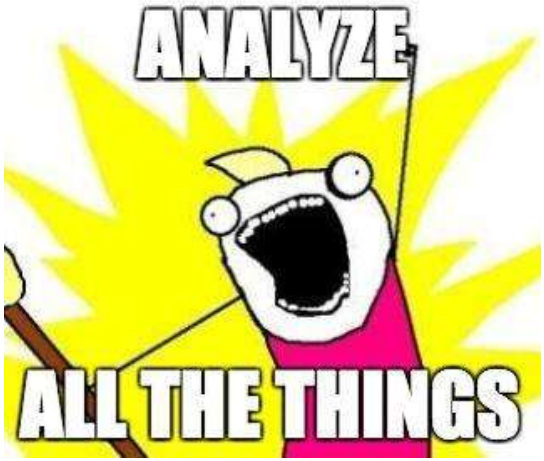


ANALYZE2016
San Francisco

Different methods of BIOS analysis: Static, Dynamic and Symbolic execution

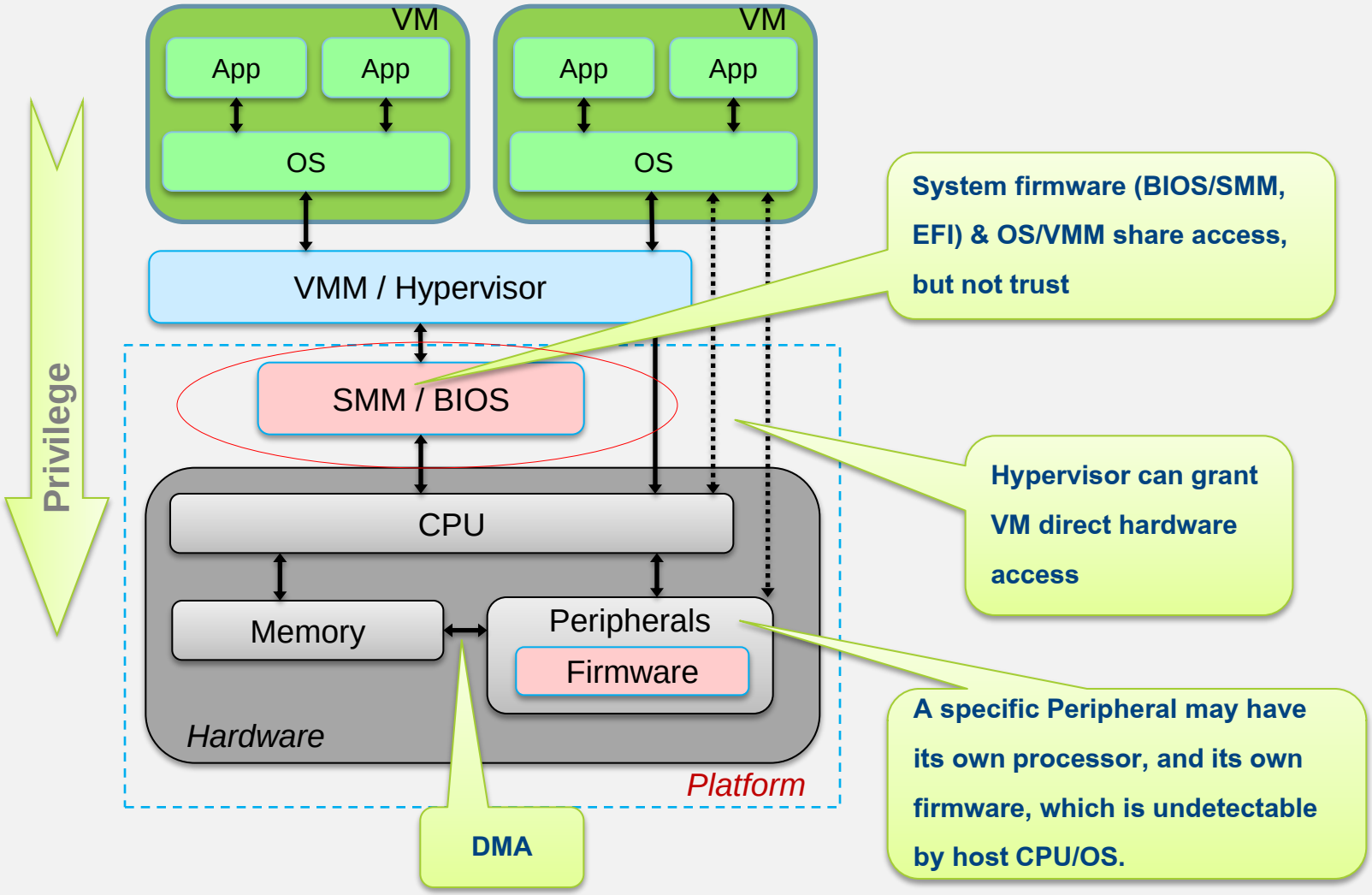
Oleksandr Bazhaniuk, Yuriy Bulygin

Agenda

- BIOS overview
- Static and dynamic methods for BIOS analysis
  - CHIPSEC tool
- Symbolic execution methods for BIOS analysis
  - Excite tool
- Conclusions

☣ BIOS overview

Where is system firmware?



