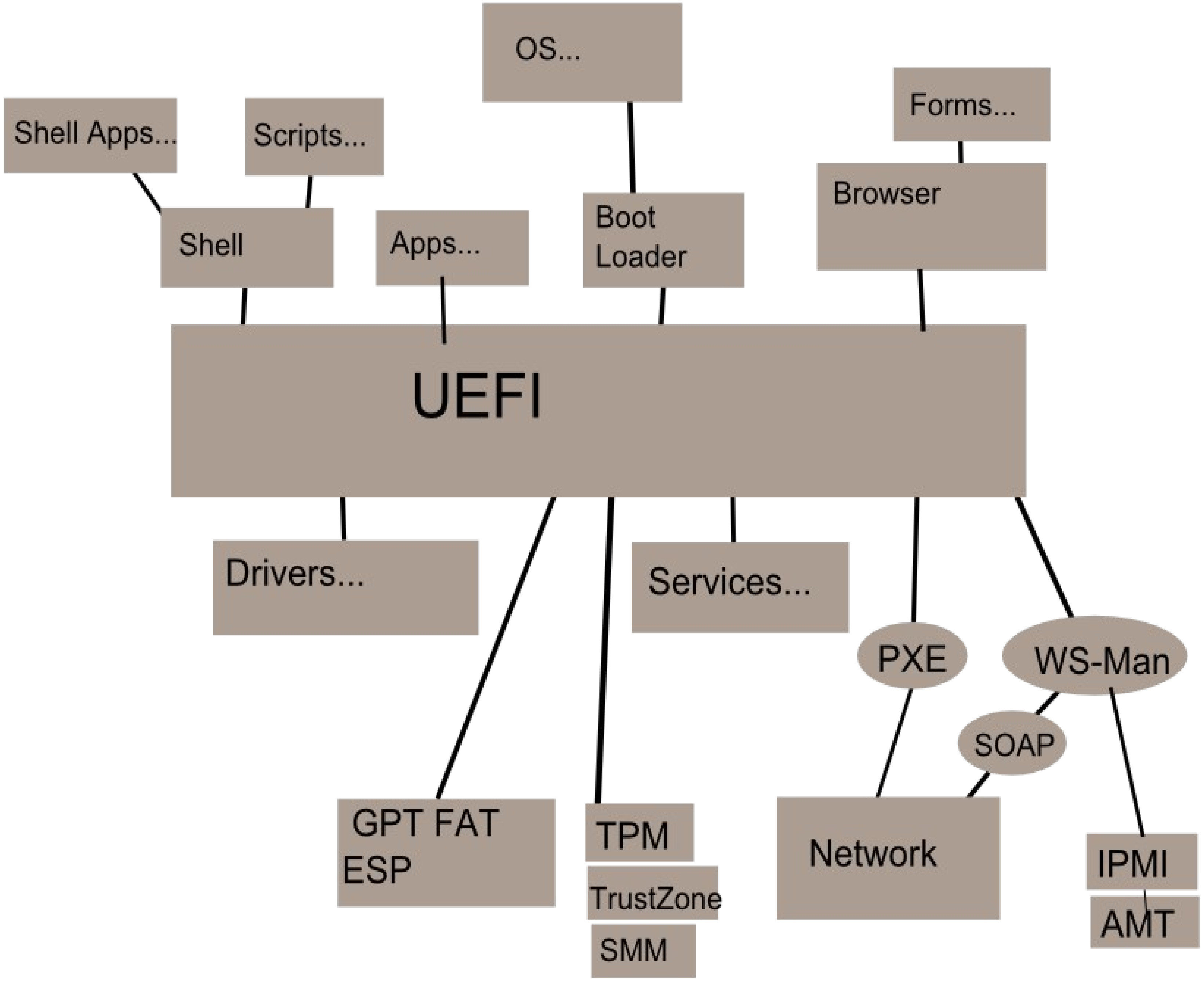
- Universal Extensible Firmware Interface (UEFI)
- A complete technical replacement for the legacy PC BIOS firmware, available for many systems and architectures.
- Created by Intel, called EFI, for Itaniums. Apple used it when switching from to Intel. Windows 8 made it common on Windows OEM systems.
- UEFI Forum the industry trade group that controls UEFI, membership at Adopter and Contributor levels.
- uefi.org (specs, tests); tianocore.org (code)

UEFI Ecosystem

- Chip vendors
  - Intel, ARM, AMD, ...
- OEMs (Original Equipment Manufacturers)
  - HP, Dell, Apple, Microsoft, ...
- ODMs (Original Device Manufacturers)
- IHVs (Independent Hardware Vendors)
- IBVs (Independent BIOS Vendors)
  - Phoenix, AMI, Insyde Software, Nanjing Byosoft
- OSVs (Operating System Vendors)
  - Apple, Microsoft, Red Hat, Canonical, SuSE, ...
- ISVs (Independent Software Vendors)
  - 'pre-OS' and 'OS-present' applications

UEFI

- GPT partition, no embedded boot loader like MBR
- Firmware can traverse FAT32 file systems
- Firmware volume FV structure, multiple .efi modules and other code/data.
- FFS3 file system for .efi-based OpROM files on flash
- TE .efi image format, based on PE/COFF, for app/driver/service – and “Pre-OS” app binaries.
- ESP (EFI System Partition), FAT32 partition, with additional firmware files, .efi, beyond firmware volume. Multiple boot loaders can exist in different subdirectories.
- NVRAM variables that point to the appropriate boot manager or boot loader to load.


OS...
Forms...
Shell Apps...
Scripts...
Browser
Shell
Apps...
Boot Loader
UEFI
Drivers...
Services...
PXE
WS-Man
SOAP
GPT FAT ESP
TPM
TrustZone
SMM
Network
IPMI
AMT

Secure Boot

- Optional build feature of UEFI 2.x
- Required by Microsoft for Windows OEMs to use Windows
- Uses multiple sets of keys
  - PK, Platform Key, verified KEKs
  - KEK, Key Exchange Keys, Verify db and dbx
  - db, Signature Database
  - dbx, Forbidden Database
- See '*Secure Boot on Linux*' talk at IDF2013
- See Intel Secure boot talk at DEF CON 21(?)

Firmware: both real and virtual

- UEFI is of course on physical hardware
  - Intel: UEFI on x86/x64 systems, Itanium systems are of course EFI-centric.
  - AMD: UEFI on most systems
  - ARM: UEFI is an option for AArch32/AArch64 by Linaro
  - Unofficial ports: MIPS, OpenPOWER (2)
- Virtualization software also provides virtual firmware
  - QEMU is the best solution for virtualized UEFI
  - VirtualBox, Xen, KVM also support OVMF
  - VMWare and Microsoft HyperV support UEFI

Terse Executables

- UEFI uses PE/COFF-based image file format.
- UEFI images (eg *.efi) have an image format called **TE** (*Terse Executable*), minor small variation to **PE** (*Portable Executable*) format.
  - Adds a few new flags to structure.
  - Signature is EFI-era's '**VZ**', not MS-DOS-era's '**MZ**'.
  - Removes some unused tables/segments that PE uses.
- Leverage existing compiler tools's PE support, then translates to TE at last minute.
- Uses code signing for security: executables are signed similar to Windows' Authenticode.
- The format of the various modules in a firmware volume.

EBC (Efi ByteCode)

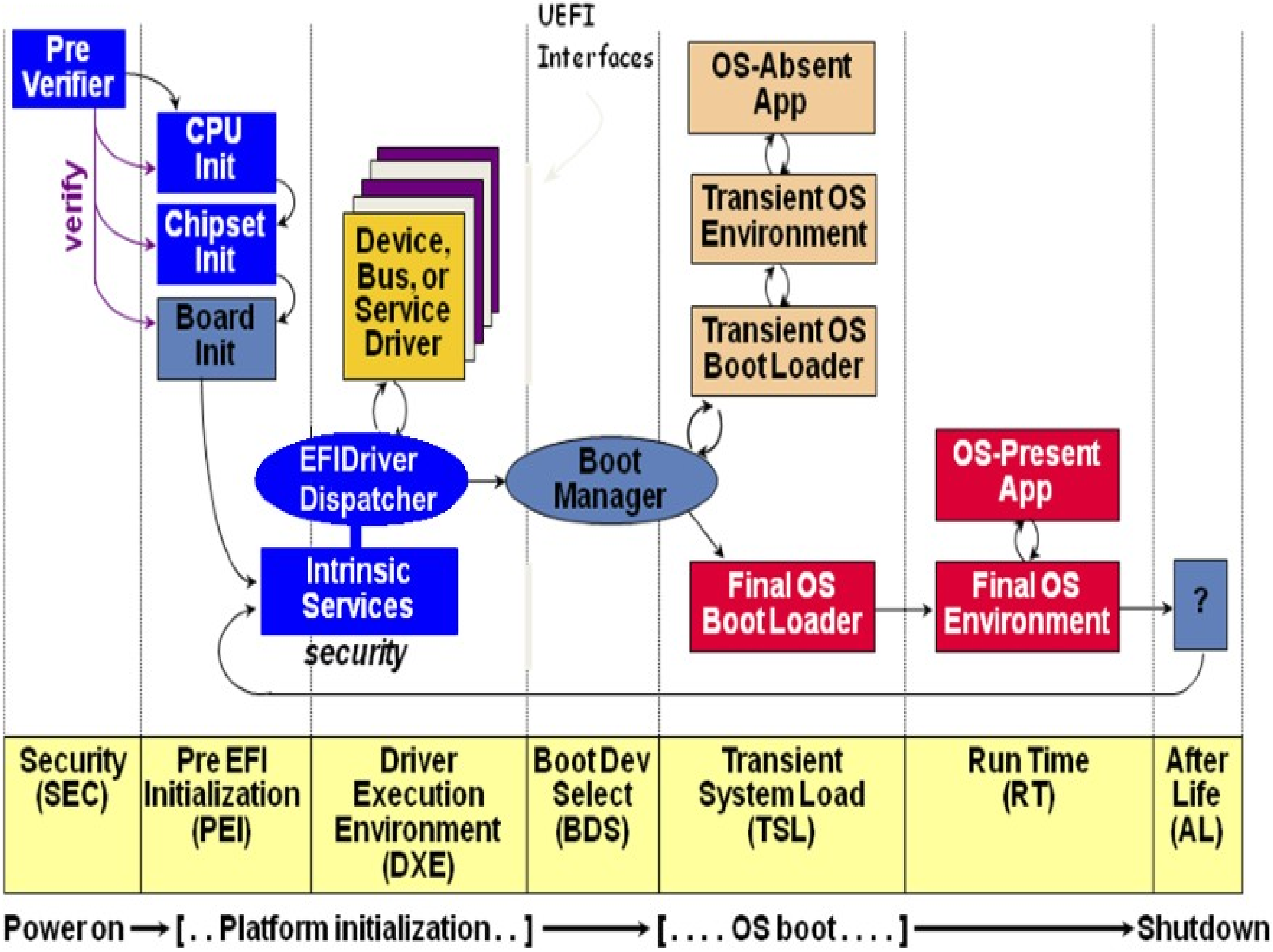
- A bytecode that supports multiple CPU opcode targets: Intel x86, Intel x64, Intel Itanium.
- Goal: compiler C code to EBC, then only 1 driver for all Intel platforms, instead of 3 separate blobs, less ROM space needed.
- Issue: EFI currently only targets Intel hardware platforms, not all EFI-targetting platforms (eg, ARM 32- and 64-bit, MIPS, OpenPOWER, ...)
- Issue: Intel C Compiler is the only compiler that targets EBC. ICC is a commercial-only product, part of Intel System Studio.
- Some UEFI drivers will be EBC-based.

Driver Models

- PEI Modules and Services
  - PEI Modules (PEIMs)
  - PEI = Pre-Efi Initialization, Modules = Drivers
- DXE Drivers and Services
  - DXE = Driver eXecution Environment
- UEFI Drivers and Services
- UEFI 'Pre-OS' Applications
  - System/security tools, boot loaders/managers, like GRUB2, rEFInd, …
- Earlier EFI 1.x had a different driver model.

PI/UEFI States

- UEFI has a 7-state lifecycle:
  - **SEC**, **Sec**urity
  - **PEI**, **P**re-**E**FI **I**nitialization
  - **DXE**, **D**river e**X**ecution **E**nvironment
  - **BDS**, **B**oot **D**evice **S**election
  - **TSL**, **T**ransient **S**ystem **L**oad
  - **RT**, **R**un**T**ime
  - **AL**, **A**fter**L**ife
- Platform Init specs define: PEI-DXE stages.
- UEFI specs define BDS-onward stages.
- Next slide's image source: uefi.org.


