Building Your Linux Firmware Security Toolkit

Lee Fisher
LinuxFestNorthWest.org
Bellingham.WA.US
2015-04-26

Last Revised: 2015-04-25 22:18

Agenda

- History/background
  - BIOS, NIST, negative rings of protection
- UEFI architecture (brief)
- Testing/developing UEFI (brief)
  - Shell, Python, EDK-II/UDK/EADK, UEFI Apps, QEMU, MinnowBoard, ...
- Firmware security tools
  - BITS, CHIPSEC, FWTS, LUV, LAVA, ...
- More information

Learning Goals

- What open source tools are available to for diagnosing Intel- and ARM-based Linux systems.
- Emphasizing UEFI-based system, not focusing on Coreboot- or BIOS-based systems.
- Focus: UEFI Shell and commands, EDK-II developer toolchain, CHIPSEC, LUV, FWTS, BITS, LAVA, a few other EFI security tools.

Prerequirements

- You are a Linux system administrator, developer, or security researcher.
- You know architectural fundamentals of: Intel hardware, IBM PC BIOS/OpROM firmware. ARM hardware also helpful.
- You can use: bash/command.com/cmd.exe shell(s) and write scripts for them. For more advanced use, Python and C language skills are needed.

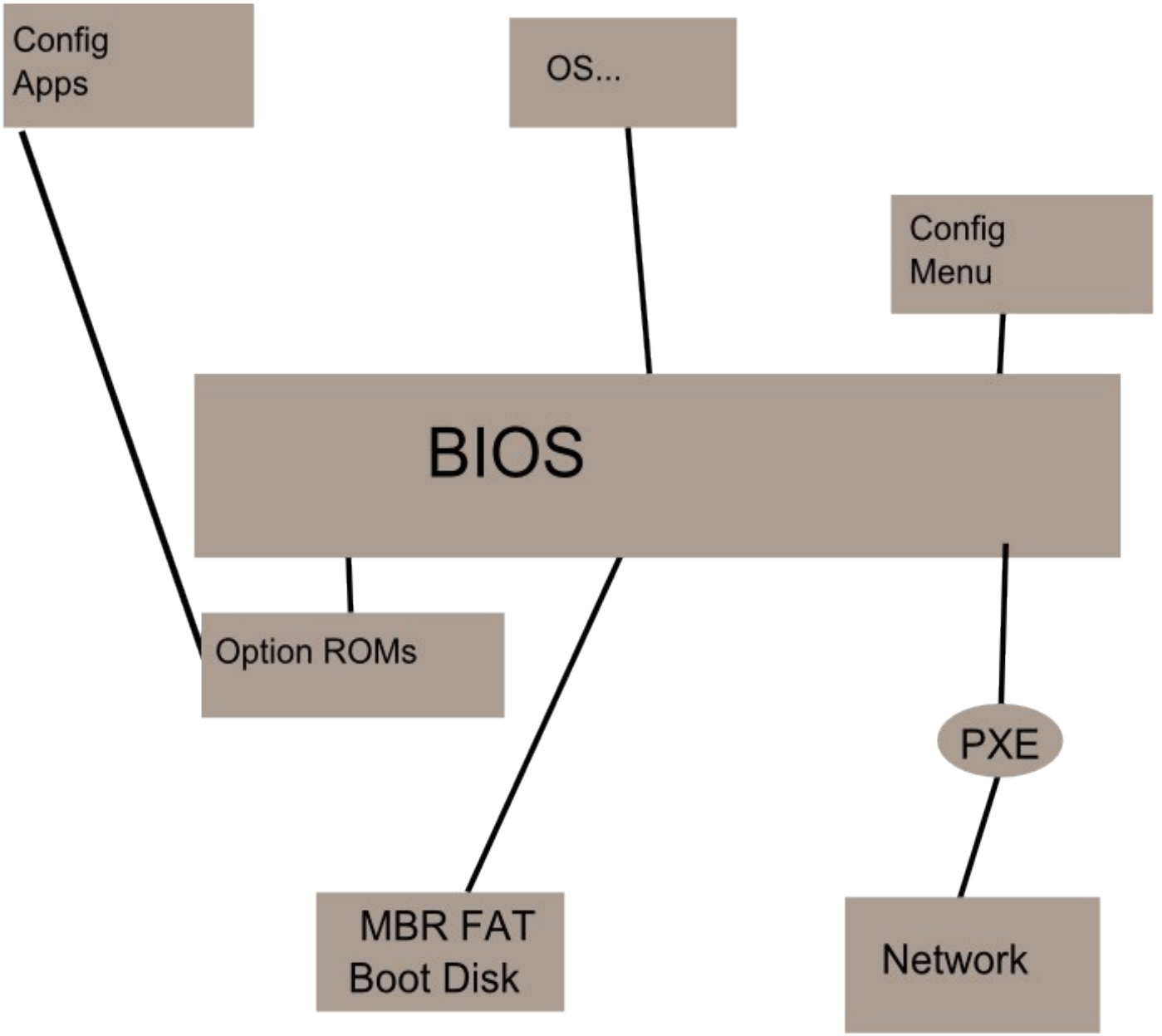
SECTION: background

Why are we talking about this?

- Bootkits are scarier than Rootkits.
- Firmware-level attacks are invisible, if you are using the wrong tools, and/or aren't looking for them.
- Malware authors (and operating system vendors) are moving components into firmware
- Linux kernel, Installers, 'Pre-OS' tools (eg, GRUB2), and other FOSS code needs to work with firmware in a safe manner.

IBM PC BIOS

- A system firmware image (unable to traverse file systems)
- A non-file-based boot loader stored in first sector of MBR partition, that loads the boot loader/OS, which might be a contiguous file next to MBR, easy to get overwritten if multiple OSes used on system.
- A collection of OEM/IHV hardware's Option ROMs (OpROMs), the undefined BIOS 'driver model'.
- Later BIOSs, with appropriate NICs, support PXE network boot, for diskless workstations.


Config
Apps
OS...
Config
Menu
BIOS
Option ROMs
PXE
MBR FAT
Boot Disk
Network

Rings of Protection

- Early Intel processor security model:
  - Ring 3 (Userspace), user mode, apps
  - Rings 2-1 often not used in OS design.
  - Ring 0 (Kernelspace), kernel mode, OS kernel/drivers
- But Ring 0 doesn't cover nuances of HW/FW:
  - Physical hardware -vs- virtualization servers with virtual hardware and firmware.
  - Not just '1 CPU', but multiple processors beyond CPUs, TPM, IPMI, some work when powered-off.
  - Systems Mgmt Mode

Negative Rings for Bootkits

- Negative rings are unofficial, not endorsed by Intel or UEFI Forum. :-)
- Proposed by various security researchers, with differing definitions. Invisible Things Labs (ITL) uses:
  - -1 for VMM
  - -2 for SMM
  - -3 for HW
- Reverend Bill and others have slight variations, but the main point is the same: the 0/3 model is insufficient, and needs to cover FW/HW in more detail.

NIST BIOS Guidelines

- NIST.gov offers guidance on firmware
- 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 authenticates BIOS update mechanism, and optional secure local update mechanism.
- Defines Integrity protection and non-bypassability features.

SP800-147B

- Expanded 147 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
- Added authenticated remote online firmware update mechanisms, in addition to local updates.

SP800-155

- Provide the hardware support to Roots of Trust for BIOS integrity measurements.
- Enable endpoints to measure the integrity of all firmware components and configuration data components.
- Securely transmit 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).

NSA – BIOS Protection Profile

- 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.”
- Good background on understanding threats.
- No Validated Products with this PP. :-(
- www.niap-ccevs.org/pp/pp.cfm

SECTION: UEFI architecture

What is UEFI?