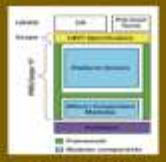
BIOS

UEFI

Source: https://www.ccn-cert.cni.es/publico/VIII_Jornadas/13-UEFI_ArmaDobleFilo_CCN.pdf

What's in UEFI

Mostly written in C.
High code re-use.

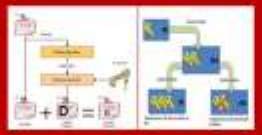


Emphasis on Specifications.
Standards compliance.

Better platform scaling. For e.g. removes shadow ROM limits.

Storage.
GPT removes 2.2 TB MBR restriction.

CPU Architecture independent. Platform design flexibility.



Secure boot solves "trust" related system integration challenges.

Pre-boot Networking.
Ipv4, Ipv6, PXE,
VLAN, iSCSI etc.

UEFI shell improves pre-boot testing & diagnostics experience.

(Unified) Extensible Firmware Interface

- Industry Standard Interface Between Firmware & OS
- Processor Architecture and OS Independent
- C Development Environment (EDK2/UDK)
- Rich GUI Pre-Boot Application Environment
- Includes Modular Driver Model
- UEFI executables: PE/COFF or TE executable files
- UEFI file system is FAT32
- UEFI OS uses UEFI compliant bootloader:

  `/efi/boot/bootx64.efi /efi/redhat/grub.efi`
- Secure Boot of Microsoft Windows 8 or above requires UEFI
- UEFI firmware supports booting legacy OS from legacy MBR via Compatibility Support Module (CSM)