UEFI Interfaces
Pre Verifier
verify
CPU Init
Chipset Init
Board Init
Device, Bus, or Service Driver
EFI Driver Dispatcher
Intrinsic Services
security
Boot Manager
OS-Absent App
Transient OS Environment
Transient OS Boot Loader
Final OS Boot Loader
OS-Present App
Final OS Environment
?
Security (SEC)
Pre EFI Initialization (PEI)
Driver Execution Environment (DXE)
Boot Dev Select (BDS)
Transient System Load (TSL)
Run Time (RT)
After Life (AL)
Power on → [. . Platform initialization . .] → [. . . . OS boot . . . .] → Shutdown

UEFI Services

- UEFI Boot Services
  - used by OS loader to transition from FW to OS kernel (or boot manager, a pre-OS UEFI app)
  - Only used during init, they're not available later.
- UEFI Runtime Services
  - Similar to BIOS interrupts like 10h (video) and 13h (disk), UEFI has services that the OS uses.
  - virtual memory, time, variables, console i/o, reset, capsule, memory, event/timer, protocol, image

Variables

- UEFI's Variables runtime service are similar to 'environment variables' of Unix/MS-DOS: key/value string pairs, but more complex.
- Besides name/value strings, each variable also has a GUID, and multiple attributes (eg, if it persists boots, when it can be used).
- Some variables stored in NVRAM, some variables not accessable by shell, etc.
- UEFI uses MS-DOS style %foo% substitution, not Unix style $foo.
- UEFI Shells PATH is “;”-separated, using absolute or relative paths, eg: ".\;\efi\tools\;\efi\boot\;\"

Variables

- Common UEFI Variables
  - Lang, PlatformLangCodes, PlatformLang, ConIn, ConOut, ErrOut, ConInDev, ConOutDev, ErrOutDev, Timeout, Boot####, BootOrder, BootNext, BootCurrent, Driver####, DriverOrder
- Common UEFI Shell-specific Variables
  - Path, Cwd, DebugLastError, LastError, Profiles, ShellOpt, ShellSupport, UefiShellSupport, UefiVersion, UefiShellVersion
- UEFI v2.5 adds a few new variables, too

UEFI Network Stack

- Besides booting locally, UEFI systems can remotely boot using PXE. UEFI 2.5 adds HTTP Boot.
- Admins can remotely control UEFI systems (even when powered off) via IPMI, using WS-MAN. UEFI 2.5 adds DMTF Redfish.
- Network stack protocols:
  - iSCSI, IPv4 and IPv6, ARP, DHCP (see RFC 5970), UDP, TCP, PXE, TFTP, IPsec, VLAN, Ethernet, CHAP, WS-MAN, …
  - UEFI 2.5 adds: HTTP, DNSv4 and DNSv6, WiFi, BlueTooth, EAP2
- UEFI Shell commands:
  - ifconfig, ping, vlan config, telnet

CSM: BIOS for UEFI

- Compatibility Support Mode (CSM), “Legacy Mode”
- Not part of TianoCore, IBVs often sell this extra module, to add to EDK-II build system. SeaBIOS is open source version, IBVs sell commercial versions.
- Classes of systems (as framed by UEFI Forum):
  - Class 1: legacy OS only (BIOS, no UEFI)
  - Class 2: hybrid system (BIOS and UEFI)
  - Class 3: legacy-free (UEFI, no BIOS)
- With class 2 systems, user can boot either BIOS or UEFI, which complicates things (which partition format, MBR or GPT), what will OS try to use first, UEFI or BIOS, etc?

UEFI toolchains

- Legacy:
  - EDK (**E**fi **D**ev **K**it): used for EFI 1.x, replaced by EDK-II. Windows-based.
  - EFI Toolkit: EDK-based app dev kit, replaced by EADK.
- Current:
  - EDK-II: used for UEFI 2.x, replaced EDK. Multi-OS, multi-compiler-based.
  - UDK (**U**efi **D**ev **K**it): snapshot releases of subsets of EDK-II trunk.
  - EADK (**E**fi **A**pp **D**ev **K**it): contained within EDK-II. Like EDK did with EFI Toolkit.
  - EDK2-Buildtools: the tools used to build the EDK-II
- Alternative: GNU-EFI (non-Tianocore, limited use)

EDK-II: developer tools

- Besides Build, there are many specialty tools available in the UDK, with C/Python source, some are useful for security research:
  - BootSectImage, BPDG, EfiLdrImage, EfiRom, GenBootSector, GenCrc32, GenDepex, GenFds, GenFfs, GenFv, GenFw ,GenPage, GenPatchPcdTable, GenSec, GenVtf, LzmaCompress, PatchPcdValue, Spd2Dec, Split, TargetTool, TianoCompress, Trim, UPT, VfrCompile, VolInfo
- Eg, CHIPSEC bundles (Tiano,Lzma)Compress

TODO: Other containers

- Besides TE image format, UEFI has a few other containers formats. There's one for the system volume, and one for Option ROM flash volumes.
- For physical systems, IBVs (or OEMs, for business-class systems) generate the firmware image, IHVs create drivers that're integrated into that image, or on their OpROM volumes, or on the ESP.
- For virtual environments, UEFI has a format for use with QEMU/VirtualBox/etc: OVMF (next slide)

OVMF

- OVMF: Open Virtual Machine Firmware
- A system firmware image, as built by EDK-II/UDK build output, that can be read by QEMU.
- Useful if you want to learn UEFI, see how an OS behaves with UEFI, etc.
- With EDK-II dev tools, you can build with debug tracing enabled, or with full source-level debugging, no danger of bricking hardware.
- QEMU is primary target, but VirtualBox and recently Xen and KVM have added.

UEFI Boot Disk

- With UEFI systems, with Secure Boot disabled if you boot from a thumbdrive that has a GPT-based partition, with a FAT32-formatted ESP-based volume, containing a UEFI Shell in for that architecture in the proper subdirectory, then the system will boot into the UEFI Shell, so you can do UEFI work. This is useful if the OEM didn't provide UEFI Shell in their system.
- The CHIPSEC user documentation covers this for their UEFI Shell target.
- Being able to directly boot into UEFI is helpful, sometimes Linux (or Windows) gets in the way of HW/FW exploration.
- DUET is UEFI emulation for BIOS, and can be used to boot “UEFI” on a BIOS system, to use UEFI tools.

UEFI Applications

- Native UEFI Applications use the same API as drivers/services, except they exit when done.
- Examples: Forms Browser, Boot Loader, Shell
- UEFI Applications don't rely on any shell interfaces, just ConIn/ConOut/ErrOut.
- GNU-EFI toolchain also targets these, but has different init code than EDK-II toolchain-based apps.
- UEFI Shell is an example of a native app.

UEFI Shell Applications

- UEFI Shell Applications are UEFI Applications which link to the UEFI Shell's libraries, for more features, like File I/O instead of just stdio, and follow input/help conventions that shell/user presumes.
- Shell applications are separate binaries in the ESP, external to the shell binary.
  - Examples: cls.efi, eficompress.efi, ping.efi

UEFI 2.5: some new features

- ESRT (EFI System Resource Table), firmware updates
- Smart Card Reader and Smart card edge drivers (previously only User ID drivers)
- UEFI HTTP Boot (alternative to PXE), HTTP API and HTTP Helper API, UEFI TLS API (latter not in TianoCore yet, no HTTS!)
- Net additions: DNSv4, DNSv6, WiFi, EAP2, Bluetooth
- BMC/Service Processor Device Path, and ReSTful protocol, for DMTF Redfish, a new alternative to IPMI.
- System Prep Applications
- Platform recovery
- RAM Disk

SECTION: hardware models

- First, looking for existing hardware lifecycle management models. A few exist, varying levels. None covered firmware. The NIST 147/155 model is firmware-centric, but rather abstract.
- Then, try to integrate some existing tools (mainly CHIPSEC) into model, and some UEFI-centric tasks for the various models' states.
- The result is not a fully integrated model yet, but hopefully you can add some states and tasks to your current organization's model...

Subset of SysAdmin tasks

- Account provisioning
- Software, installing
- Recommending hardware/software for internal purchase
- Researching hardware/software from vendors
- Software, upgrading/updating
- Hardware, adding
- Hardware, updating (new ram/disk/cards/…)
- Monitoring system, health and security
- Hardware, reusing older hardware internally
- Hardware, recycling old hardware

Common Hardware lifecycle states

- Plan
- Purchase/Lease
- Deploy/Install
- Use
- Incident Response
- Maintain
- Upgrade
- Parts/Repair
- Extend
- Buy-Back/Trade-In
- Dispose/Recycle

Device travel

- Between Vendor and org: Ordered, Shipped, In-Repair, Returned
- Within org: Received, Inventoried, Installed, Decomissioned
- Leaves Org to Recycler: Disposed
- Normal path: Ordered, Shipped, Received, Installed, Decomissioned, Disposed
- Attackers have additional opportunities when device is not being actively monitored, especially when outside of your control.

Microsoft's PC lifecycle model

- 2008-era Enterprise PC lifecycle model, created by the “War-on-Costs team”
- 7-steps:
  - Plan
  - Buy
  - Deploy
  - Operate
  - Support
  - Upgrade
  - Retire

Step 1: Plan

- PC Strategy
  - IT Pro Time
  - BDM/TDM time (manager acronyms)
  - Consulting
- Demand Forecasting
- PC Selection
  - Hardware
  - Software stack
  - Peripherals
- Design Configuration

Step 2: Buy

- Purchase Process
  - Negotiation
    - OEM; Software; Accessories
  - Procurement & Vendor Management
- Hardware
  - Screen, OS, CPU, Motherboard, Floppy, Case/Power Supply, Keyboard/Mouse, Box, RAM, Disk Drives
  - Warranty
  - Fees: OEM Gross Margin, Upfront Recycling Fee, Sales Tax

Step 2: Buy (continued)

- Software Image
  - Productivity, Anti-Virus/Anti-Spam, Messaging & Groupware, Others
- Accessories
  - Cables, Power Supply, Battery, Bags, Cards, Hubs, Ergonomic Aids, Docs
- Delivery
  - Shipping, Box Disposal

Step 3: Deploy

- Logistics
  - Storage/Staging/Inventory
  - Asset Tagging & Admin
  - Inform end-users
  - Hardware Deployment
- Software Deployment
  - Configuration
  - Standardize PC images
  - Identity Management
  - IT configuration of PC
  - Network Configuration
- Data Migration

Step 4: Operate

- Facilities
  - Electricity, Cooling & Heating, Ergonomic Furniture, Consumables, Asset Insurance
- PC Security
- Software Updates
- Data Protection
  - Backups, Disaster recovery, Liability Insurance
- IT Admin
  - Audit, Asset Management

Step 5: Support

- Training
  - End-User Training
  - Training
- IT Support
  - Help Desk
  - On-site support
- Service Hardware
  - Depot
  - Break Fix
  - Warranty handling

Step 6: Upgrade

- Facilities
  - Equipment
  - Labor
  - Upgrade accessories
- Software Upgrades
  - New Software Installation
  - Packaging Software
- PC Transfer to New User
  - PC Pick-up
  - Archiving & Sanitizing HD
  - Reloading OS & Testing PC

Step 7: Retire

- PC Pick-up
- PC Re-sell Preparation
  - Archive
  - Sanitizing HD
  - Reloading OS
  - Testing PC
- Administrative Processing
- Shipping & Packing
- Residual Value

NIST 147/155 Platform Lifecycle

- Firmware security-centric
- 5 States:
  - Provisioning
  - Platform Deployment
  - Operations and Maintenance
  - Recovery
  - Disposition
- Next few slides have some of my UEFI-centric steps integrated with some of NIST's platform lifecycle states:

Before Purchase

- Vendor research, RFP, request CHIPSEC logs
- Make internal product recommendations
- Set company policy about hardware/firmware security
  - Eg, All new Intel UEFI systems must pass all current relavent CHIPSEC security tests.
- Set policy about what silicon/firmware security features (TPM, Secure Boot, etc.) must be enabled or not disabled
- Set policy about what firmware technologies (BIOS, UEFI, coreboot, U-Boot, etc.) can be used.
- Set policy about running unsigned drivers: some Secure Boot-enabled Linux distros allow unsigned dynamic post-boot Linux drivers to be loaded.