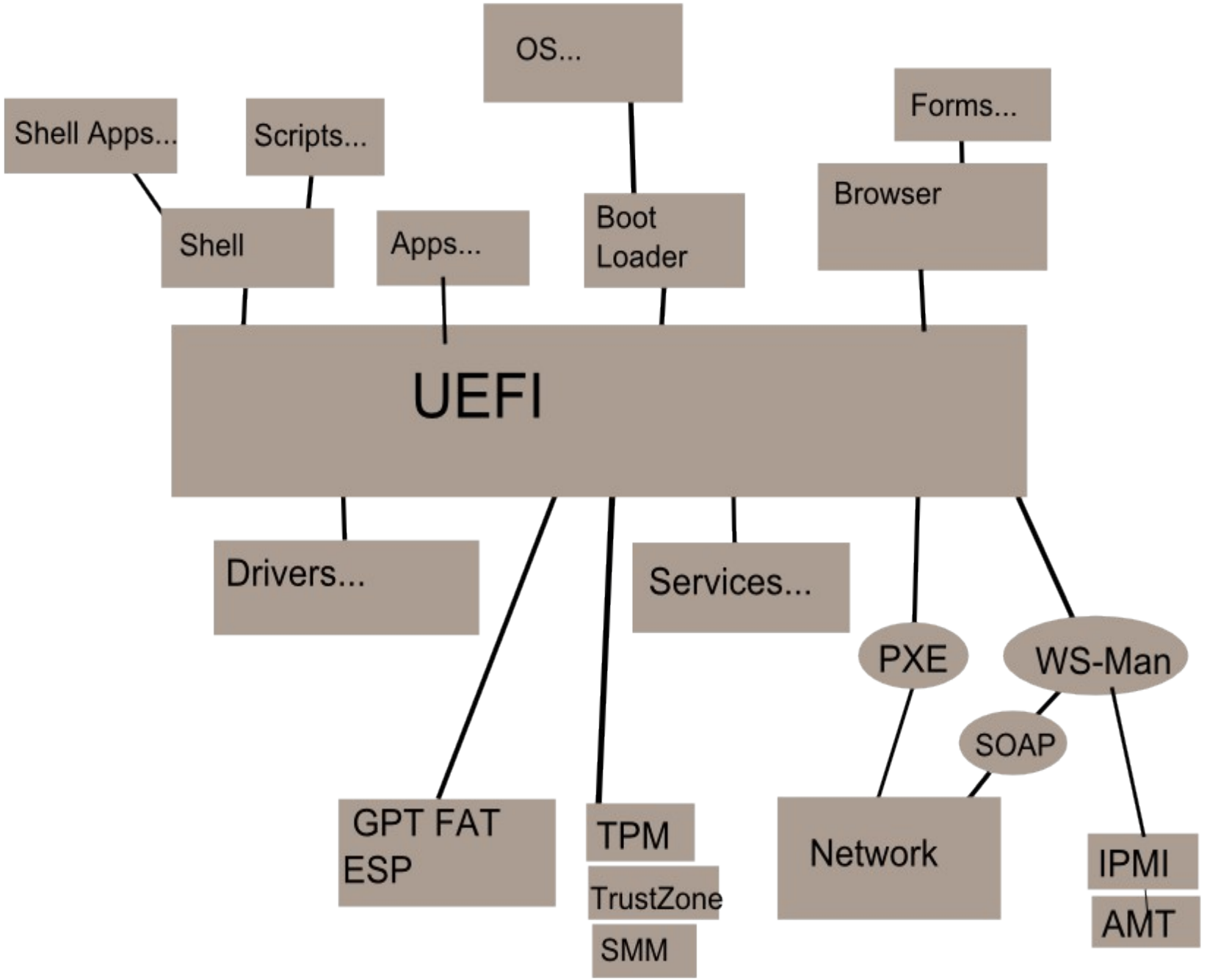
- Universal Extensible Firmware Interface
- A complete technical replacement for the legacy PC BIOS firmware, available for many systems and architectures.
- Created by Intel, initially to boot their Itanium systems, called EFI. Mac used it when switching to Intel. Windows8 requires it.
- Now Unified EFI (UEFI) is controlled by the UEFI Forum, with Adopter and Contributor levels.
- uefi.org

The UEFI solution

- A system firmware image (that can traverse FAT volumes)
- A new GPT partition table format
- A collection of OEM/IHV EFI drivers on hardware's option ROMs
- A collection of drivers/app/files on special FAT volume, the "ESP"
- A collection of NVRAM variables that configures how the system firmware image loads boot loader/OS.

The boot process: UEFI

- No code in boot sector, code is in UEFI firmware system image.
- OS code is in “ESP” in \EFI\vendorname
- Firmware has NVRAM environment variables that point to which vendor file to run.
- Installers add new files to ESP and change variables.


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

What is UEFI?

- BIOS was a barebones system loader with rudimentary interfaces to hardware, for 16-bit real-mode Intel systems. Useful for MS-DOS, and init code of later protect mode OSes.
- UEFI is a complex embedded OS, whose driver models replace the need for any upstream firmware layer to depend on, and currently acts as merely a firmware layer for Linux/Windows, but with no extra OS to load, acts much like a modern MS-DOS/command.com.

History

- 2000, EFI 1.0 (PI = Platform Interface specs)
- 2001, EFI 1.10 (SCT = test suite)
- 2004, TianoCore (EDK = Efi Dev Kit)
- 2005, UEFI 2.0; UEFI Forum (EDK-II = newer EDK)
- 2006, PI 1.0 (UDK = EDK-II snapshots)
- 2007, UEFI 2.1; EDK 1.01 (UEFI 2.0); SCT (UEFI 2.0); SCT (PI 1.0); EDK 1.04 (UEFI 2.1, PI 1.0)
- 2008, PI 1.1; EDK 1.05 (UEFI 2.1, PI 1.0); UEFI 2.2; UEFI Shell 2.0
- 2009, SCT (UEFI 2.1); UEFI 2.3; PI 1.2; EDK-II (UEFI 2.1+, PI 1.0); PI 1.2; EDK-II (UEFI 2.1+, PI 1.0)
- 2010, UDK 2010 (UEFI 2.3+, PI 1.2+)

UEFI Ecosystem

- Chip vendors (Intel, ARM, AMD)
- OEMs (Original Equipment Manufacturers)
- ODMs (Original Device Manufacturers)
- IHVs (Independent Hardware Vendors)
- IBVs (Independent BIOS Vendors)
  - Phoenix, AMI, Insyde Software, Nanjing Byosoft, …
- OSVs (Operating System Vendors)
  - APPL, MSFT, RedHat, SuSE, Canonical, ...
- ISVs (Independent Software Vendors)
  - 'pre-OS' & 'OS-present' apps, UEFI RT svcs, UI, ...

UEFI Forum - security

- UEFI Forum has recently increased their security presence.
- Security web sites with contacts and other info.
- Started 'EDK-II Security Advisories', PDFs containing CVE-like vulnerabilities in EDK-II code, with a TianoCore-Security announce list.
  - tianocore.sourceforge.net/wiki/Security
  - www.uefi.org/security
  - www.uefi.org/revocationlistfile
  - tianocore-security@lists.sourceforge.net
  - sf.net/projects/edk2/files/Security_Advisory/

UEFI Alternative(s)

- Intel Itanium systems NEED EFI.
- Intel x86/x64 and ARM systems can use either BIOS, UEFI, or CoreBoot.
  - Win8 HW logo entices most OEMs to use UEFI.
- ARM never used BIOS, and can use UEFI if vendor wants it, but does not require it.
- CoreBoot is used by Google on ChromeBooks.
  - See the CoreBootPkg in UEFI for a solution – recently checked into EDK-II trunk -- that uses CoreBoot+UEFI, and potentially removes many of the 'binary blobs' Linux community hates!

CSM, Compatibility Support Module

- CSM adds legacy BIOS support to UEFI systems.
- I don't who creates CSM(s), Intel, one/more IBVs, or the UEFI Forum. I think it is closed-source, not part of TianoCore's projects. AFAICT, it is new module that can be added to the open-source EDK-II build system.
- UEFI-framed classes of systems:
  - Class 1: legacy OS only (BIOS, no UEFI)
  - Class 2: hybrid system (BIOS and UEFI)
    - Complicates OS installers, MBR or GPT, and OS init code, ...
  - Class 3: legacy-free (UEFI, no BIOS)
- Read 1st edition of *Beyond BIOS*, it has a CSM chapter; 2nd edition omits that chapter.

Physical and Virtual HW/FW

- Phyical hardware
  - Intel: UEFI is on Intel (x86, x64, and of course Itanium) hardware
  - ARM: 32- and 64-bit hardware has UEFI support

  Virtualization software
  - QEMU is the best solution for virtualized UEFI (see dev slides, later)
  - VirtualBox has some UEFI support.
  - Recently Xen and KVM have support, unclear about how good it is.

Partition Table formats

- BIOS: MBR (Master Boot Record)
  - Max 4 partitions, 2.2TB disk size limit
- UEFI: GPT (GUID Partition Table)
  - More than 4 partitions, 9.4TB disk size limit.

ESP: EFI System Partition

- The ESP (Efi System Partition) is a FAT32-based partition with all the UEFI-centric files that aren't part of the main system firmware image, or baked into device Option ROMs.
- UEFI spec requires BACKSLASHES as path delimiters.
- Examples:
  - \EFI\tools, \EFI\boot(ia32,x64,ia64,arm).efi
- ISVs copy their apps there, IHVs copy their UEFI drivers. Python is installed there. UEFI Shell normally installed there.

ESP

- FAT32-based means no security, have to trust code signing of *.efi files.
- UEFI Forums' ESP Subdirectory Registry
  - A list of UEFI Forum vendors who are known to store files in the ESP, and the name of their directory. List not enforced in code.
  - \EFI\<vendorname>*
  - www.uefi.org/specs/esp_registry

PE/COFF-based Terse Executables

- EFI uses Microsoft PE/COFF-based image file format, not Linux-centric ELF format.
- EFI images called TE (Terse Executable), small variation to PE (Portable Executable) format.
  - Signature is EFI-era's VZ, not MS-DOS-era's MZ.
  - Removes some unused tables/segments that normal PE apps use, that firmware won't need.
- Leverage existing compiler tools's PE support, then translates to TE at last minute.
- Code signing: executables are signed similar to Windows' Authenticode.

EBC (Efi ByteCode)

- A bytecode that supports multiple CPU opcode targets: Intel x86, Intel x64, Intel Itanium.
- Goal: compile to EBC, then IHVs only need ROM space for 1 driver for all Intel platforms, instead of 3, less ROM needed, cheaper units.
- Issue: EFI currently only targets Intel hardware platforms, not all EFI-targetting platforms (eg, ARM 32- and 64-bit, ...)
- Issue: Intel C Compiler is the only compiler that targets EBC. ICC is a commercial-only product, but they do have a Linux edition.
- (I wish GCC and/or LLVM targetted EBC...)