(U)EFI Firmware

OS Kernel / Drivers

UEFI OS Loaders

DXE Driver

DXE Driver

DXE OROM

DXE OROM

UEFI App

UEFI App

UEFI Boot Loader

Bootx64.efi
Bootmgfw.efi

HDD

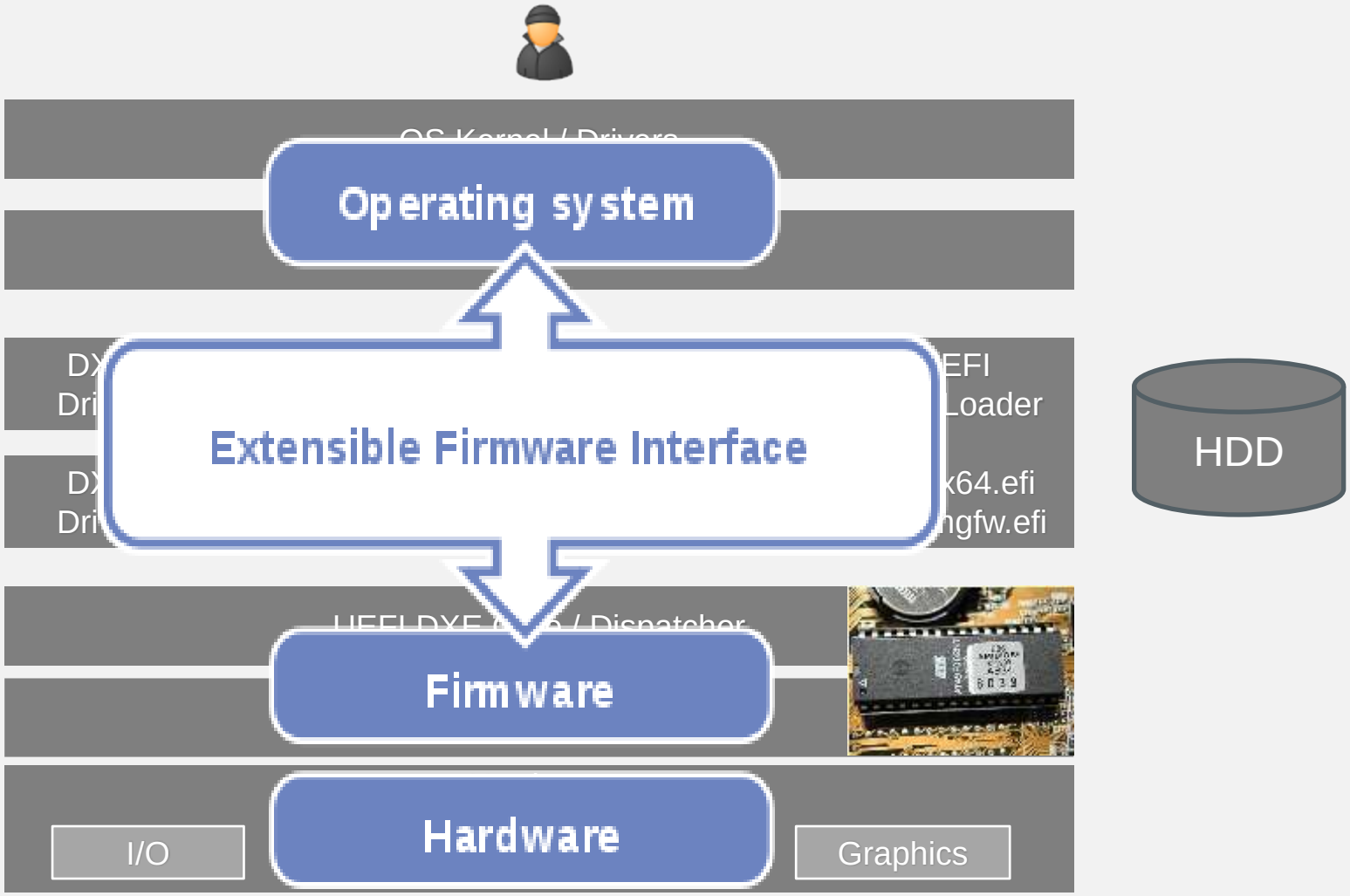
UEFI DXE Core / Dispatcher

System Firmware (SEC/PEI)