Hardening silicon

- Using/disabling silicon security tech (VT, TPM, TXT, TZ, TEE, Intel-ME, …)
- Using/disabling ports/hardware (WiFi, USB, Thunderbolt, BT, Ethernet, …)
- Smartcard authentication for boot/network access

Hardening firmware

- Secure/Trusted/Measured/Verified Boot
- Clarify system boot order
- WakeOn<N> features
- No unencrypted storage (Self-Encrypting Drives, SEDs, NVMe + TPM)
- Firmware/BIOS password
- Selecting UEFI over CSM/LegacyBoot

Hardening pre-OS software

- Limit 'pre-OS' software installed (UEFI drivers, services, apps, UEFI shell scripts.
- There are UEFI compilers for C, Python, Lua, Ruby, Rust, probably others which target UEFI these days.
- Check ESP, look for startup.nsh (unlike autoexec.bat, there can be N of these, in any directory, if in UEFI Shell's %PATH%)
- Look for *.efi, look on OEM/IHV flash for OpROMs and UEFI drivers
- Control/disable net access (IPMI, PXE, HTTP Boot, SMASH and DASH, Redfish, ...)

Provisioning and Deployment

- Maintain “golden” BIOS image for each platform
  - Save this when you first get it, and archive it
- Create and maintain configuration baseline
- Maintain a copy of the RTU, if applicable
- Register endpoint identity and BIOS integrity information in system inventory

Upon Purchase

- Test CHIPSEC, confirm logs did pass
- If CHIPSEC failed, return new system.
  - System is broken, security design, their QA team failed to run CHIPSEC, like UEFI Forum recommends that they do.
  - If you keep it, you're going to have to defend the indefensible.
- Dump initial rom image, backup (permanently)
- Add system to organization’s IT asset inventory

CHIPSEC - Initial use

- Look at CHIPSEC results when evaluating possible systems for purchasing.
- SAVE the test logs for later use!
- SAVE a binary copy of your BIOS image.
- CHIPSEC test results should help you understand whether certain attacks have been mitigated, useful to know before paying for expensive hardware.
- Varies due to relationships with the vendor and the ability to obtain/install BIOS updates.

CHIPSEC - ongoing

- Watch for CHIPSEC updates, and LUV (and LUV-live) updates.
- When CHIPSEC adds new security test modules, get a fresh copy of CHIPSEC, then re-run CHIPSEC on all hardware, to see if new tests detect any vulnerable systems.

Vendors: we need more data!

- 147 guidance on recording information about firmware is impossible without proper vendor data. Vendors need to start providing more detailed pre-sales technical information about the firmware they use. If enterprises don't ask for it, vendor's won't magically start providing it.
- OEMs: data about the system firmware you've chosen
  - Firmware technology (UEFI, coreboot, etc.), it's version, build-date, org of builder.
  - What modules (eg, Absolute's Persistence(R), other OEM add-ons beyond device drivers, network support, etc.) are added beyond Tianocore open source core.

Vendors: we need more data!

- IHVs: data about the drivers used:
  - What OpROMs/UEFI/other drivers are on flash, on the ESP, in the system firmware volume, or elsewhere. What drivers are needed to boot, how these drivers can be configured, enabled/disabled.
  - For each driver, just like OS-level driver, vendor name, unique ID(s), version and build-date information.
- Big enterprises: please help improve vendor situation, only you can influence them.

Firmware updates

- Once you've taken initial baseline of image from vendor, time to check for updates.
- With Windows 8+, Microsoft works with Windows OEMs to get firmware updates to Windows systems via Windows Update.
- UEFI 2.5's ESRT makes it easier for OS code to update firmware.
- In the past, OEMs currently often issue firmware updates in .ISOs, or worse yet, in Windows-only .EXEs, unusable on Linux/BSD.

Firmware Updates

- Linux has a new Firmware Update service: fwupd.org is an effort to unify Linux vendor's firmware updates to a single place for users to locate. Ask your Linux vendor to participate, or provide you with an easier/safer update mechanism. BSD-users, ask your BSD vendors for an equivalent to this Linux-centric service.
  - http://www.fwupd.org/
  - https://beta-lvfs.rhcloud.com/
  - https://github.com/hughsie/fwupd
  - freedesktop.org/software/appstream/docs/

Operation and Maintenance

- Perform updates using a change management process
- Use 800-147 authenticated update mechanism
- BIOS updates require new measurements to be registered
- Ensure proper configuration is maintained
- Monitor deployed BIOSes

Monitoring

- Check systems' firmware for network use
- UEFI has full network stack.
- With OOB management, traffic can happen when system is "powered off" with network (Ethernet, WiFi, Bluetooth, etc.).
- IPMI via WS-MAN (remotely) or Intel AMT (locally)
- Other "lights out management" tech beyond IPMI.
- PXE network boot
- DNS, DHCP, FTP, TFTP, IPsec, etc. use from within UEFI Shell
- DMTF SMASH-and-DASH

Management

- IPMI and DMTF Redfish:
  - Besides IPMI, now UEFI 2.5 has Redfish awareness
  - DMTF Redfish 1.0 just released recently.
  - HP already supports it, others are starting to.
  - Use WS-Man for IPMI, JSON for Redfish
- DMTF SMASH-and-DASH:
  - Server and Desktop management protocols for pre-OS interactions.
  - Generally not tied to core UEFI.
  - Some vendors (eg. AMD) use it.

Remote Booting

- UEFI remotely boots via both PXE and HTTP.
- PXE has been main method of remote booting
- Recently, with version 2.5, UEFI added "HTTP Boot".
- HTTP code is in Tianocore (but only HTTP, no HTTPS yet.)
- HP already supports it in some models.

CHIPSEC – post-Security Incident

- Look at CHIPSEC results after any kind of security event, losing physical control of a laptop or getting infected by malware, etc.
- After security event, compare CHIPSEC test results from earlier logs for differences, or if there is some change in a portion of the SPI flash image. Eg, if SPI flash protections were enabled before you lost physical control of the laptop but disabled afterward.
- Parse the SPI flash image with "chipsec_util.py decode" to do forensics on deltas.

Recovery

- Use only when authenticated updates are not possible
- Can be used to rollback from a buggy BIOS update
- If your OEM/IBV didn't provide a tool for you, you might have a problem here. Advanced firmware users may be able to recover some systems, with the right tools, if the right backup images were taken. Ask your vendor how this will be done before you purchase the hardware.

Disposition

- Reset BIOS configuration to defaults
- Remove passwords and organization-specific cryptographic keys
- Remove organization-specific customizations

Disposition

- Reset any TPM secrets
- Sanitize media
- UEFI has user data, User ID and Smartcard drivers, for PXE remote boot and IPsec use (CHAP auth used somewhere).
- Be clear on what user credentials are stored in your firmware, before you recycle it!
- Ask your vendor how to purge PII from their firmware product, for safe disposition.
  - Eg, Lenovo has a CD that resets the TPM data.

SECTION: tools

- UEFI Shell and commands
- UEFI port of Python
- BITS
- FWTS, FWTS-live
- CHIPSEC
- LUV, LUV-live
- UEFITool
- LAVA
- Other tools...

UEFI Shell's commands

alias attrib bcfg cd cls comp connect cp date dblk devices devtree dh disconnect dmem dmpstore dp drivers drvcfg drvdiag echo edit eficompress efidecompress else endfor endif exit for getmtc goto help hexedit if ifconfig load loadpcirom ls map memmap mkdir mm mode mv openinfo parse pause pci ping reconnect reset rm sermode set setsize setvar shift smbiosview stall time timezone touch type unload ver vol

- Most are console, a few (edit, hexedit) are 'curses'-style full-screen character mode.

Shell Scripts

- UEFI Shell supports shell scripts, in addition to to interactive user input
- Similar language to MS-DOS command.com (for, if, echo, pause, goto, pause, shift), except that FOR and IF have ends (endFor, endIf).
- UEFI Shell scripts have .nsh extension (“new shell”), not .bat or .cmd.
- Command line has 10 viewable args, %0 - %9, use SHIFT to see others, 256 maximum.
- Shells can invoke other shells, exit returns to parent. Errors returned in %lasterror%.

Startup.nsh

- Startup.nsh
- Shell's init script
- Similar to MS-DOS command.com's autoexec.bat.
- Shell looks for it in **MULTIPLE** locations: in dir where shell image was launched, or in PATH, not just root of boot device.
- Shell is only run: if available, if built with needed Level and Profile, and if invoked with args to permit this access.

Python

- Intel maintains a port of CPython 2.7x, ported to an EFI Shell Application. Separate from the Python includes in BITS.
- To use it, you must read the readme: to build it, and learn what modules are available, and where files are located, and other caveats.
- Once installed, easy to use Python inside UEFI!
- Intel's CHIPSEC tool is Python-based, and works under EFI (as well as Linux and Windows), their user docs on getting Python going under EFI is probably current the best available.
- Upcoming release: threads, UEFI structs, access to Boot Time Services, ...

BITS

- BIOS Implementation Test Suite (BITS)
- Intel-based project, created by Burt and Josh Triplett
- biosbits.org
- A bootable pre-OS environment for testing BIOSes and their initialization of hardware
- Enables access to EFI data structures, BIOS data structures, ACPI, CPU, PCI, PCIe as ring 0 in 32-bit ring 0 w/o any OS to hinder you

BITS

- Useful to check if BIOS is configured as Intel recommends, or for HW/FW research.
- It has batch and interactive (GRUB menu, interactive console, Python console) modes.
- Now includes Python, and exposes modules for many things
- Started as a BIOS-centric tool, but now also supports UEFI

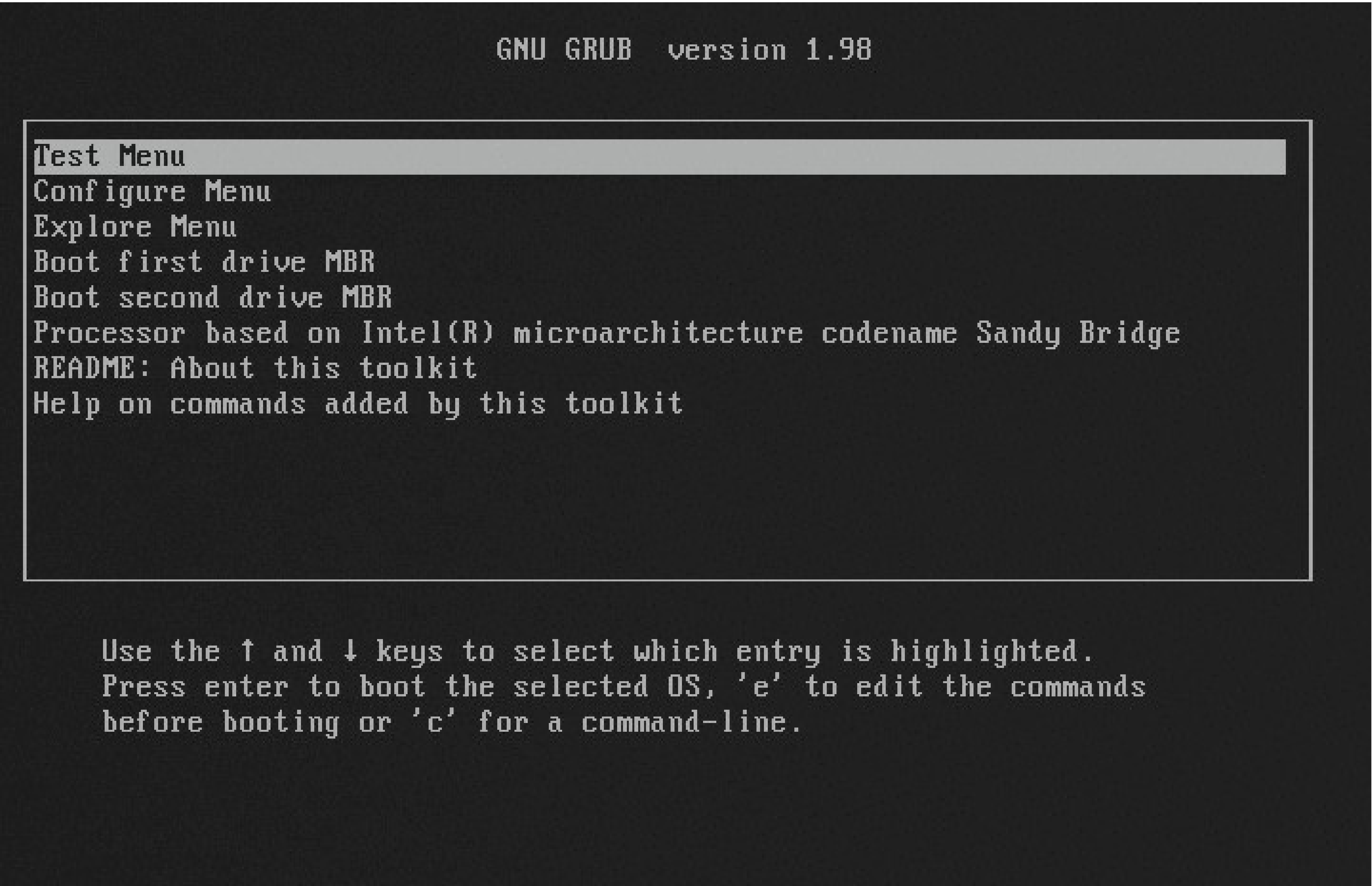
BITS - booting

- Builds for either 32-bit BIOS or 32-bit EFI.
- 64-bit EFI is planned for future release.
- Releases a .iso that boots on 32-bit BIOS or 32-bit EFI systems.
- Boots via the standard BIOS. Does not support native EFI booting; boot via the CSM.