Browser and Forms

- Forms browser, intended to be universal installer engine for all hardware vendors, can be reskinned by OEM/ODM.
- Each vendor provides new content (new form(s)), and browser looks similar to use between hardware. BIOS boot menu...
- Forms written by vendors, stored in Internal Forms Representation (IFR), lists of of raw resources, fonts, strings, images (BMP, JPEG), forms, and a form language parser. Vaguely similar to .rc data compiled into .exes.

Driver Models

- UEFI 2.x has dozens of driver/service models. EFI 1.x had dozens of other driver/service models.
- EFI drivers were meant to replace existing EFI drivers, like how an OpROM updates functionality in the main BIOS. A UEFI driver/service can filter/impersonate another EFI driver/service.
- Most UEFI code terminates when OS loads. Some Runtime Services remain resident and OS is able to call these, disk/video IO during OS int, variables for booting.

Driver Models

- UEFI has drivers for
  - Initializing system (memory, USB, PCI, ACPI, TPM, TrustZone, IPMI, SMBIOS, ...)
  - Booting from local media (ROM, flash, disk, floppy, CD, DVD, …)
  - Booting from network (NIC, ...)
  - Interfacing with user during boot (keyboard, mouse, …)
- No printer drivers or scanner drivers etc., not required to boot a system

Driver Models

- PEI Modules and Services
  - PEI = Pre-Efi Initialization
  - PEI Modules (Drivers)
- DXE Drivers and Services
  - DXE = Driver eXecution Environment
- UEFI Drivers and Services
- UEFI 'Pre-OS' Applications
  - System tools, boot loaders/managers, like GRUB2, eLILO, ...

UEFI Services

- UEFI Boot Services
  - used by OS loader to transition from FW to OS kernel (or boot manager, a pre-OS UEFI app)
  - ExitBootServices() ends this state, then only Runtime Services are available
- UEFI Runtime Services
  - Services available after ExitBootServices() for UEFI-aware OSes to consume, instead of BIOS interrupts.
  - virtual memory, time, variables, console i/o, reset, capsule, memory, event/timer, protocol, image

File System Drivers

- Unlike BIOS, UEFI can traverse file systems, during init.
- UEFI's ESP must be FAT32.
  - Apple ignores FAT-only rule, and has an HFS+ driver with their systems.
- UEFI FAT driver is non-BSD, additional Microsoft legal restrictions. Binaries are in EDK-II, sources are on edk2-fatdriver2 project.
- Other companies and projects (eg, VirtualBox, rEFInd, etc.) have other file systems (UDF, HFS+, ISO 9660, NTFS, Ext2, etc.)

UEFI Network Stack

- UEFI systems can boot via network (PXE), in addition to booting via local storage media.
- Can remotely control UEFI systems (even when powered off) via IPMI. IPMI uses WS-MAN protocols when used remotely.
- Network stack protocols:
  - iSCSI, IPv4 and IPv6, ARP, DHCP (see RFC 5970), UDP, TCP, PXE, TFTP, IPsec, VLAN, WiFi, ethernet, CHAP, WS-MAN, Bluetooth, ...
- UEFI Shell commands:
  - ifconfig, ping, vlan config, telnet

Testing UEFI's Network Stack

- Enterprise admins (and end-users) should explicitly configure IPMI and network use on systems. IPMI is on laptops, not just high-end servers.
- Evaluate new UEFI systems, sniffing 24x7, wired and ethernet, powered on and off, w/ and w/o OS, to determine what the vendor's firmware might be doing over a network.
- UEFI speaks WS-Management, CHAP, IPsec, IPMI, PXE (network boot), and most core IPv4/IPv6 network protocols, including iSCSI.

User Identity (UID) Drivers

- A credentials provider driver, for user authentication devices (smartcards, biometrics, etc.), to control access to network boot, or booting from secure local media (like some fancy CryptoStick with UEFI support).
- Used in UEFI for PXE and IPsec, maybe elsewhere.

Boot Loaders/Managers

- UEFI has a 'Pre-OS application' model, using UEFI Applications, for apps that run before OS, mainly boot loaders and OS init code.
- Some of the FOSS EFI-aware boot managers:
  - GRUB2
  - rEFInd (replaces rEFIt)
  - Clover EFI bootloader
  - Gummiboot
  - XPC EFI Bootloader
  - eLILO
- The best information on this: RodsBooks.com

Linux Kernel: EFI stub

- Linux kernel has built-in support for EFI.
- Initially x86, but later ARM too.
- Shell> bzImage console=ttyS0 root=/dev/sdb initrd=initrd.img
- Both BIOS and EFI boot loaders can still load and run the same bzImage (kernel image)
- See the CONFIG_EFI_STUB directive for more information.

Secure Boot

- Secure Boot is an optional build feature of UEFI 2.x, based on signature checks.
- Uses multiple sets of keys
  - PK, Platform Key, verified KEKs
  - KEK, Key Exchange Keys, Verify db and dbx
  - db, Signature Database
  - dbx, Forbidden Database
- It can use, but does not rely on TPM.
- See '*Secure Boot on Linux*' talk at IDF2013

Linux: Shim

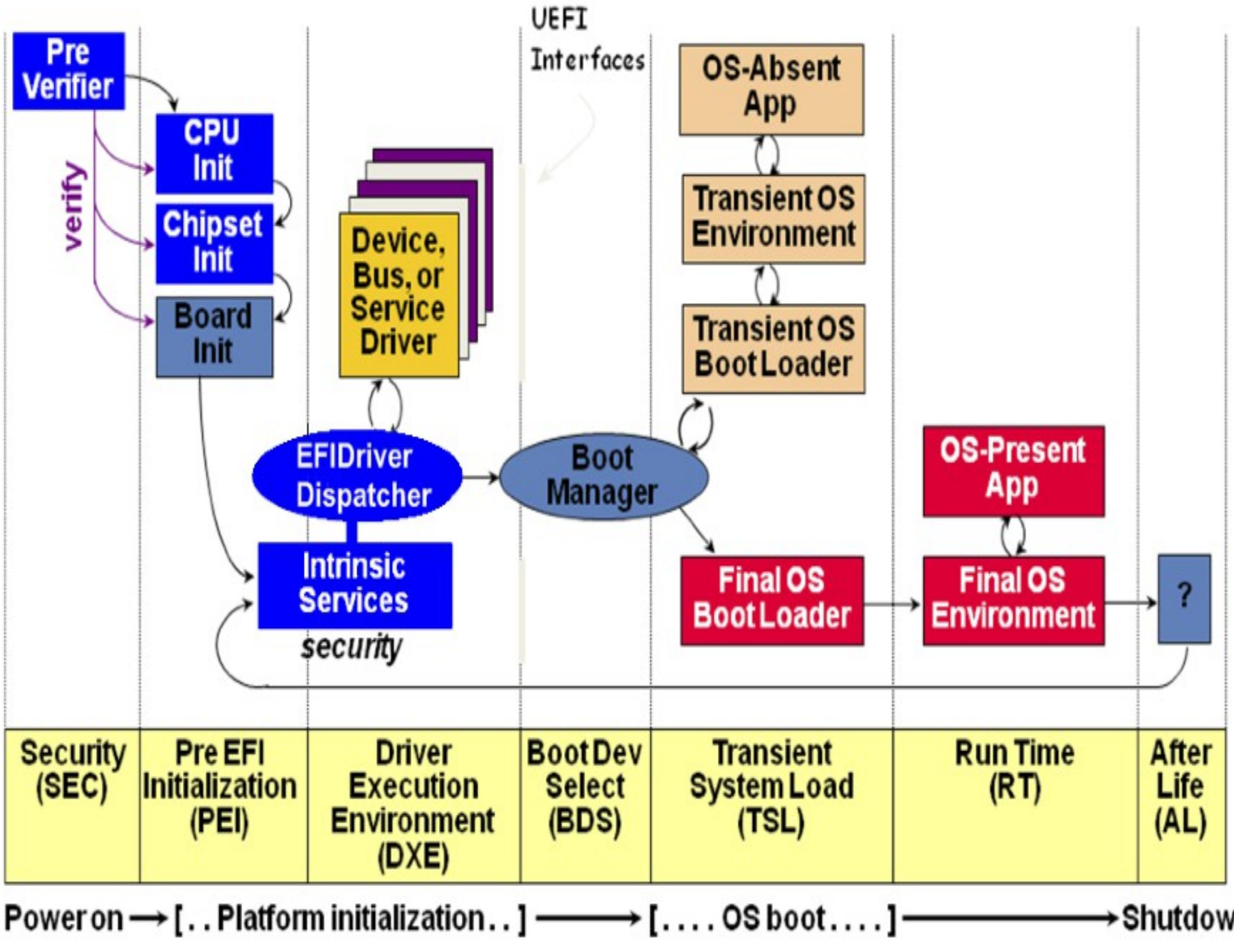
- Shim, a shim loader for Linux
- Workaround to boot Linux on a Win8 SecureBoot system.
- Signed by Microsoft via Linux Foundation.
  - linuxfoundation.org/publications/making-uefi-secure-boot-work-with-open-platforms
- Matthew Garrett
  - github.com/mjg59/shim
- See also MachineOwnerKey (MOK)

Linux distributions

- Linux OSVs members of UEFI Forum: Canonical, SuSE, RedHat
- These days, most large distributions have some form of UEFI support, including community-based ones (eg, Debian, Arch, etc. ...even FreeBSD).
- In addition to Linux, FreeBSD also supports UEFI
- (I wish Linux OSVs worked with OEMs to get SecureBoot+Linux working properly on laptops and desktops, not just high-end servers.)