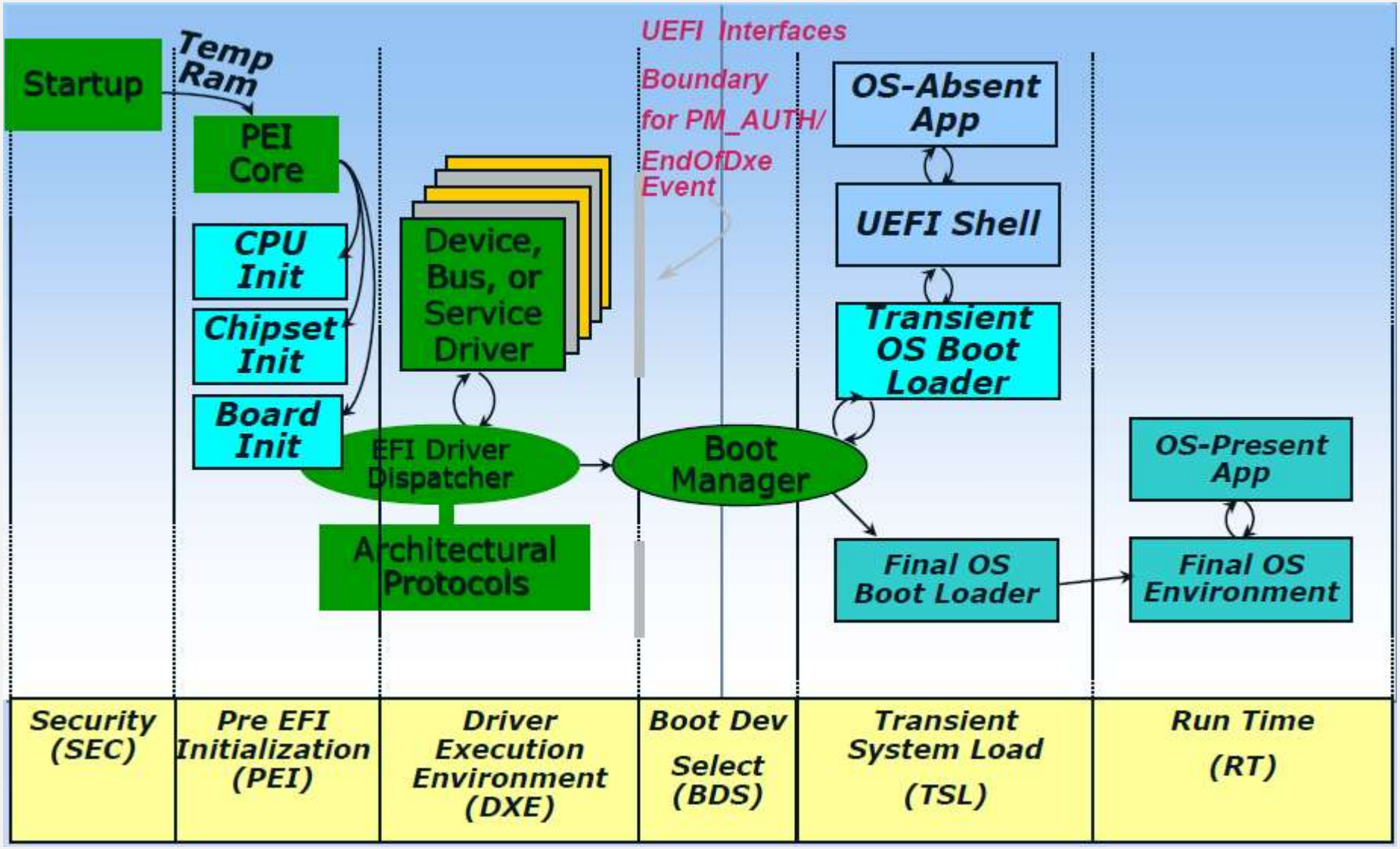
Hardware

I/O | Memory | Network | Graphics

(U)EFI Firmware



UEFI Boot



From Secure Boot, Network Boot, Verified Boot, oh my and almost every publication on UEFI

UEFI Configuration: UEFI “Variables”

UEFI variables contain configuration setup, vendor information, language information, input/output console, error console, and boot order setting, secure boot configuration, long information after capsule update, pointer to S3 boot script and so on.

Attributes of UEFI variables:

- NV (Non-Volatile)
- BS (Boot Service)
- RT (Run-Time)
- Authentication attributes

NV variables are stored in NVRAM in SPI flash memory

Windows 8+ provides user mode API to access run-time UEFI variables:
`GetFirmwareEnvironmentVariable`, **`SetFirmwareEnvironmentVariable`**

EDK I, EDK II, Tianocore

TianoCore is an open source implementation of **UEFI**, the **Unified Extensible Firmware Interface**:

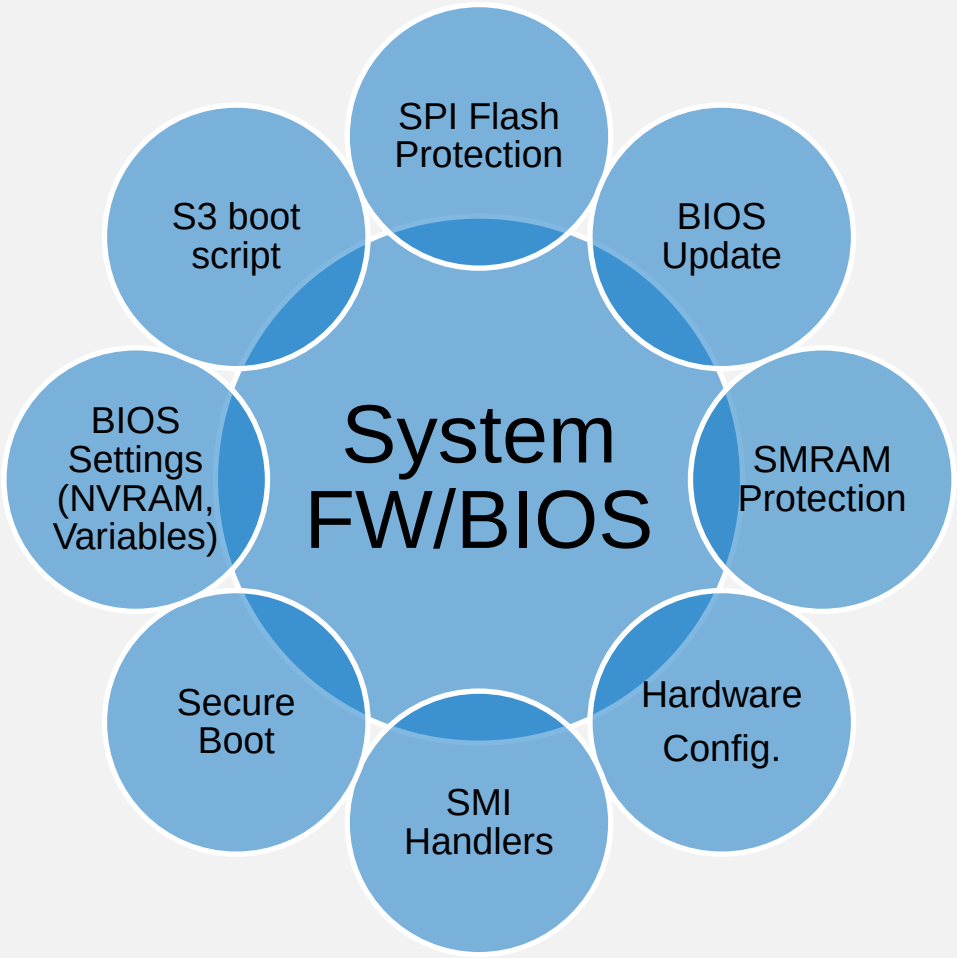
http://www.coreboot.org/TianoCore

EDK I - this is the older development environment

EDK II - a modern, feature-rich, cross-platform firmware development environment for the UEFI and PI specifications.