BITS - feature categories

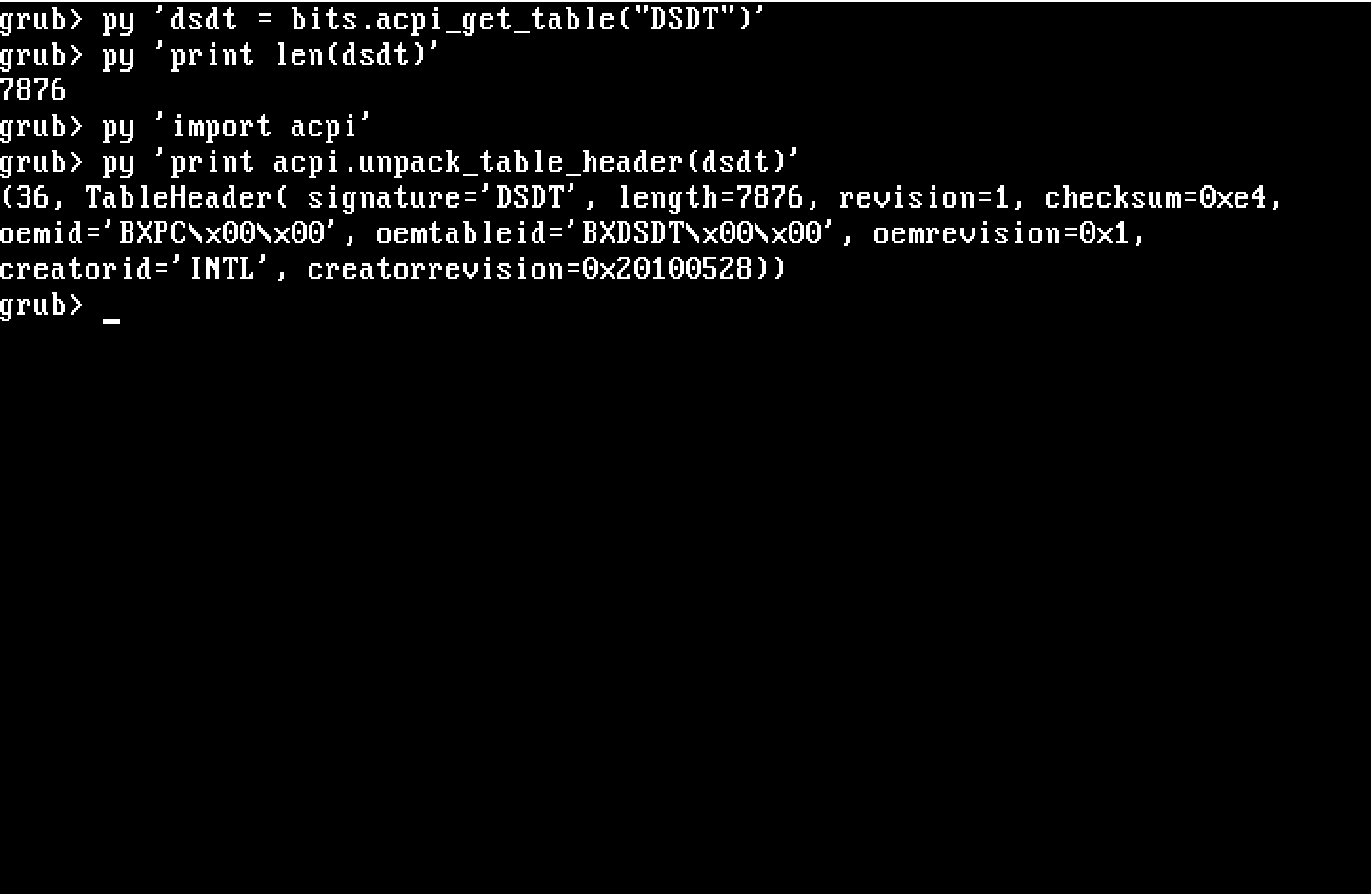
- Validate
  - run test suites to verify recommendations
- Configure
  - override BIOS using Intel reference code
- Explore
  - experimental tools and information gathering

Image source: http://biosbits.org/screenshots



```
                         GNU GRUB  version 1.98

Test Menu
Configure Menu
Explore Menu
Boot first drive MBR
Boot second drive MBR
Processor based on Intel(R) microarchitecture codename Sandy Bridge
README: About this toolkit
Help on commands added by this toolkit

    Use the ↑ and ↓ keys to select which entry is highlighted.
    Press enter to boot the selected OS, 'e' to edit the commands
    before booting or 'c' for a command-line.
```

Image source: http://biosbits.org/scripting



```
grub> py 'dsdt = bits.acpi_get_table("DSDT")'
grub> py 'print len(dsdt)'
7876
grub> py 'import acpi'
grub> py 'print acpi.unpack_table_header(dsdt)'
(36, TableHeader( signature='DSDT', length=7876, revision=1, checksum=0xe4,
oemid='BXPC\x00\x00', oemtableid='BXDSDT\x00\x00', oemrevision=0x1,
creatorid='INTL', creatorrevision=0x20100528))
grub> _
```

BITS Python EFI HelloWorld

```
import bits, efi

greeting = efi.encode_UCS2_mem("Hello
world!\r\n")

efi.call(efi.system_table.ConOut.OutputString,

        efi.system_table.ConOut._addr,

        bits.memory_addr(greeting))
```

FWTS

- FirmWare Test Suite (FWTS)
- Command line use, batch and interactive
- Created around 2010 by Canonical for Ubuntu QA
- Fwts are a package of tools you can install on an existing Ubuntu system.
- Test harness in C++, with many tests, and an optional curses-like user interface.
- Has both BIOS and UEFI tests.
- Multiple tests are useful for security research

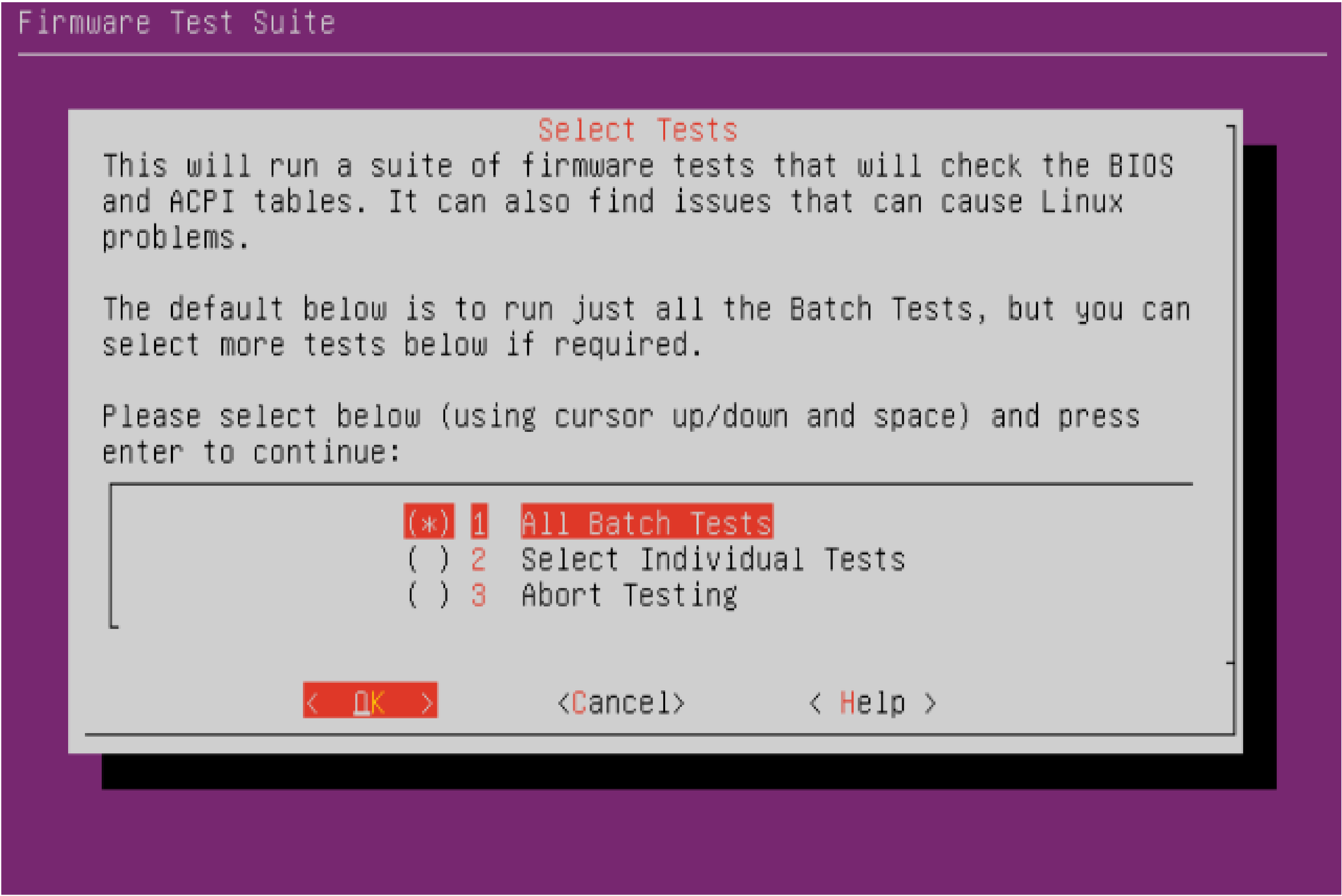
FWTS

- Useful to help find interesting Linux kernel warnings from firmware.
- New fwts user list for sysadmin-centric questions.
- launchpad.net/~firmware-testing-team
- fwts-announce@lists.ubuntu.com
- wiki.ubuntu.com/Kernel/Reference/fwts