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
- Next slide's image source: uefi.org.


UEFI
Interfaces
Pre Verifier
verify
CPU Init
Chipset Init
Board Init
Device, Bus, or Service Driver
EFIDriver 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

SECTION: development

- UEFI coding models:
  - UEFI Shell scripts
  - Python scripts
  - Native UEFI applications
  - UEFI Shell Applications
  - GNU-EFI-based UEFI applications

The UEFI Shell

- The UEFI Shell is the command line interpreter, used for interacting with UEFI system/drivers, originally made by EFI driver developers at Intel, to test their code.
- It works in interactively with users or in batch mode with scripts, like most modern shells.
- The UEFI Shell replaces the older EFI Shell.
- The UEFI Shell is a UEFI Application.
- Shell is often not exposed/available to user on most UEFI-based consumer devices.

Shell modularity

- For each architecture, there can be multiple shells available, depending on how built.
- Like most of UEFI code, can be built for Retail or Debug builds.
- Profiles: Install1, Debug1, Driver1, Network1
- Support Levels: 0=Minimal, 1=Scripting, 2: Basic, 3=Interactive
- Vendor can omit some commands, and add new commands.

Shell console I/O

- Text can be either ASCII or Unicode (UCS-2)
- No readline ability, but can up/down-arrow through command history
- Regular expressions: only a handful of expressions (*, ?, and []) supported, barely better than MS-DOS
- Invalid characters: * ? < > \ / " 0x0001 0x0002

Redirection and Pipes

- Shell supports 'normal' redirection and piping:
  - “|”, "<", ">>", "1>>", "2>>", ">", "1>", "2>"
- To force ASCII, instead of Unicode:
  - "<a", ">>a", "1>>a", "2>a", "1>a", ">a"
- Can redirect to/from an environment variable, not just stdin/stdout/stderr/file, or the NUL file/device:
  - ">i" (StdIn), ">o" (StdOut), ">e" (StdErr),
  - ">v" (environment variable)

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.

Variables

- UEFI's Variables runtime service are vaguely like to 'environment variables' of Unix/MS-DOS: key/value string pairs, ...but a lot more complex.
- Some usage is like 'environment variables', like UEFI Shell's use. Others are NVRAM-based firmware config settings, others are used on for FW/OS handoff.

Variables

- Besides name/value strings, each variable also has a GUID, and multiple attributes (eg, if it persists boots, when it can be used).
- In UEFI Shell, variable usage uses MS-DOS style %foo% substitution, not Unix style $foo.
- UEFI Shells PATH is semicolon-separated, using absolute or relative paths
  - ".\;\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

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 .ns extension, not .bat.
- 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.
- 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 a UEFI Shell Application.
- 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.
- But once installed, easy to use Python on EFI!
- Intel's CHIPSEC tool is Python-based, and works under EFI (as well as Linux and Windows), their user docs on getting Python working using UEFI is probably current the best available.

Styles of UEFI Applications

- Native UEFI Apps
- UEFI Shell Apps
  - Shell provides libraries for File I/O, etc.
- EADK Apps
  - EADK provides some C Std Lib (LibC) support for EFI

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 Applications

- UEFI Shell Applications have richer libraries than native applications – mostly File I/O instead of just stdio/stdout/stderr – 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 Shell Applications

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

UEFI Shell App: hello world

```
#include <Uefi.h>
#include <Library/UefiLib.h>
#include <Library/DebugLib.h>
#include <Library/ShellCEntryLib.h>
INTN EFIAPI ShellAppMain(
    IN UINTN Argc, IN CHAR16 **Argv)
{
    Print(L"Hello, EFI world!\n");
    return 0;
}
```

UEFI toolchains

- Legacy:
  - EDK (**E**fi **D**ev **K**it): replaced by EDK-II
  - EFI Toolkit: legacy EDK-based app dev kit, replaced by EADK.
- Current:
  - EDK-II: replaced EDK
  - 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 (limited use)

UDK/EDK-II

- EDK-II is the trunk project.
- UDK is a snapshot of some TianoCore projects, mostly EDK-II, and some documents, as a single ZIP.
- EDK-II/UDK can build system firmware images, standalone *.efi drivers or applications.
- EDK-II/UDK includes EADK as a module.
- UDK2010 means it uses 2010-era UEFI specs.
- UDK targets Windows. For Linux, EDK-II trunk is generally more useful.

EADK

- EADK: EFI Application Development Kit
- The EADK provides subset of C Standard Library, to make it easier to port C apps to UEFI Apps.
- Tries to conform to "C 95 spec", ISO/IEC 9899-1990 C Language Standard with Addendum 1, plus also includes system calls, defined in sys/EfiSysCall.h and/or unistd.h, and sockets.
- Eg: Cpython 2.7x ported to EFI using EADK.

TianoCore.org

- TianoCore.org is UEFI Forum's front-end to the multiple TianoCore projects hosted on SourceForge.net.
  - edk2.tianocore.org maps to edk2.sf.net
- Main project: edk2 (aka EDK-II), and occasional UDK snapshot releases
- Many other projects (esp. edk2-buildtools, and edk2-fatdriver2), ...and multiple other legacy EDK-I-era ones.
- Most of the code is BSD-licensed.
- Code uses Subversion; there is a Github mirror.

EDK-II build environment

- EDK-II uses a unique build environment.
- Uses a config file that has definitions for all supported toolchains; you can add new ones.
- Supports GCC, LLVM, Microsoft C (Pro, not Express), Intel C, ARM's C Compiler.
- Builds for Debug/Retail targets, like Windows kernel/DDK/SDK model.
- Intel C Compiler (commercial) is the only one that targets the EFI ByteCode (EBC).

EDK-II Build tool: Build

- EDK-II doesn't use Make, CMake, etc, there is a new build tool called Build, which invokes C compiler and other tools.
- Build has multiple config files, with a spec for each one:
  - Build Spec, INF, FDF, DSC, DEC, VFR, PCD
  - sf.net/projects/edk2/files/Specifications
- Example: building the EADK samples for x64.

  cd ~/fw/edk2

  ./edksetup.sh

  build -a X64 -p AppPkg/AppPkg.dsc

EDK-II: developer tools

- Besides Build, there are many specialty tools available in the UDK, many called by Build, 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

GNU-EFI

- An alternative toolchain for creating UEFI Applications
- Translates ELF binaries to UEFI PE+ TE images; see docs for limitations.
- Uses GCC tools, no Intel or Microsoft compilers; only works on UNIX systems.
- Uses GNU Make, not EDK-II's build.
- Study and the rEFIt/rEFInd build notes for working examples of using GNU-EFI.
- sf.net/projects/gnu-efi

OVMF

- Open Virtual Machine Firmware - A system firmware image, as built by EDK-II/UDK build output, that can be read by QEMU.
- OVMF is also the name of the EDK-II module. Use the EDK-II (or UDK) to build your own. You can build with debug tracing enabled, or with full source-level debugging.
- Useful if you want to explore Linux init and use of UEFI Runtime Services.
- Besides QEMU, apparently Xen and KVM have recently added support! VirtualBox has some EFI support as well.