http://tianocore.sourceforge.net/wiki/Projects

BIOS Attack Surface



- Static and dynamic methods for BIOS analysis

CHIPSEC

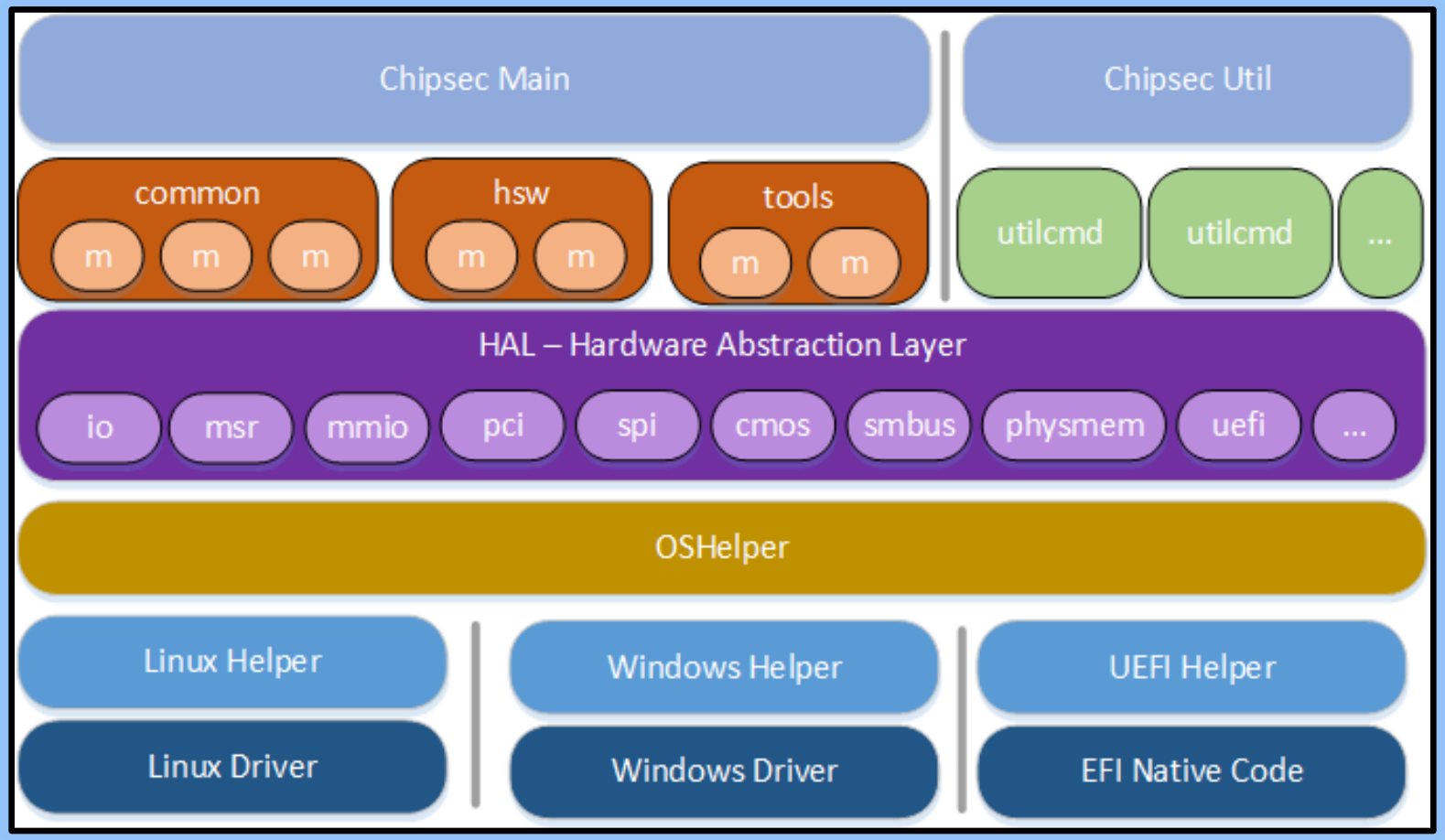
Platform Security Assessment tool:

https://github.com/chipsec/chipsec

Support: Windows, Linux, UEFI shell

CHIPSEC functionality:

1. Tests for known vulnerabilities in firmware
2. Tests for insufficient or incorrectly configured hardware protections
3. Hardware/firmware-level security tools
   - Fuzzing tools for firmware interfaces/formats
   - Manual security checkers (e.g. TE checker, BWP module,..)
4. Forensic analysis of the BIOS


Chipsec Main
Chipsec Util
common
m
m
m
hsw
m
m
tools
m
m
utilcmd
utilcmd
...
HAL – Hardware Abstraction Layer
io
msr
mmio
pci
spi
cmos
smbus
physmem
uefi
...
OSHelper
Linux Helper
Linux Driver
Windows Helper
Windows Driver
UEFI Helper
EFI Native Code

HW Abstraction Layer (HAL)

File name	Description
`hal/pci.py`	Access to PCIe configuration space
`hal/physmem.py`	Access to physical memory
`hal/msr.py`	Access to CPU resources (for each CPU thread): Model Specific Registers (MSR), IDT/GDT
`hal/mmio.py`	Access to MMIO (Memory Mapped IO) BARs and Memory-Mapped PCI Configuration Space (MMCFG)
`hal/spi.py`	Access to SPI Flash parts
`hal/ucode.py`	Microcode update specific functionality
`hal/io.py`	Access to Port I/O Space
`hal/smbus.py`	Access to SMBus Controller in the PCH
`hal/uefi.py`	Main UEFI component using platform specific and common UEFI functionality
`hal/uefi_common.py`	Common UEFI functionality (EFI variables, db/dbx decode, etc.)
`hal/uefi_platform.py`	Platform specific UEFI functionality (parsing platform specific EFI NVRAM, capsules, etc.)
`hal/interrupts.py`	CPU Interrupts specific functions (SMI, NMI)
`hal/cmos.py`	CMOS memory specific functions (dump, read/write)
`hal/cpuid.py`	CPUID information
`hal/spi_descriptor.py`	SPI Flash Descriptor binary parsing functionality

CHIPSEC for access HW resources

```
chipsec_util msr 0x200
chipsec_util mem read 0x41E 0x20
chipsec_util pci enumerate
chipsec_util pci 0x0 0x1F 0x0 0xDC byte
chipsec_util io 0x61 byte
chipsec_util mmcfg 0 0x1F 0 0xDC 1 0x1
chipsec_util mmio list
chipsec_util cmos dump
chipsec_util ucode id
chipsec_util smi 0x01 0xFF
chipsec_util idt 0
chipsec_util cpuid 1
chipsec_util spi info
chipsec_util spi read 0x700000 0x100000 bios.bin
chipsec_util decode spi.bin
chipsec_util uefi var-list
chipsec_util spd dump
chipsec_util acpi list
```

Summary of Modules in CHIPSEC

Issue	CHIPSEC Module	References
SMRAM Locking	`common.smm`	CanSecWest 2006
BIOS Keyboard Buffer Sanitization	`common.bios_kbrd_buffer`	DEFCON 16
SMRR Configuration	`common.smrr`	ITL 2009, CanSecWest 2009
BIOS Protection	`common.bios_wp`	BlackHat USA 2009, CanSecWest 2013, Black Hat 2013, NoSuchCon 2013
SPI Controller Locking	`common.spi_lock`	Flashrom, Copernicus
BIOS Interface Locking	`common.bios_ts`	PoC 2007
Secure Boot variables with keys and configuration are protected	`common.secureboot.variables`	UEFI 2.4 Spec , All Your Boot Are Belong To Us (here & here)
Memory remapping attack	`remap`	Preventing and Detecting Xen Hypervisor Subversions
DMA attack against SMRAM	`smm_dma`	Programmed I/O accesses: a threat to VMM?, System Management Mode Design and Security Issues
SMI suppression attack	`common.bios_smi`	Setup for Failure: Defeating Secure Boot
Access permissions to SPI flash descriptor	`common.spi_desc`	Flashrom
Access permissions to UEFI variables defined in UEFI Spec	`common.uefi.access_uefispec`	UEFI 2.4 Spec
Module to detect PE/TE Header Confusion Vulnerability	`tools.secureboot.te`	All Your Boot Are Belong To Us
Module to detect SMI input pointer validation vulnerabilities	`tool.smm.smm_ptr`	CanSecWest 2015