FWTS-live

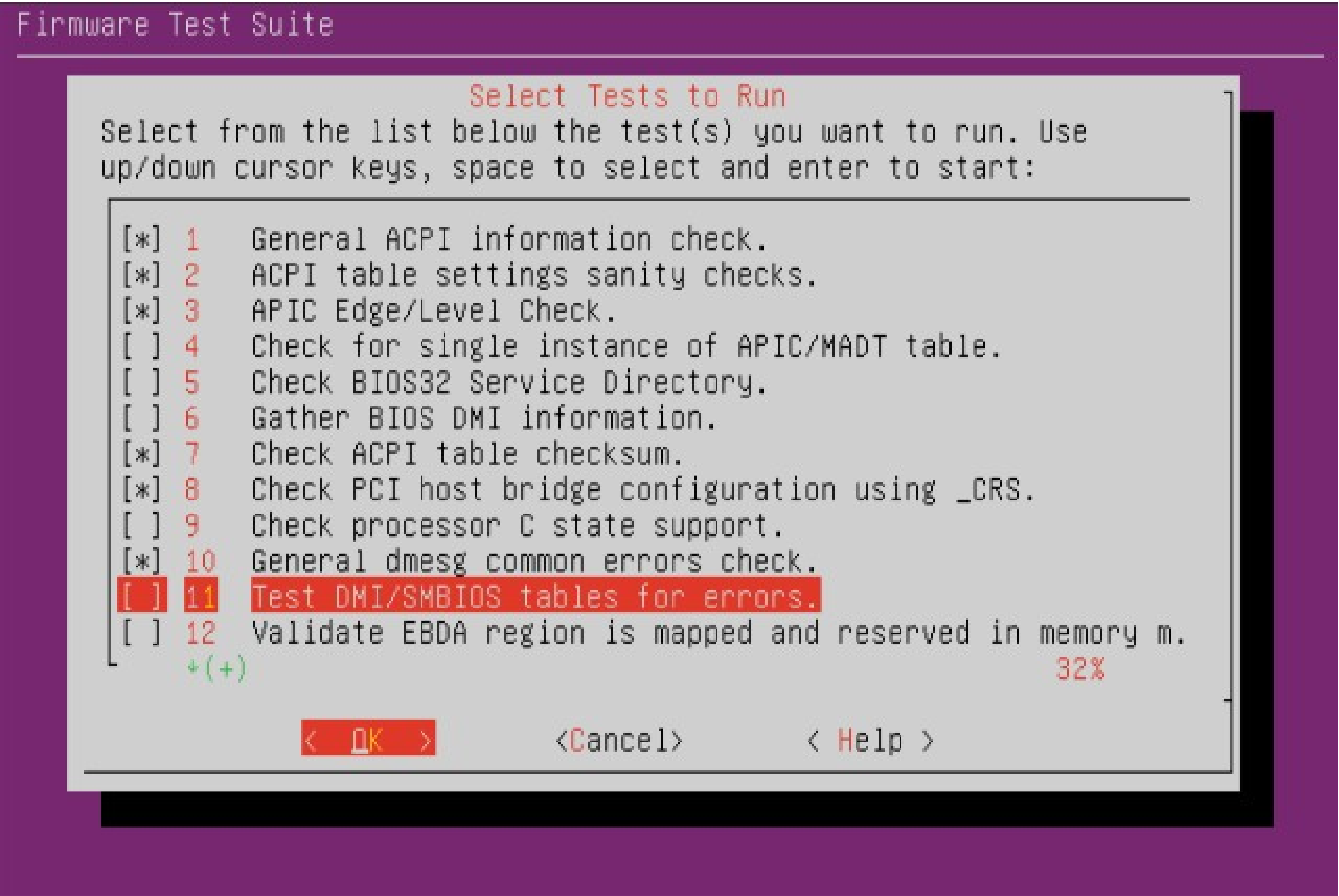
- Bootable USB image with Ubuntu and FTWS pre-installed, autoruns the curses-based menu UI for FWTS.
- Logs saved on USB for later review.
- wiki.ubuntu.com/HardwareEnablementTeam/Documentation/FirmwareTestSuiteLive

Image source: https://wiki.ubuntu.com/Hardware



```
Firmware Test Suite

                              Select Tests
 This will run a suite of firmware tests that will check the BIOS
 and ACPI tables. It can also find issues that can cause Linux
 problems.

 The default below is to run just all the Batch Tests, but you can
 select more tests below if required.

 Please select below (using cursor up/down and space) and press
 enter to continue:

                    (*) 1  All Batch Tests
                    ( ) 2  Select Individual Tests
                    ( ) 3  Abort Testing

              <  OK  >          <Cancel>          < Help >
```

Image source: https://wiki.ubuntu.com/Hardware



```
Firmware Test Suite

                         Select Tests to Run
 Select from the list below the test(s) you want to run. Use
 up/down cursor keys, space to select and enter to start:

  [*] 1    General ACPI information check.
  [*] 2    ACPI table settings sanity checks.
  [*] 3    APIC Edge/Level Check.
  [ ] 4    Check for single instance of APIC/MADT table.
  [ ] 5    Check BIOS32 Service Directory.
  [ ] 6    Gather BIOS DMI information.
  [*] 7    Check ACPI table checksum.
  [*] 8    Check PCI host bridge configuration using _CRS.
  [ ] 9    Check processor C state support.
  [*] 10   General dmesg common errors check.
  [ ] 11   Test DMI/SMBIOS tables for errors.
  [ ] 12   Validate EBDA region is mapped and reserved in memory m.
      ↓(+)                                                   32%

          <  OK  >           <Cancel>           < Help >
```

CHIPSEC

- Platform Security Assessment Framework
- “CHIPSEC is a framework for analyzing security of PC platforms including hardware, system firmware including BIOS/UEFI and the configuration of platform components. It allows creating security test suite, security assessment tools for various low level components and interfaces as well as forensic capabilities for firmware”
- github.com/chipsec/chipsec
- Newly-created CHIPSEC user discuss mailing list for sysadmin-centric discussions.

Intel CHIPSEC

- A Python-based (with some C, and a bit of assembly) open source project targetting hardware/firmware security researchers using Intel BIOS/UEFI systems.
- Contains two top-level programs
  - chipsec_utils.py, a collection of small tools to look at a specific FW/HW resource, for doing research
  - chipsec_main.py, a modular test harness, and a collection of existing security tests.
- python chipsec_main.py --help
- python chipsec_util.py help

CHIPSEC

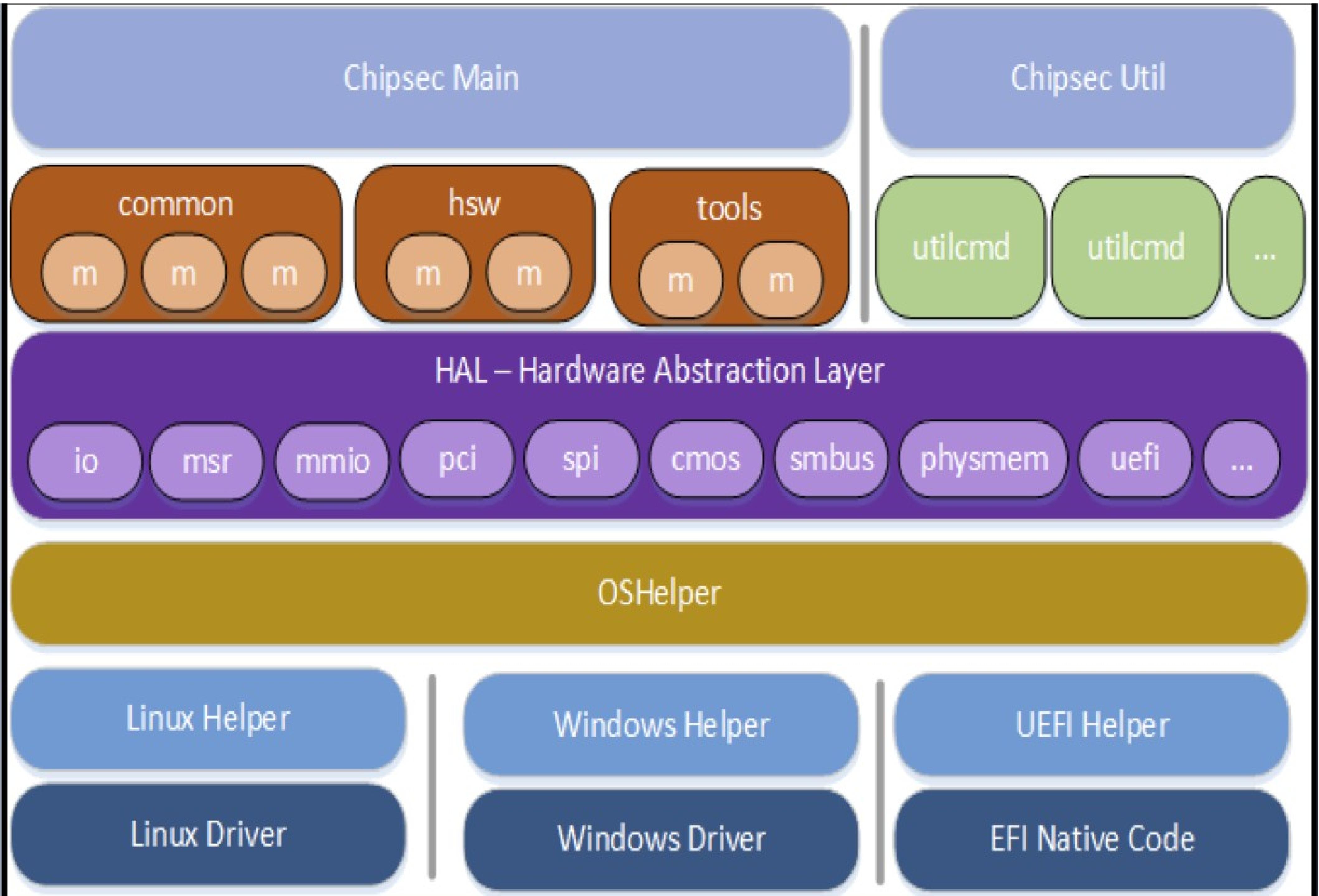
- The CHIPSEC python module can be run from a Python interactive shell, or used in other Python scripts. You can use chipsec as a library and write your own tools. Example:

```
import chipsec_main

chipsec_main._cs.init(True)

chipsec_main.load_module('chipsec/modules/common/bios_wp.py')

chipsec_main.run_loaded_modules()
```

CHIPSEC - targets

- Runs on three targets platforms:
  - Windows, with the help of a Windows kernel driver.
  - Linux, with the help of a Linux kernel driver.
  - UEFI, via UEFI Shell, loaded possibly via an 'EFI boot disk'.
- Non-targets:
  - OSes: FreeBSD, Android, Mac OS X, ...
    - Liveboot Linux-based distro like LUV-live
  - Chips: AMD, Itanium, ARM, ...
    - Linaro may be porting it to AArch64 as part of LUVos port

Image source: CHIPSEC documentation



CHIPSEC – some modules

- common.smm: SMRAM locking
- common.bios_kbrd_buffer: IOS keyboard buffer sanitization
- common.smrr: SMRR configuration
- common.bios_wp: BIOS protection
- common.spi_lock: SPI Controller locking
- common.bios_ts: BIOS interface locking
- common.secureboot.keys: Secure Boot keys
- common.secureboot.variables: Secure Boot variables

chipsec-util: live examples

- chipsec_util.py spi info
- chipsec_util.py spi dump rom.bin
- chipsec_util.py spi read 0x700000 0x100000 bios.bin
- chipsec_util.py uefi var-list
- chipsec_util.py uefi-var-read db D719B2CB-3D3A-4596-A3BC-DAD00E67656F db.bin

chipsec_util: offline examples

- chipsec_util.py uefi keys PK.bin
- chipsec_util.py uefi nvram vss bios.bin
- chipsec_util.py uefi decode rom.bin
- chipsec_util.py decode rom.bin

Using CHIPSEC

- Read the PDF included in the source release.
- Join the CHIPSEC mailing list on 01.org.
- Read the Intel Advanced Threat Research blog on Hacking Team malware for an example of how CHIPSEC is used to detect modern UEFI malware.
- Use chipsec_main.py to test security, and chipsec_util.py to dump/analyze ROMs.
- CHIPSEC works on modern Intel UEFI/BIOS systems. Does not work on old Intel chips, on AMD64 chips, or MIPS/ARM/POWER/etc chips.

CHIPSEC - enterprise caveat

- CHIPSEC uses a kernel driver on Linux (or Windows).
- The CHIPSEC team warns to NOT use the CHIPSEC driver on operational systems, as the driver ‘bypasses all OS security’.
- It should only be used in attacker-free test lab environments.
- Using CHIPSEC w/o a driver on local OS means rebooting into a Linux live CD, which has other issues…
- See warning.txt that ships with it.

LUV

- Linux UEFI Validation (LUV, LUVos, luvOS)
- A “Linux-readiness" distribution based on Yocto, from Intel
- “LUV also provides tests in areas not previously available, such as the interaction between the bootloader, Linux kernel and firmware.”
- “This integrated solution is invaluable when testing components that require cooperation across multiple runtime phases. It allows the luvOS to test whether UEFI capsules work correctly across a reboot, for instance.”