MinnowBoard

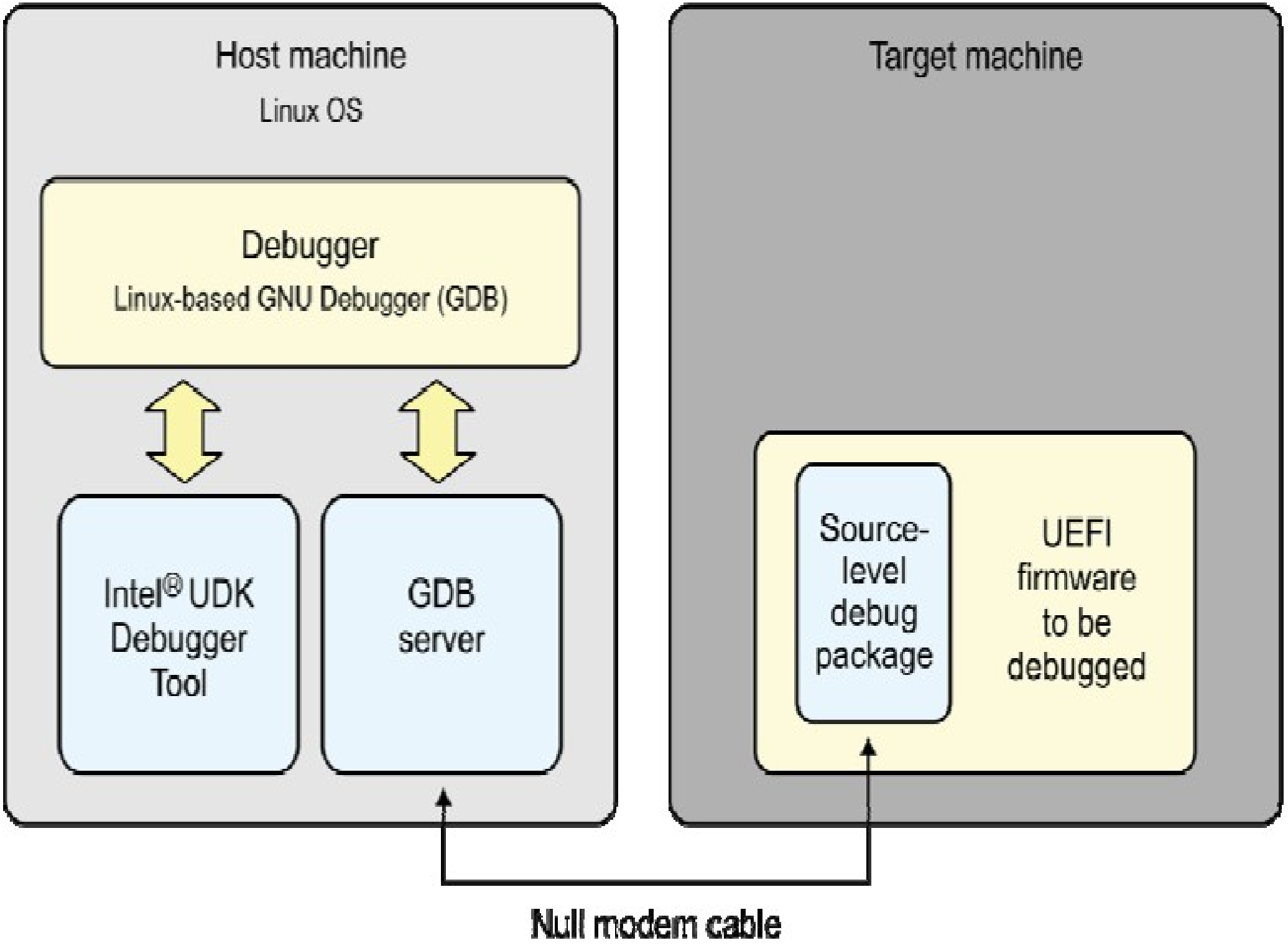
- minnowboard.org
- Intel Atom-based, Linux-based dev platform for hackers/hobbyists of both UEFI and Yocto.
- MUCH cheaper than Tunnel Mountain board.
- Not as capable as Tunnel Mountain.
- Original MinnowBoard out-of-print, these days it is the Minnow MAX, single or dual core.
- Useful box for developing CHIPSEC modules (you can update firmware and test your modules).

Tunnel Mountain

- A pre-assembled Intel UEFI Dev board that lets you re-flash the system BIOS.
- Dev Platform (DevPlt), 3 kinds.
- ICE(sp)-enabled, if you have $nnK to spend on an Arium ICE and debugger.
- www.tunnelmountain.net
- Located in Bellevue, WA, at HDNW/CompStop
- Uses a DediProg.com device for flashing
- uefidk.com

Intel UDK Debugger Tool

- Intel UEFI Development Kit (UDK) Debugger Tool
- Intel-specific debugger solution, that requires Tunnel Mountain dev platform.
- Uses GDB on Linux, Windbg on NT, not supported on Mac OS x (or FreeBSD).
- Needs proper debug cables to talk between two systems.
- Next image, source: www.intel.com/udk


Host machine
Linux OS
Debugger
Linux-based GNU Debugger (GDB)
Intel® UDK Debugger Tool
GDB server
Target machine
Source-level debug package
UEFI firmware to be debugged
Null modem cable

ARM: BeagleBoard

- The EDK-II initially only supported Intel targets. The first ARM target was the BeagleBoard, a 32-bit ARM dev board.
- See EDK-II's BeagleBoardPkg module
- See-Also: Linaro dev boards, which has a fork of EDK-II for newer ARM dev boards, which are not yet supported in EDK-II trunk.

ARM and Linaro

- Linaro has a fork of TianoCore which they focus on their ARM-centric toolchain.
- EDK-II has fewer ARM dev boards targets supported.
- Linaro systems also often have a complete QEMU solution for that board.
- The hobbyist-level boards include Pandas, Beagles, and Origen.
- Linaro.org
- ARM has a Community Edition of their Eclipse-based DS-5, for GUI firmware emulation/dev.

UEFI Boot Disk(s)

- AFAIK, there are 2 kinds of UEFI Boot disks.
- One is for BIOS-based systems. Using DUET, you can boot into a UEFI environment. See EDK-II's DuetPkg readme for more information.
- Another is for UEFI-based systems, usually with Secure Boot disabled. Partition thumbdrive using GPT, with a FAT32 drive, with a “\EFI” subdirectory setup like a normal ESP would be, with UEFI Shell in it's proper directory for the appropriate architecture(s). Useful if the OEM didn't provide UEFI Shell in their system.
- Being able to directly boot into UEFI is helpful, sometimes Linux (or Windows) gets in the way of HW/FW exploration. See-also: BITS and CHIPSEC.

SECTION: test tools

SCTs

- Self Certification Test (SCT)
- UEFI Forum's test suites, for PI and UEFI protocol conformance.
- Created by Intel.
- Linaro has ARM-centric fork.
- Older EDK-I-era SCTs are on TianoCore.org.
- Current EDK-II-era SCTs are only given out in binary form, in occasional drops, to public. UEFI Forum members have current trunk access.

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, 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
- Was BIOS-centric, but as of build 945, BITS also has EFI support