BIOS/Firmware Forensics: Online

Live system firmware analysis

```
chipsec_util spi info
chipsec_util spi dump rom.bin
chipsec_util spi read 0x700000 0x100000 bios.bin
chipsec_util spi disable-wp
chipsec_util uefi var-list
chipsec_util uefi var-read db
    D719B2CB-3D3A-4596-A3BC-DAD00E67656F db.bin
chipsec_util uefi var-write db D719B2CB-3D3A-4596-A3BC-DAD00E67656F
db.bin
chipsec_util uefi var-delete db D719B2CB-3D3A-4596-A3BC-DAD00E67656F
chipsec_util uefi s3bootscript
chipsec_util uefi tables
chipsec_util acpi list
```

BIOS/Firmware Forensics: Offline

Offline system firmware analysis

```
chipsec_util uefi keys PK.bin
chipsec_util uefi nvram vss bios.bin
chipsec_util uefi decode rom.bin
chipsec_util decode rom.bin
chipsec_util spidesc spi.bin
```

Reverse engineering of the BIOS

1. Dump BIOS from SPI chip (or download from vendor web-site)
   - Software method: using CHIPSEC tool
   - HW programmer, for example: dediprog
2. Unpack all PEI/DXE executables.
   - `chipsec_util decode rom.bin`
3. Load to IDA and have fun ☺
   - ida-efiutils - useful scripts for reverse engineer BIOS/UEFI binary (from snare): https://github.com/snare/ida-efiutils
   - Useful blogposts from: @d_olex and http://blog.cr4.sh/

MinnowMax as a platform for BIOS research

Open hardware platform

Baytrail single or dual core

From http://firmware.intel.com/projects

This project focus in on the firmware source code (and binary modules) required to create the boot firmware image for the MinnowBoard MAX. The UEFI Open Source (EDKII project) packages for MinnowBoard MAX are available at http://tianocore.sourceforge.net/wiki/EDK2. To learn more about getting involved in the UEFI EDKII project visit the How to Contribute page.

Support:

- Debug capability through UART (including BIOS source level debug)

- VMX, TPM2,…

- See more at: http://firmware.intel.com/projects#sthash.1oOc8srY.dpuf

- Symbolic execution methods for BIOS analysis

Excite tool

Symbolic Execution for BIOS Security

Based on S2E framework

Goal: automatic search for vulnerabilities

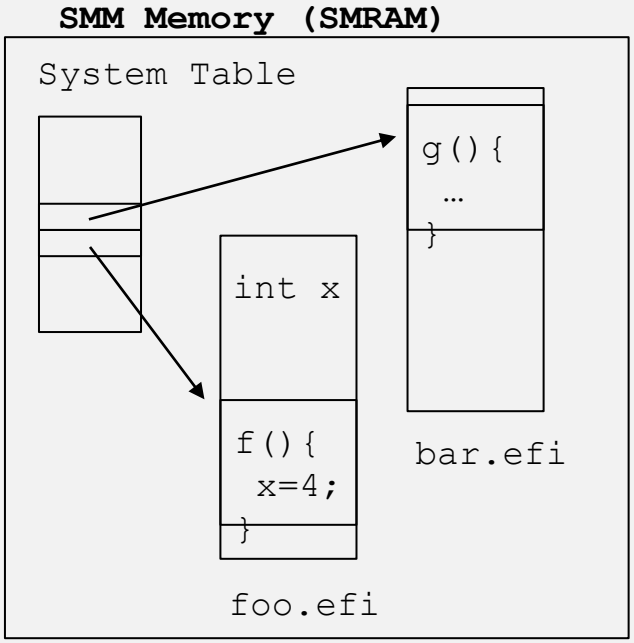
Target: SMI handlers

Approach: Search for vulnerabilities with S2E

- Integer overflow
- Pointers invalid or out of range buffer overflow
- Insecure memory references