LUV coverage

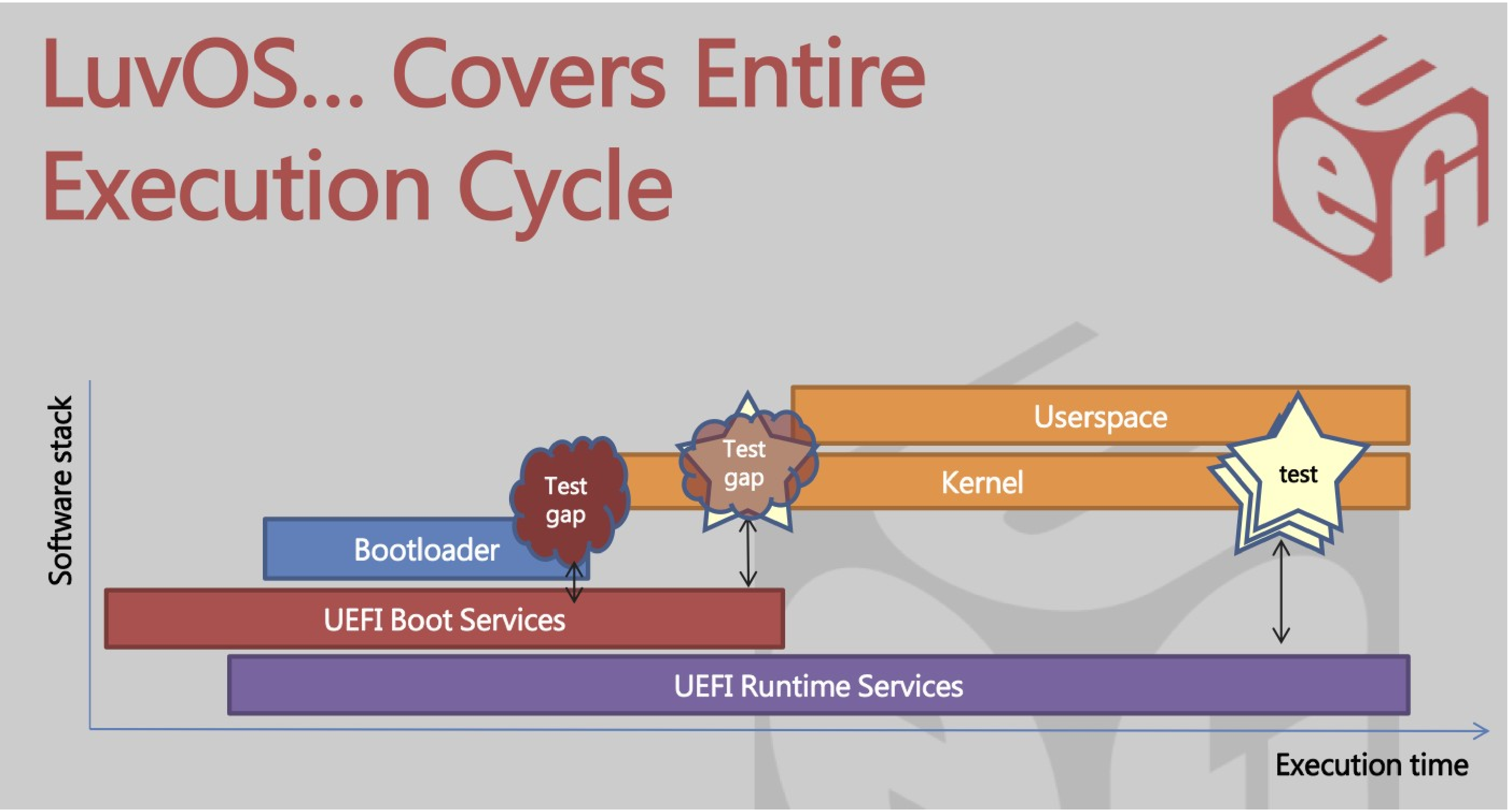


Image source: LinuxCon 2014 talk by Brian Richardson (Intel) and Alex Hung (Canonical), “UEFI Test Tools for Linux Developers”

LUV

- Contains FWTS, CHIPSEC, BITS, and other UEFI-centric HW/FW QA tools. Nearly a “one-stop shop” for firmware security tools!
- Fork of Yocto’s Poky, which includes the meta-luv layer, intended to make it easy to build LUV, everything in a single Git repo.
- LUV's meta-luv contains all of LUV’s additional features. You can use this package in your own Yocto-based distro. Or you can use their LUV-live (next slide)
- Community: Git hosted code and issue tracking, mailing list, IRC.
- 01.org/linux-uefi-validation

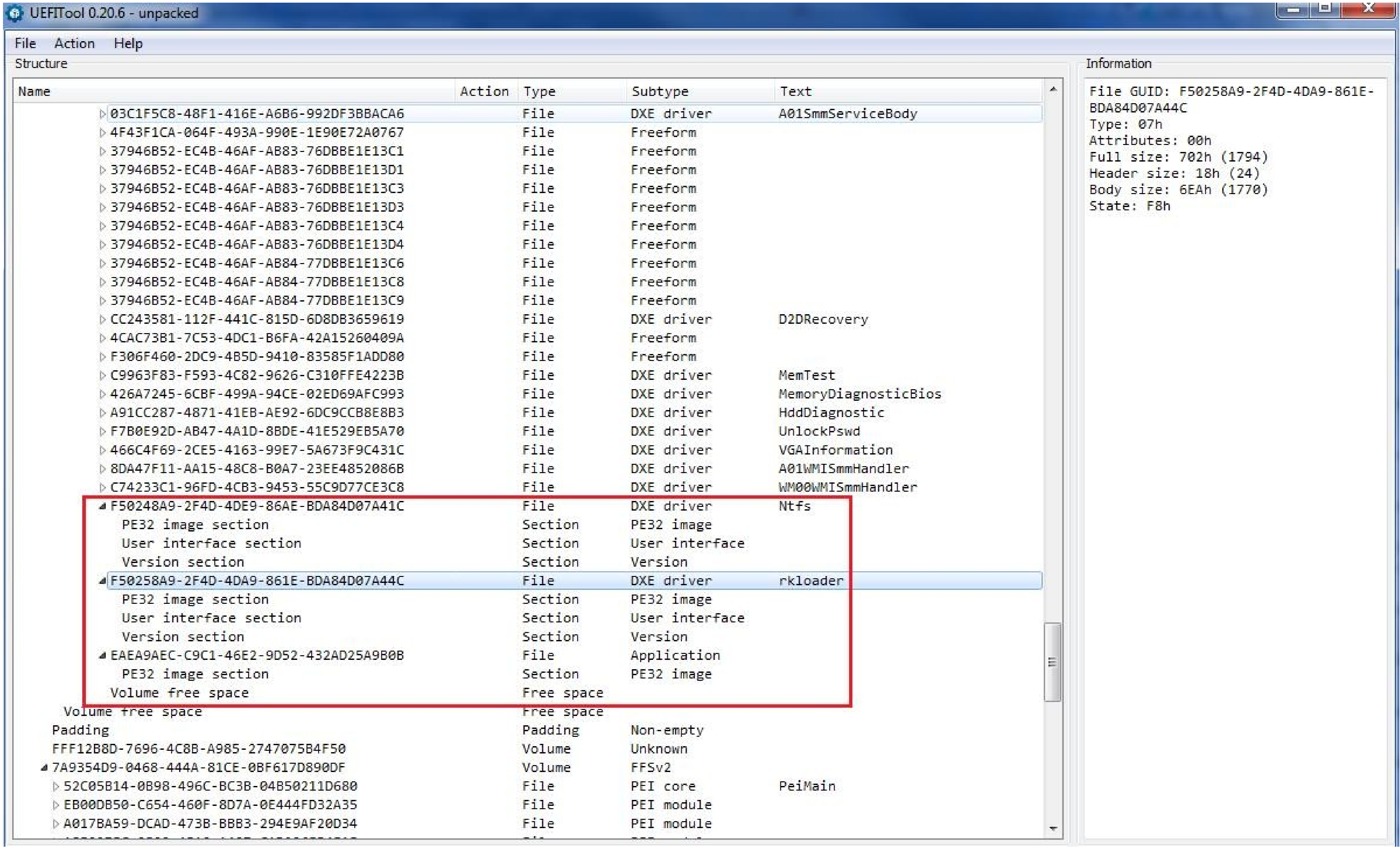
LUV-live

- A Yocto-based live-boot style distro for LUV.
- Recently includes Secure Boot support.
- Check latest CHIPSEC release, then check if that release is in LUV yet.
- If you have a local build setup, you can build a fresh live solution with the latest BITS, CHIPSEC, FWTS, etc. and add your own tests.

UEFITool

- github.com/LongSoft/UEFITool
- C++-based, Qt GUI app, cross-platform.
- Examines UEFI ROMs, shows different modules by GUID, lets you search for strings and GUIDs, lets you extract and insert modules from images.
- Originally written by Nikolaj Schlej

Image credit: Intel ATR blog



```
Name                                              Action  Type        Subtype          Text
    ▷ 03C1F5C8-48F1-416E-A6B6-992DF3BBACA6                File        DXE driver       A01SmmServiceBody
    ▷ 4F43F1CA-064F-493A-990E-1E90E72A0767                File        Freeform
    ▷ 37946B52-EC4B-46AF-AB83-76DBBE1E13C1                File        Freeform
    ▷ 37946B52-EC4B-46AF-AB83-76DBBE1E13D1                File        Freeform
    ▷ 37946B52-EC4B-46AF-AB83-76DBBE1E13C3                File        Freeform
    ▷ 37946B52-EC4B-46AF-AB83-76DBBE1E13D3                File        Freeform
    ▷ 37946B52-EC4B-46AF-AB83-76DBBE1E13C4                File        Freeform
    ▷ 37946B52-EC4B-46AF-AB83-76DBBE1E13D4                File        Freeform
    ▷ 37946B52-EC4B-46AF-AB84-77DBBE1E13C6                File        Freeform
    ▷ 37946B52-EC4B-46AF-AB84-77DBBE1E13C8                File        Freeform
    ▷ 37946B52-EC4B-46AF-AB84-77DBBE1E13C9                File        Freeform
    ▷ CC243581-112F-441C-815D-6D8DB3659619                File        DXE driver       D2DRecovery
    ▷ 4CAC73B1-7C53-4DC1-B6FA-42A15260409A                File        Freeform
    ▷ F306F460-2DC9-4B5D-9410-83585F1ADD80                File        Freeform
    ▷ C9963F83-F593-4C82-9626-C310FFE4223B                File        DXE driver       MemTest
    ▷ 426A7245-6CBF-499A-94CE-02ED69AFC993                File        DXE driver       MemoryDiagnosticBios
    ▷ A91CC287-4871-41EB-AE92-6DC9CCB8E8B3                File        DXE driver       HddDiagnostic
    ▷ F7B0E92D-AB47-4A1D-8BDE-41E529EB5A70                File        DXE driver       UnlockPswd
    ▷ 466C4F69-2CE5-4163-99E7-5A673F9C431C                File        DXE driver       VGAInformation
    ▷ 8DA47F11-AA15-48C8-B0A7-23EE4852086B                File        DXE driver       A01WMISmmHandler
    ▷ C74233C1-96FD-4CB3-9453-55C9D77CE3C8                File        DXE driver       WM00WMISmmHandler
    ◢ F50248A9-2F4D-4DE9-86AE-BDA84D07A41C                File        DXE driver       Ntfs
        PE32 image section                                Section     PE32 image
        User interface section                            Section     User interface
        Version section                                   Section     Version
    ◢ F50258A9-2F4D-4DA9-861E-BDA84D07A44C                File        DXE driver       rkloader
        PE32 image section                                Section     PE32 image
        User interface section                            Section     User interface
        Version section                                   Section     Version
    ◢ EAEA9AEC-C9C1-46E2-9D52-432AD25A9B0B                File        Application
        PE32 image section                                Section     PE32 image
      Volume free space                                   Free space
   Volume free space                                      Free space
  Padding                                                 Padding     Non-empty
  FFF12B8D-7696-4C8B-A985-2747075B4F50                    Volume      Unknown
◢ 7A9354D9-0468-444A-81CE-0BF617D890DF                    Volume      FFSv2
   ▷ 52C05B14-0B98-496C-BC3B-04B50211D680                 File        PEI core         PeiMain
   ▷ EB00DB50-C654-460F-8D7A-0E444FD32A35                 File        PEI module
   ▷ A017BA59-DCAD-473B-BBB3-294E9AF20D34                 File        PEI module
```

```
File GUID: F50258A9-2F4D-4DA9-861E-
BDA84D07A44C
Type: 07h
Attributes: 00h
Full size: 702h (1794)
Header size: 18h (24)
Body size: 6EAh (1770)
State: F8h
```

LAVA

- Linaro Automated Validation Architecture (LAVA)
- “A continuous integration system for deploying operating systems onto physical and virtual hardware for running tests. Tests can be simple boot testing, bootloader testing and system level testing, although extra hardware may be required for some system tests. Results are tracked over time and data can be exported for further analysis.”
- LAVA has 2 components, the lava-server (Django web app) and the lava-dispatcher.
- Useful for updating firmware and doing “Pre-OS” app tests on devices which're supported by LAVA.
- Targets QEMU, not only physical ARM hardware.
- validation.linaro.org

Other Tools…

Standard RE tools….