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

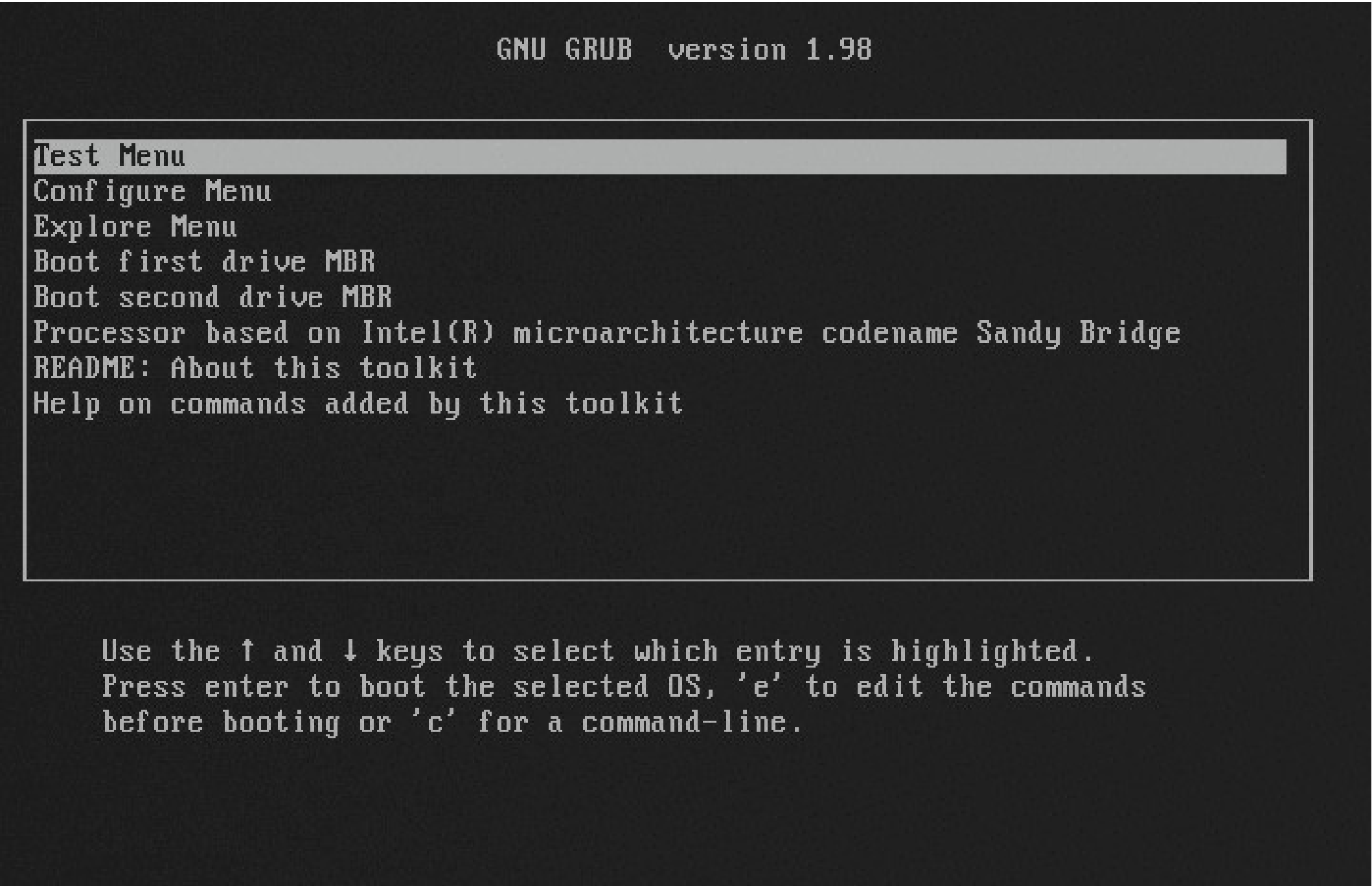


Image source: http://biosbits.org/scripting

```
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.

FWTS

- Useful to help find interesting Linux kernel warnings from firmware.
- 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

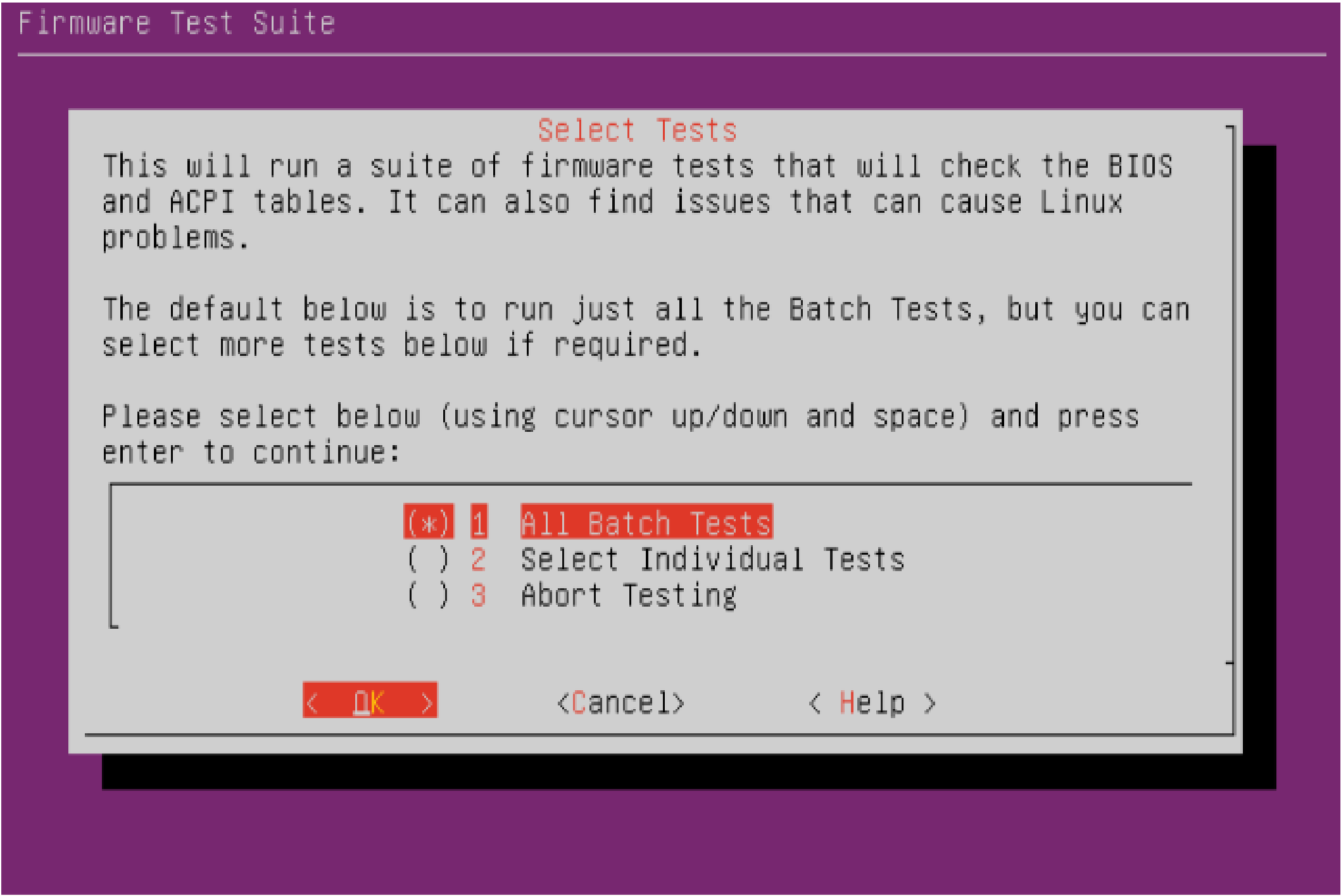
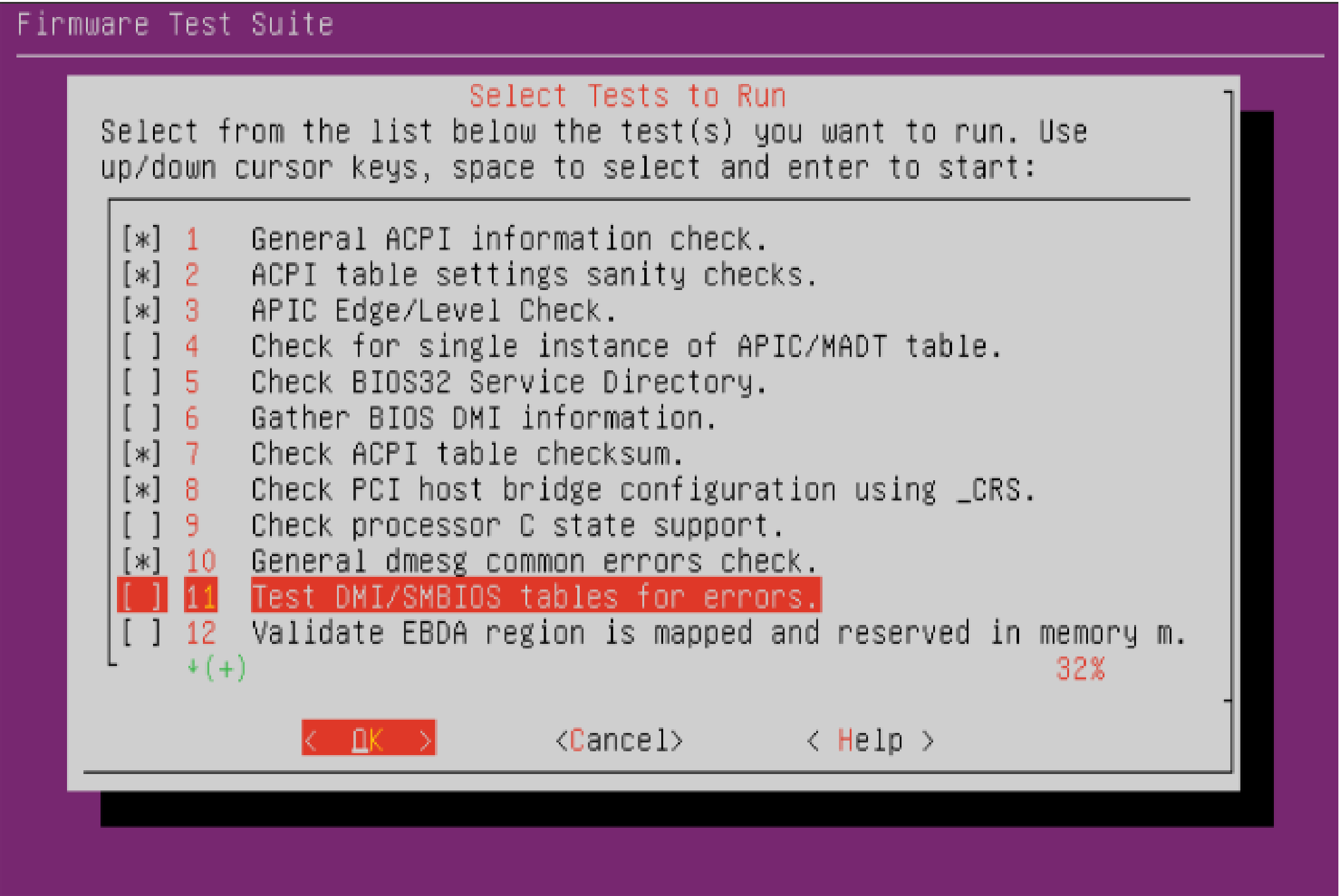


Image source: https://wiki.ubuntu.com/Hardware



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
- chipsec@intel.com

CHIPSEC

- A Python-based open source project targetting hardware/firmware security researchers.
- 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

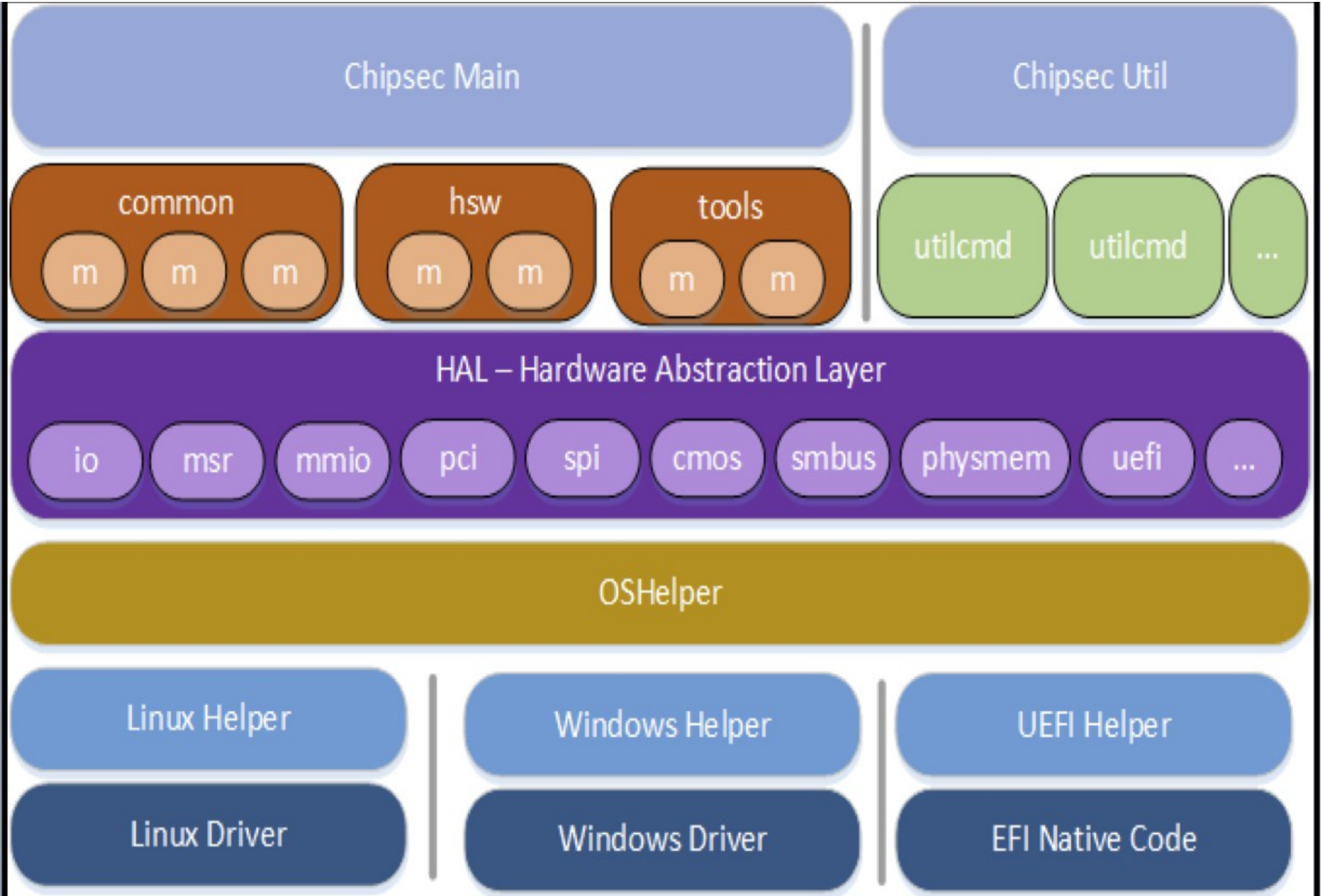
- The CHIPSEC python module can be run from a Python interactive shell, or used in other Python scripts. Example:

```
import chipsec_main

chipsec_main._cs.init(True) # if chipsec driver is not running

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

chipsec_main.run_loaded_modules()
```

CHIPSEC - targets

- Intel UEFI Linux system usage types:
  - Install CHIPSEC OS driver, and use CHIPSEC.
  - Boot a Linux LiveCD with CHIPSEC installed (such as LUV-live), use CHIPSEC.
  - Boot a 'UEFI boot disk', boot into UEFI Shell, use CHIPSEC.
- Intel FreeBSD/MacOSX/Android system:
  - Boot a Linux LiveCD with CHIPSEC installed.

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 examples

- Live
  - 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
- Offline
  - 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

CHIPSEC

- Their user docs are good for installation and general use. Their security conference presentations are even better at describing the specifics of each vulnerability and how to test it. Read all of their post-conference slides, in addition to normal docs, RuxCon 2014, BlackHat 2014, CanDecWest 2014, etc.
  - RuxCon 2014 BIOS Attack Summary pdf
  - https://cansecwest.com/slides/2014/Platform%20Firmware%20Security%20Assessment%20wCHIPSEC-csw14-final.pdf
  - https://www.blackhat.com/docs/us-14/materials/arsenal/us-14-Bulygin-CHIPSEC-Slides.pdf

CHIPSEC - enterprise caveat

- CHIPSEC uses a kernel driver on Linux (or Windows, even with UEFI usage) to provide their HAL.
- The CHIPSEC team warns to NOT use the CHIPSEC driver on operational systems, as the driver allows user mode access to HW resources and may allow malware to access privileged hardware resources. See their warnings.txt for more info. It should only be used in test environments.
- Using CHIPSEC w/o a driver on local OS means rebooting into a Linux live CD, which has other issues...

CHIPSEC - Initial enterprise 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 enterprise use

- 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.

CHIPSEC: patches wanted

- CHIPSEC team is interested in patches from Linux and open source community. Ports to other archs/OSes, new tools (eg, fuzzers) that use CHIPSEC.
- For Linux, increased access to interesting kernel data structures from CHIPSEC, to enable more interesting tests and useful reports on Linux.
- A FreeBSD port would also be nice.

LUV (Linux UEFI Validation)

- 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

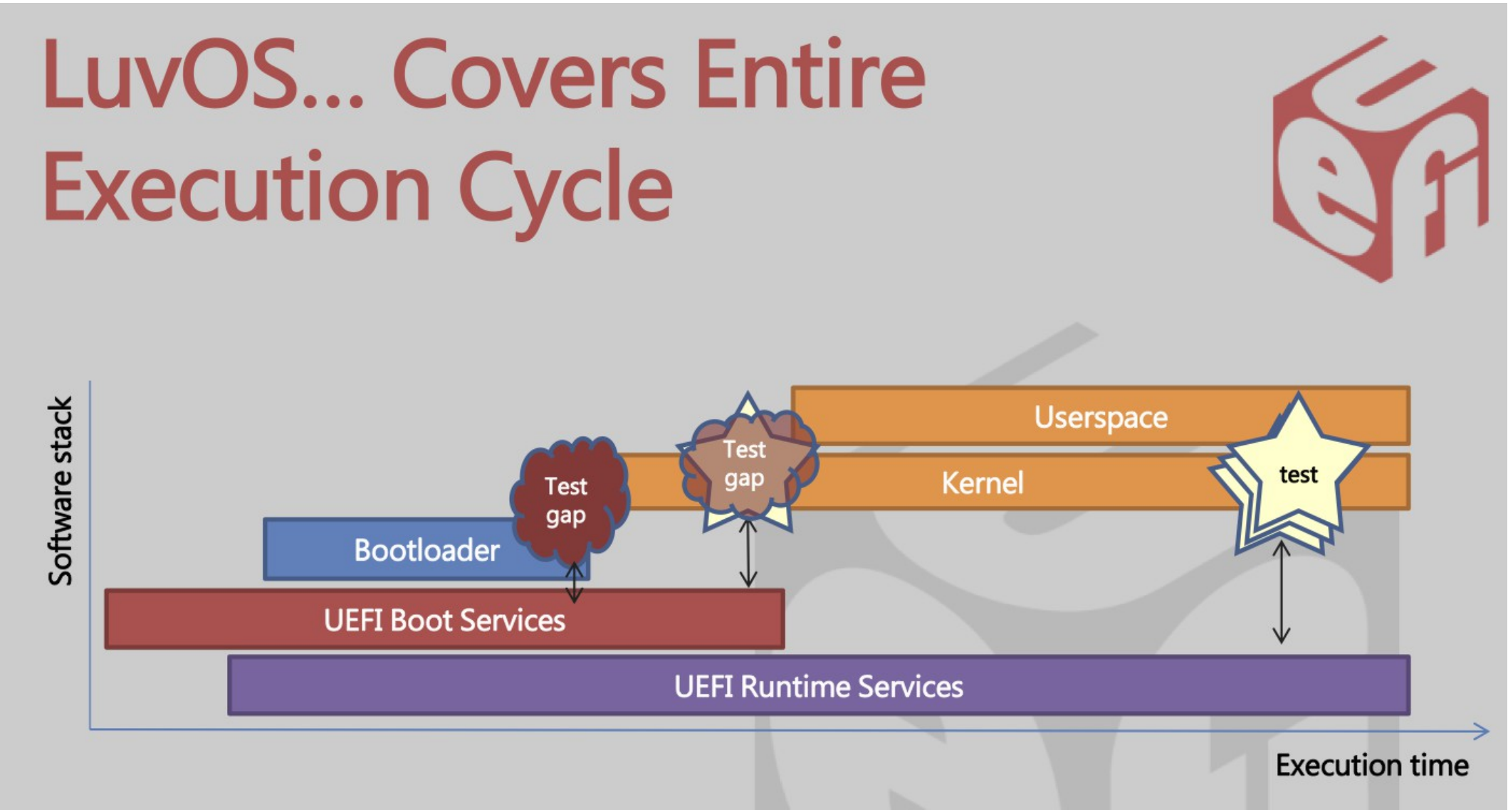


Image source: LinuxCon 2014 talk by Brian Richardson (Intel) and Alex Hung (Canonical), “UEFI Test Tools for Linux Developers”

LUV

- Contains FWTS, CHIPSEC, BITS, and other UEFI-centric HW/FW QA tools.
- 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. 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.
- 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.

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/
- wiki.linaro.org/LAVA%20Team%20wiki%20page

Other Tools

- FlashROM and other tools from Coreboot.org
- There are dozens of small UEFI security tools, most on Github. Samples:
  - EFIPWN, github.com/G33KatWork/EFIPWN
  - Ida-efiutils, github.com/snarez/ida-efiutils
  - EFInject, bitbucket.org/troeger/efinject
  - FMK, firmware-mod-kit.googlecode.com
  - UEFITool, github.com/LongSoft/UEFITool