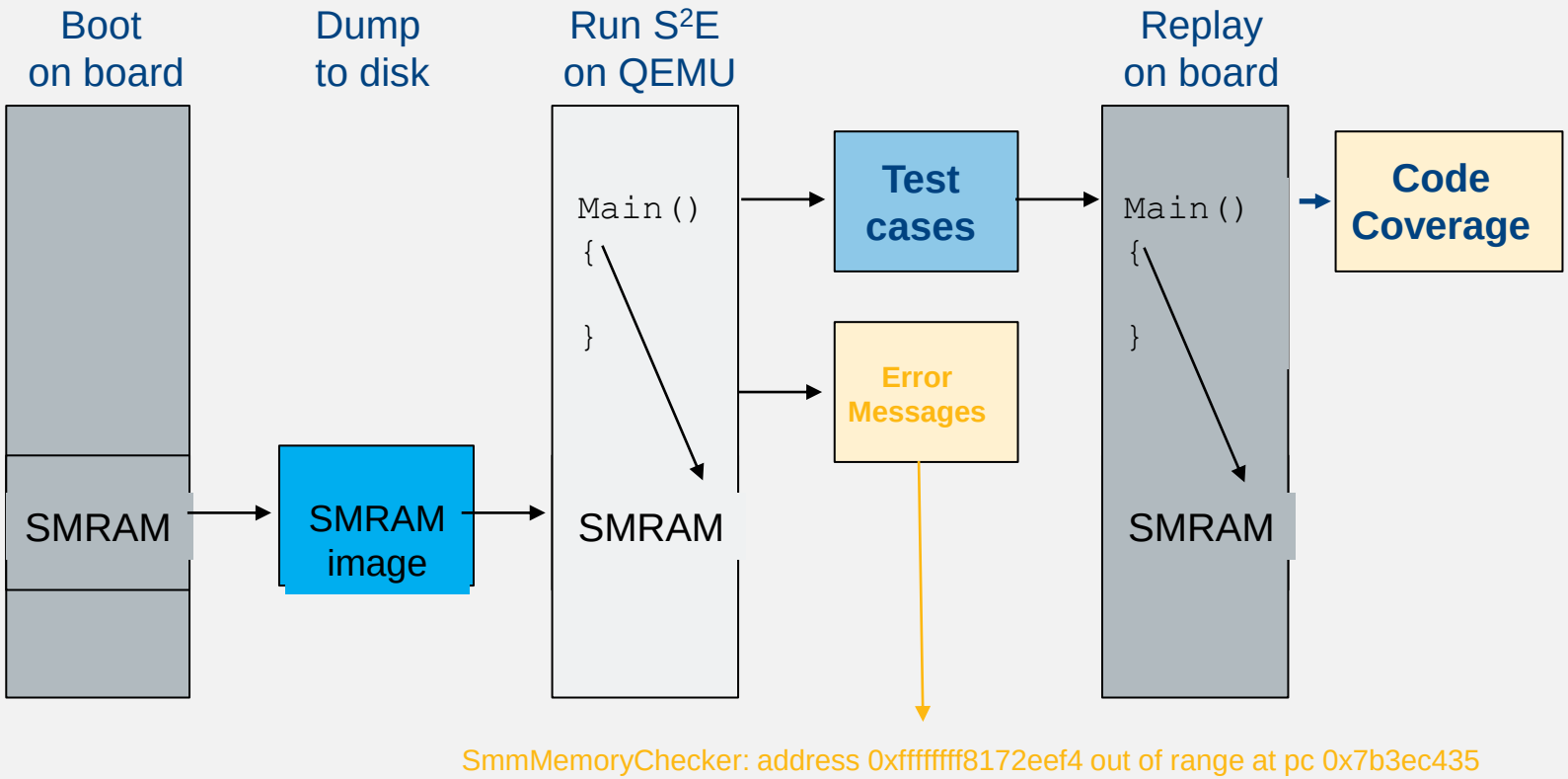
Test harness

Has to model the environment of the software under test



- SMRAM is the model
  - The code is there
  - The data and data layout there
- S^2E lets us use SMRAM
  - Boot to SMRAM and dump it
  - Load it into S^2E
  - Jump to an entry point
  - And execute symbolically

Execution flow



Our results

For a SMM handler, we need:

- SMRAM image, its base & size and the address of the entry point

We have three tools

- ***excite-generate***: generate test cases from Linux shell
  - Generates 4000 tests in 4 hours [1]
- ***excite-replay***: replay test cases from Linux shell
- ***s2eReplay.nsh*** - UEFI shell application:
  - replay test cases on the board in 30 min
  - and measure the code coverage

[1] Intel® Core™ 2 Quad 2.66 GHZ CPU with 2GB ram running Ubuntu 14.04 LTS

For SmmVariableHandler in MdeModulePkg\Universal\Variable\RuntimeDxe\VariableSmm.c

☣ Conclusions

Best practice: use static plus dynamic methods for BIOS analysis and check already existence tools, like CHIPSEC, Excite, ...

Symbolic execution – complicated and can be applied as advanced method

Acknowledgement

We'd like to thank the following teams or individuals for making the BIOS and EFI firmware a bit more secure

- Excite project: Mark Tuttle, Lee Rosenbaum
- Nick Adams, Aaron Frinzell, Sugumar Govindarajan, Jiewen Yao, Vincent Zimmer, Bruce Monroe, John Loucaides from Intel
- Corey Kallenberg, Xeno Kovah, Rafal Wojtczuk, @snare, Trammell Hudson, Dmytro Oleksiuk, Pedro Velaça
- UEFI Forum (USRT, USST), OEMs and IBVs who suggest solutions