- IDA Pro, with multiple UEFI-centric plugins
- Radare2, Angr, Capstone, Yara, BinView, BinDiff, etc.
- Multiple PE exe tools work with UEFI TE images.

UEFI Firmware Parser

- Python tool to parse UEFI images, for forensic/malware analysis.
- fv_parser.py, Firmware Volume Parser
- github.com/theopolis/uefi-firmware-parser

coreboot's BIOS Extract

- Multiple OEM/IHV-specific tools to extract BIOS (and UEFI) ROMs, and analyze them.
- Coreboot.org
- Also, coreboot's FlashROM.
- Flashrom.org

Thundergate

- Python script that lets you do most anything with PCI Option ROMs
- firmwaresecurity.com/2015/09/07/tool-thundergate/
- https://github.com/sstjohn/thundergate
- http://thundergate.io/

Legbacore's T2E Integrity Check

- LegbaCore's Apple Thunderbolt to Ethernet adapter Option ROM checking tool
- http://www.legbacore.com
- https://github.com/legbacore/t2e_integrity_check

BIOS_Diff.py

- MITRE Copernicus is a mostly a Windows-centric tool.
- It also includes an includes a portable open source Python tool, useful to comparing two ROMs – different versions from one system, or from 2 separate but same batch of systems – for changes.
- firmwaresecurity.com/2015/05/21/tool-mini-review-bios_diff-py/

BIO Unpack

- BIO Unpack
- https://gist.github.com/XVilka/8163272

ReadPhysMem

- ReadPhysMem
  - firmwaresecurity.com/2015/09/03/tool-readphysmem/

DumpFlash

- DumpFlash
  - firmwaresecurity.com/2015/08/24/tool-dumpflash/

Xpwntool-light

- Xpwntool-light
- https://github.com/sektioneins/xpwntool-lite
- http://github.com/planetbeing/xpwn
- firmwaresecurity.com/2015/08/05/tool-mini-review-xpwntool-lite/

UEFI Reverse

- UEFI Reverse
- https://github.com/jethrogb/uefireverse

UEFI xmod

- UEFI Xmod
- https://github.com/danse-macabre/uefi_xmod

Untermensch's UEFI injector

- UEFI Windows Secure Boot injector tools
- firmwaresecurity.com/2015/07/24/tool-mini-review-untermensch-uefi-windows-secure-boot-injection-tools/
- https://github.com/untermensch/WindSLIC
- https://github.com/untermensch/Injector
- https://github.com/untermensch/FirmwareModule

UEFI Spider

- A set of Python-based web crawling scripts that gather many firmware binaries from vendor sites.
- https://github.com/theopolis/uefi-spider

Python EFI Compressor

- As well, Edk2 had TianoCompress and LzmaCompress. The UEFI spec has compress implementation code in it's appendixes.
- https://github.com/sstjohn/python-eficompressor
- https://github.com/linearregression/python-eficompressor

Android VTS

- Android Vulnerability Test Suite
- https://github.com/nowsecure/android-vts
- Android-specific, not UEFI-centric. If using Android-IA, using Android VTS + CHIPSEC may be useful.

More Tools: freeware

- The firmware modding community has a few non-open source freeware binaries, for MS-DOS, UEFI, and Windows
- UEFI BIOS Updater (UBU) and UBU Helpers
  - UBU is freeware, UBU-Helpers is open source.
  - firmwaresecurity.com/2015/07/12/tool-review-ubu-helpers/
  - http://www.win-raid.com/t154f16-Tool-Guide-News-quot-UEFI-BIOS-Updater-quot-UBU.html
  - https://github.com/LongSoft/UBU-helpers

More Tools: Freeware

- RW Everything
  - firmwaresecurity.com/2015/08/14/tool-mini-review-rweverything/
  - http://rweverything.com/
- Read Universal utility (RUEXE)
  - http://ruexe.blogspot.de/
  - https://code.google.com/p/ru-uefi/
  - firmwaresecurity.com/2015/08/13/tool-mini-review-read-universal-utility-ruexe/

Other Tools

- There are a few dozen other speciality tools, many specialty for a specific vendor model and firmware type.
- Google Search, Twitter, and Github Search are your friends. :-)
- I've been told there are a few good modding communities that're helpful with tools.
- You'll probably also have to spend time in OEM/IHV web support forums. :-(
- Whenever I find a new tool, I post a blog entry on it: firmwaresecurity.com/feed

SECTION: More Information

More Info: tools

- LUV[-live]: https://01.org/linux-uefi-validation/downloads/luv-live-image
- CHIPSEC: https://github.com/chipsec/chipsec
- FWTS[-live]: http://fwts.ubuntu.com/fwts-live/
- UEFItool: https://github.com/LongSoft/UEFITool
- BITS: http://biosbits.org/

More Info: OEM-specific

- Sorry, “Exercise left for reader”
- Study your OEM support resources and specifications.
- You need to understand all of the FW issues for all of the different systems in your organization.

OEM-specific tools

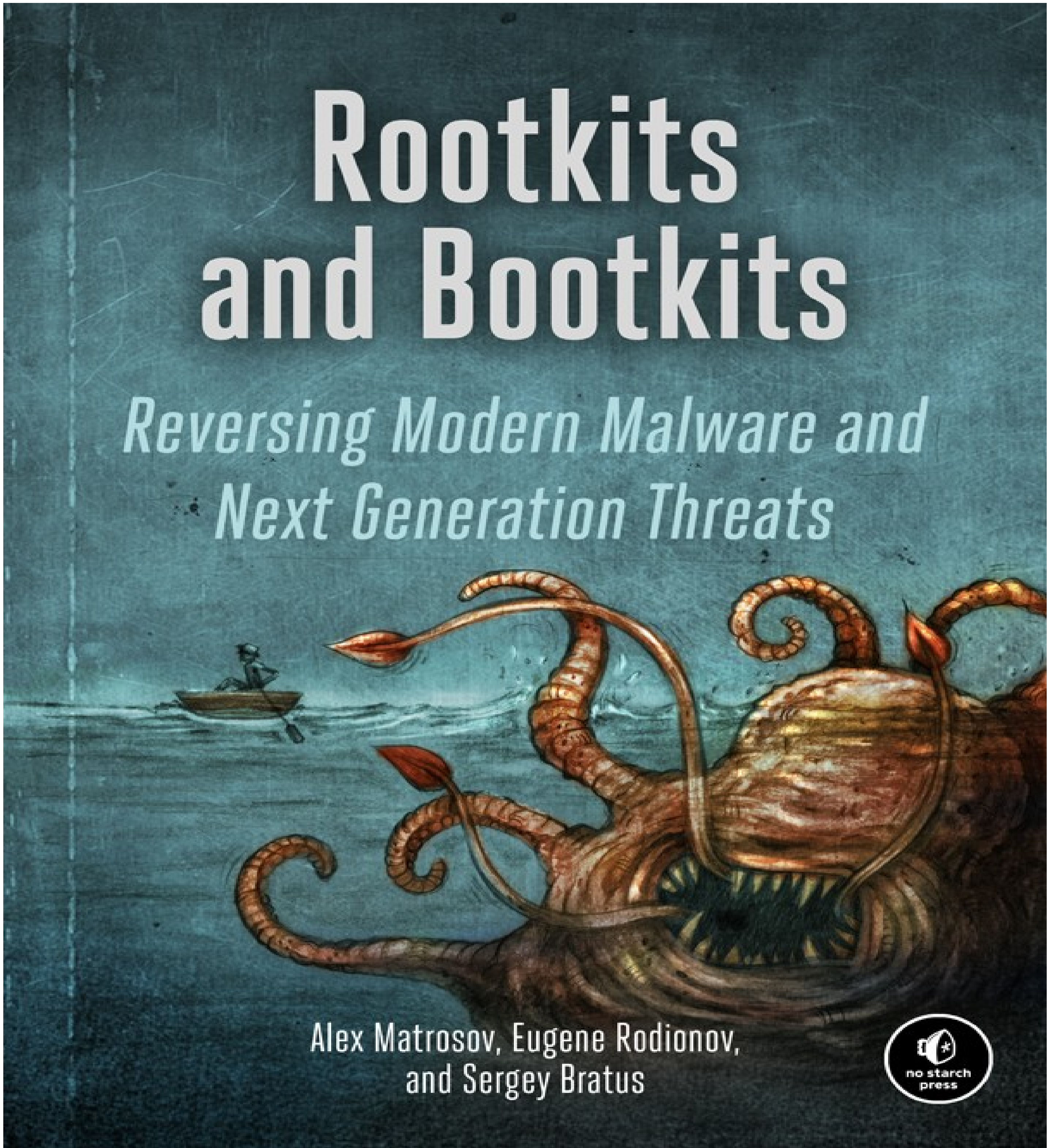
- Examples:
  - Lenovo has a UEFI diagnostic CD, and a TPM reset CD.
  - HP has a set of UEFI diagnostic tools.
  - Apple has a firmware reset CD.
  - Cisco has a set of UEFI diagnostic tools.

More Info: IBV resources

- IBVs (AMI, Insyde, Phoenix, Byosoft, etc.) often have tools to help with diagnostics and updates.
- But they’re often only available to OEMs for bundling, not end-users.
- Check your IBV to see if they have any tools to help you.
- OEMs often don't use an IBV for their business-class systems, and do the IBV work in-house, along with any related tools, so double-check OEM for firmware tools.

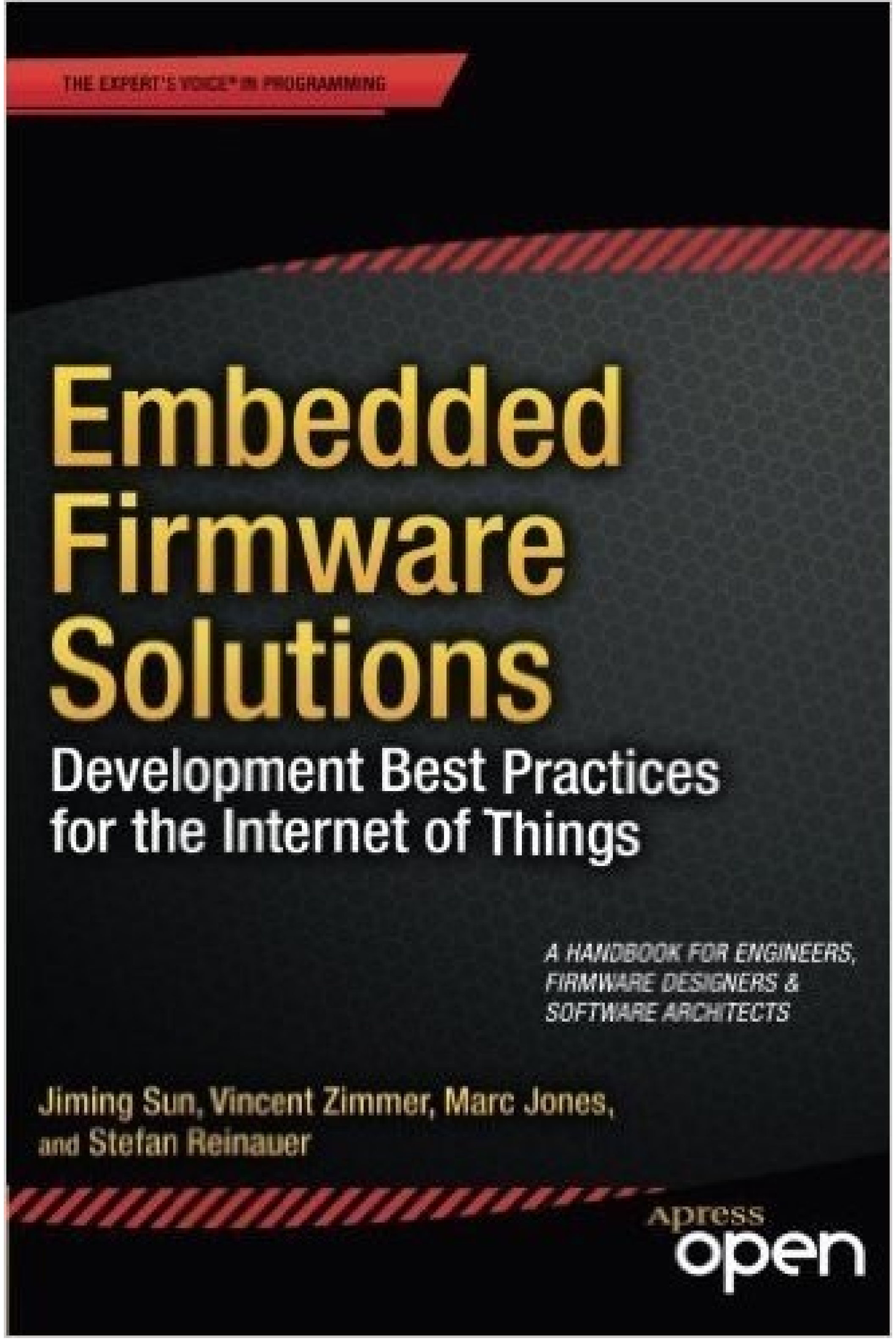
More Info: books

- Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats
- Alex Matrosov, Eugene Rodionov, and Sergey Bratus
- ISBN: 978-1-59327-716-1
- No Starch Press
- https://www.nostarch.com/rootkits
- Spring 2016 (some ebook chapters available now)