  - Universal-IFR-Extractor, github.com/donovan6000/Universal-IFR-Extractor
  - Binwalk, github.com/devttys0/binwalk
  - Sign Search, freecode.com/projects/signsrch

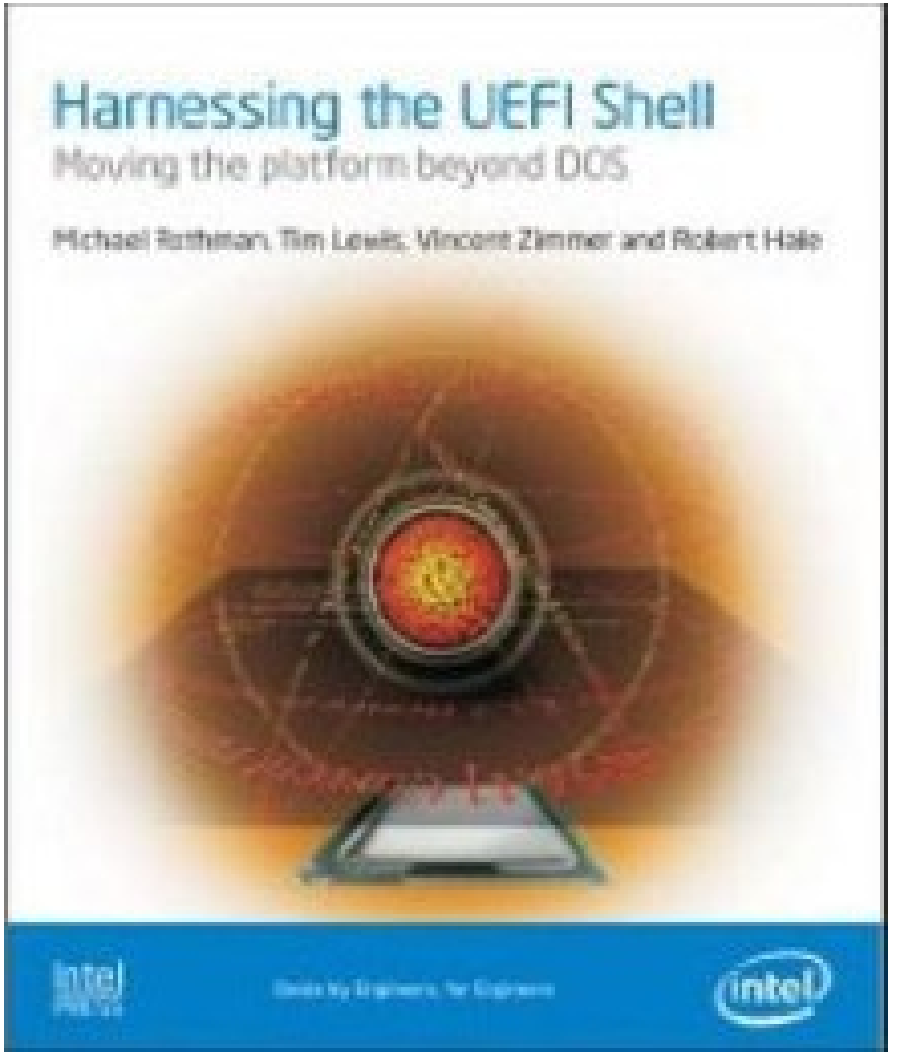
SECTION: more info

Professional Firmware Help?

- Firmware Security Firms I know about:
  - Invisible Things Labs (security)
  - LegbaCore (security)
  - UEFI Forum ISVs (development)
- Others?

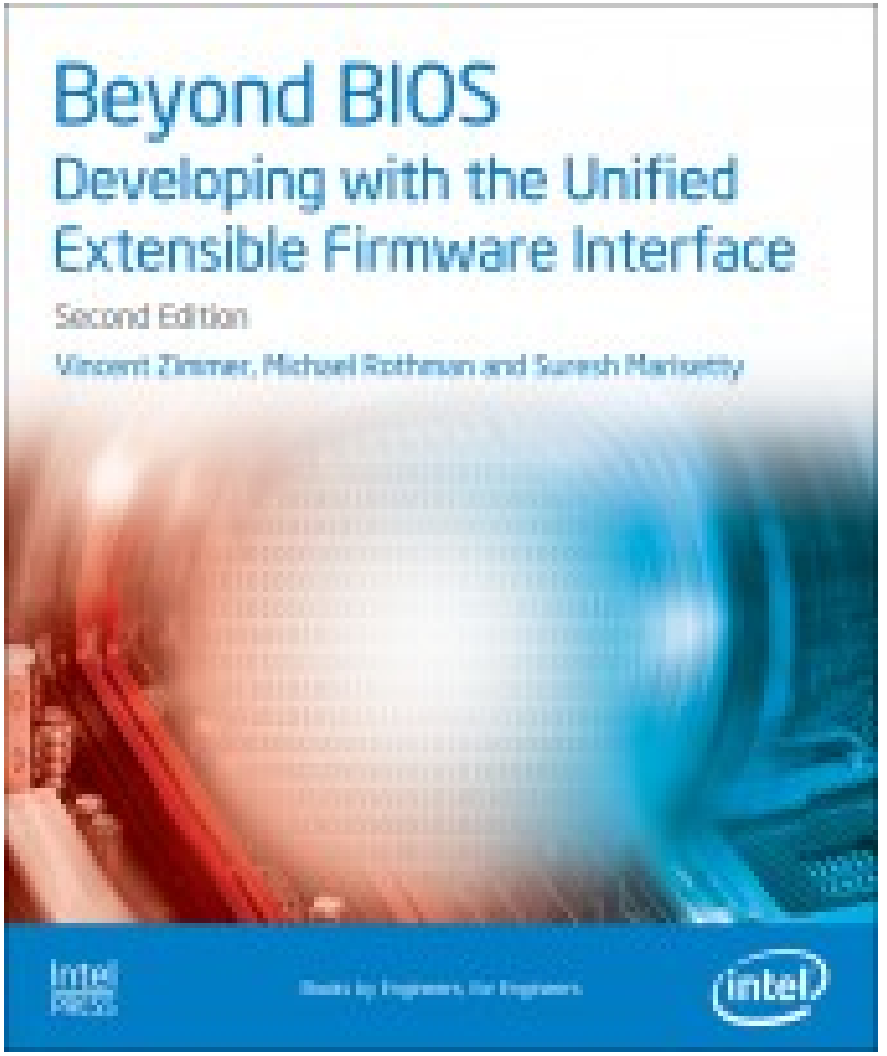
More Information

- Harnessing the UEFI Shell
  - Michael Rothman, Tim Lewis, Vincent Zimmer, Robert Hale
  - Intel Press, 2009
  - The main book on using the UEFI Shell.



More Information

- Beyond BIOS, 2nd Ed.
  - Vincent Zimmer, Michael Rothman, Suresh Marisetty
  - 2012, Intel Press
  - The main book on UEFI architecture and development.
  - 1st ed has chapter on CSM.



More Information

- UEFI Forum
  - uefi.org
    - The UEFI specifications, and security web page
    - Members have access to more lists and security working groups.
  - tianocore.org
    - The source code, and mailing lists, esp. edk2-devel
    - The 'EDK-II Security Advisories” PDFs announced on the “TianoCore-Security” list
    - tianocore.sourceforge.net/wiki/Security
    - www.uefi.org/security
    - tianocore-security@lists.sourceforge.net
    - sf.net/projects/edk2/files/Security_Advisory/

More Information

- Intel.com
  - UEFI content, and Intel-centric HW/FW content...
- Intel CHIPSEC documentation
  - INVALUABLE to understand the various known firmware exploits covered in their modules.
- Intel SSG's UEFI training courseware/labs
  - Slides and labs for Intel's 3-day internal training are publicly available. Small amount of Linux content.
  - sf.net/projects/edk2/files/Training/TrainingMaterial/
- Intel web-based, Flash-centric training
  - sf.net/projects/edk2/files/Training

More Information

- Linux-efi mailing list on vger.kernel.org
- Linaro.org UEFI and Validation (LAVA) mailing lists.
- LUV mailing list
- FWTS mailing list
- Rods Books's web site on EFI boot loaders
  - rodsbooks.com

More Information

- Blog of Vincent Zimmer (Intel)
  - vzimmer.blogspot.com
- Blog of Tim Lewis (Phoenix)
  - uefi.blogspot.com
- Blog of James Bottomley (Parallels)
  - blog.hansenpartnership.com/author/jejb
- Blog of Matthew Garrett (Red Hat)
  - mjg59.dreamwidth.org
- Microsoft UEFI OEM Requirements
  - www.microsoft.com/whdc/system/platform/firmware/uefireg.mspx

More Information

- LinuxCon North America August 2014
  - Series of Linux-centric, UEFI-centric training presentations
- NIST SPs and IAD PP with BIOS guidance
- InfoSec conferences (DefCon/BlackHat, RuxCon, ReCon, ...)

Summary

- Learn to use CHIPSEC, LUV, FWTS, BITS, maybe LAVA
- Run CHIPSEC/FWTS, perhaps via LUV-liv, save logs and FW image, track changes over history.
- Track EDK-II Security Advisories
- Track CHIPSEC for new vulnerabilities.
- Track new HW/FW exploits at security conferences

CREDITS

- Thanks to John, Vincent, Yuri, the edk2-devel list, Black Lodge Research, DC206, Intel and UEFI Forums' tech writers.
- Thanks to UEFI Forum for making specifications publicly available (again)!
- Especially thanks to John for enterprise CHIPSEC usage guidance!

End of Slides (!!)

- Questions?
- Discussion...
  - What tools did I omit?
  - …
- Thanks for attending!