More Info: books

- Embedded Firmware Solutions: Development Best Practices for the Internet of Things
- Jiming Sun, Marc Jones, Stefan Reinauer, Vincent Zimmer
- APress Media
- ISBN 978-4842-0071-1
- 2015
- http://www.apress.com/9781484200711


THE EXPERT'S VOICE® IN PROGRAMMING

Embedded Firmware Solutions

Development Best Practices for the Internet of Things

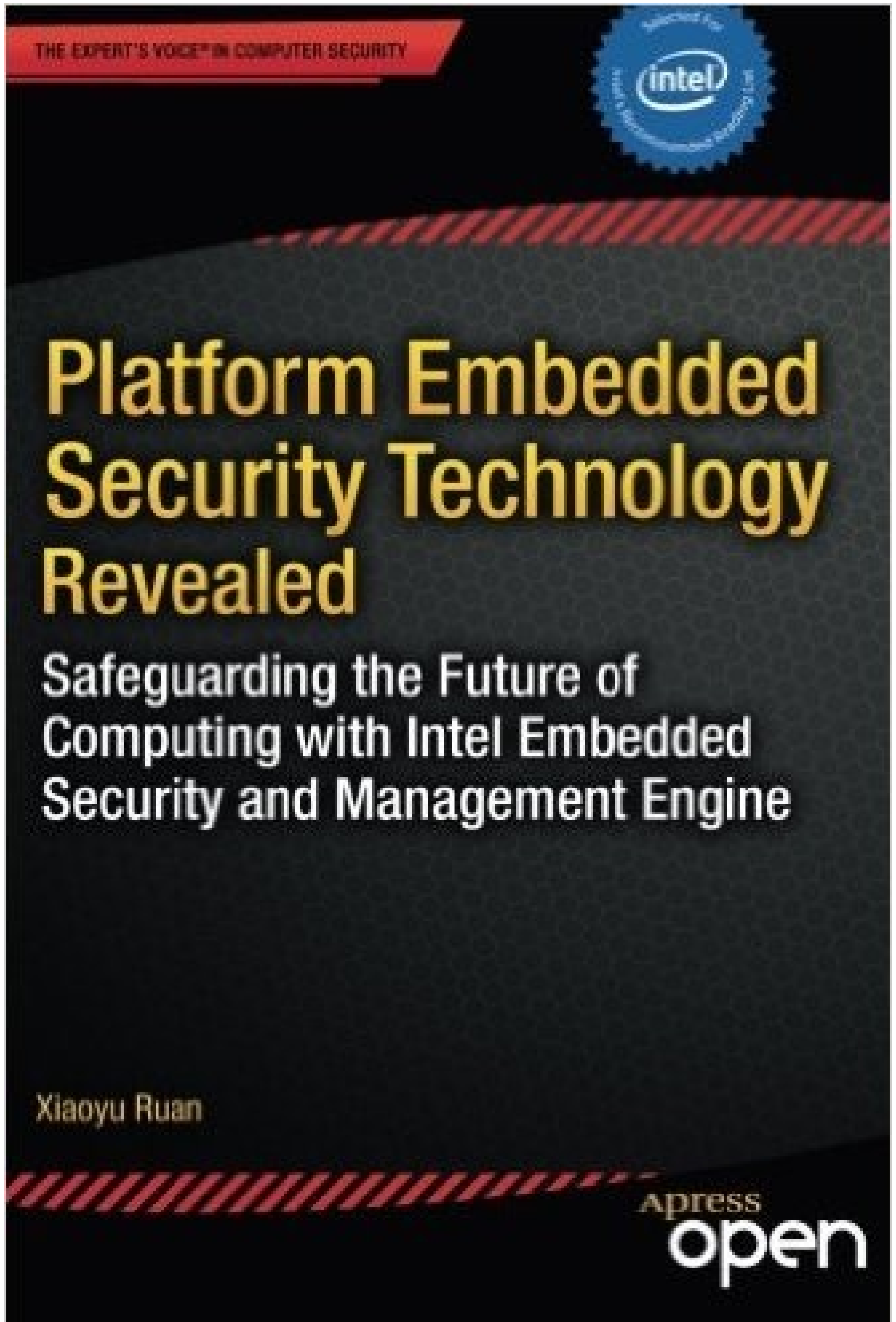
A HANDBOOK FOR ENGINEERS, FIRMWARE DESIGNERS & SOFTWARE ARCHITECTS

Jiming Sun, Vincent Zimmer, Marc Jones, and Stefan Reinauer

Apress open

More Info: books

- Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine
- Xiaoyu Ruan
- APress Media
- ISBN13: 978-1-4302-6571-9
- http://www.apress.com/9781430265719
- 2014



THE EXPERT'S VOICE® IN COMPUTER SECURITY

Platform Embedded Security Technology Revealed

Safeguarding the Future of Computing with Intel Embedded Security and Management Engine

Xiaoyu Ruan

Apress open

More Info: online

- Bibliography of x86 BIOS/UEFi exploits: usually found on the last slides of recent talks from Intel CHIPSEC or LegbaCore researchers.
- Conference proceedings from the big security cons (eg, DEF CON)
- Many leading firmware security researchers use Twitter; Jacob Torrey maintains the list:
  - https://twitter.com/JacobTorrey/lists/firmware-security
- FirmwareSecurity.com/feed (disclaimer: my blog)

More Info: Twitter

- Jacob Torrey's firmware-security list on Twitter
- firmwaresecurity.com/2015/08/15/twitters-firmware-researcher-community-list/
- https://twitter.com/JacobTorrey/lists/firmware-security/members

More Info: SANS

- SANS: Implementing a PC Hardware Configuration (BIOS) Baseline
- David R. Fletcher Jr (author), Rich Graves (advisor)
- 2013
- Like this presentation, this document focuses on NIST 147 firmware lifecycle integration into enterprises!
- http://www.sans.org/reading-room/whitepapers/basics/implementing-pc-hardware-configuration-bios-baseline-34370

More info: misc

- Firmware Security Docs
  - https://github.com/robguti/firmware_security_docs
- Firmware Vault (collection of Apple ROMs)
  - https://github.com/gdbinit/firmware_vault
- Firmware.RE (upload ROM to site for analysis)
  - http://firmware.re
- Matthew Garrett's Secure Boot best practices:
  - firmwaresecurity.com/2015/09/23/matthew-garrett-publishes-secure-boot-best-practices/

More Info: training

- Multiple security firms offer security training for hardware/firmware, on-site or as pre-conf training at the big security conferences (DEF CON, etc.). Some trainers:
  - LegbaCore: LegbaCore.com
  - Joe Fitzpatrick: SecuringHardware.com
  - Joe Grand: GrandIdeaStdios.com
  - ...

More Info: UEFI advisories

- Track UEFI vulnerabilities, via UEFI Forum web site, and/or TianoCore developer web site.
- UEFI Forum's security web site
  - uefi.org/security
- UEFI Forum's security web site, announce list, and PDFs of advisories, for TianoCore
  - Web: tianocore.sourceforge.net/wiki/Security
  - List: tianocore-security@lists.sourceforge.net
  - PDFs: sf.net/projects/edk2/files/Security_Advisory

More Info: vendor advisories

- Intel: Intel Product Security Center (for systems with Intel-based firmware), web site and announce list (the latter does not work for me).
  - https://security-center.intel.com/default.aspx
- Look to your OEM/IBV for any firmware update mechanism(s) [if available]. Uefi.org's security web site has some email/web sites for a few vendors. See previous slide for UEFI Forum's security page, it has links to some OEMs and other UEFI Forum vendors.

NIST SCAP: barely useful

- NIST SCAP has no OVAL definitions for firmware.
- So any results only found via FTS, if report happened to include valid firmware technology terms. MANY firmware-centric CVEs can't be found by useful terms.
- Too bad NIST 147/155 didn't also recommend that NIST SCAP to also be updated, then we could use CM/SIEM tools, like multiple NIST docs recommend, for firmware, like NIST 147 recommends. IETF SACM also needs firmware metdata coverage.

More Info: Help!!

- First escalate to your normal CERT, they need to understand how to deal with firmware security.
- If Intel-based, they have a firmware security contact.
- If Intel-based, the CHIPSEC team has a contact, and might be interested in the malware.
- If UEFI-based, Tianocore has a firmware security contact.
- For OEMs, you need to find your own; results vary by vendor.

More Info: Help!!

- If public disclosure is not an issue, the Twitter list of firmware security researchers has some of the smartest people for this topic, if it is new and interesting, someone may help you out.
- If you need **professional** for-fee help, try:
  - LegbaCore
    - http://www.legbacore.com/Services.html
  - Invisible Things Lab
    - http://invisiblethingslab.com/itl/Services.html

Calls To Action: learn firmware security

- Read NIST BIOS specs, 147 (esp section 3.2), 147b, and 155. Integrate their lifecycle model with your organizations model.
- Learn to use CHIPSEC, FWTS, BITS, LUV, UEFITool, UEFI Firmware Parser, and other firmware tools.
- Learn to save, archive, diff, and examine your ROMs for changes and strings.
- Request CHIPSEC log results from vendors before purchasing them. Test newly-purchased hardware with CHIPSEC, return broken (insecurely designed) system to vendor if it fails.

Calls To Action: learn firmware security

- Watch Intel firmware, EDK-II Security Advisories, and vendor-specific firmware announcements.
- Watch CHIPSEC for new security test modules. When new tests arrive, re-run fresh CHIPSEC on all systems. As well, watch for new FWTS tests, and more generally, any new/udpated tools.
- Watch for new HW/FW vulnerabilities at security conferences.

Calls To Action: improve ecosystem

- Get Linux OEMs to support fwupd.org
- Get OEMs/IHVs to include detailed firmware-level pre-sales technical information, including CHIPSEC test results, ACPI table info, firmware info, etc.
- Intel aside, ask your other chip vendors what firmware vulnerability assessment tools they have, or when they're going to port CHIPSEC to their platform.
- Update sysadmin best practices and job-training certifications to include firmware-level security.

ACKnowledgements

- Vincent of Intel UEFI; John and Yuriy of Intel CHIPSEC; Corey and Xeno of LegbaCore; Nikolaj of UEFITool, Matt and Ricardo of LUV; Gary of AMD.
- Most from the Twitter firmware security researchers group. Many from the EDK2-devel, FWTS, CHIPSEC, LUV, and other firmware- and security-related mailing lists.
- Many at DC206 and BlackLodgeResearch.org.

EOF

- Questions?
- Discussion…
  - What other tools are you using today?
  - What is missing from model?
  - …?
- Thanks for attending!