A Tale of One
Software Bypass of
Windows 8 Secure Boot

Yuriy Bulygin
Andrew Furtak
Oleksandr Bazhaniuk
• UEFI and Bootkits
• Windows 8 Secure Boot
• Attacking Secure Boot
• Recommendations
Unified Extensible Firmware Interface (UEFI)
Unified Extensible Firmware Interface (UEFI)
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 Bootkits

- Malware
- OS Kernel / Drivers
- UEFI OS Loaders
  - UEFI OROM
  - DXE Driver
  - UEFI Boot Loader
    - Bootx64.efi
    - Bootmgfw.efi
- UEFI DXE Core / Dispatcher
- System Firmware (SEC/PEI)
- Hardware
  - I/O
  - Memory
  - Network
  - Graphics
UEFI Bootkits

- Malware
- OS Kernel / Drivers
- UEFI OS Loaders
  - UEFI OROM
  - DXE Driver
  - UEFI OROM
  - DXE Driver
- UEFI Boot Loader
  - Bootx64.efi
  - Bootmgfw.efi
- UEFI DXE Core / Dispatcher
- System Firmware (SEC/PEI)
- Hardware
  - I/O
  - Memory
  - Network
  - Graphics
Replacing Windows Boot Manager
EFI System Partition (ESP) on Fixed Drive
ESP\EFI\Microsoft\Boot\bootmgfw.efi
UEFI technology: say hello to the Windows 8 bootkit! by ITSEC

Replacing Fallback Boot Loader
ESP\EFI\Boot\bootx64.efi
UEFI and Dreamboot by Sébastien Kaczmarek, QUARKSLAB

Adding New Boot Loader (bootkit.efi)
Modified BootOrder / Boot##### EFI variables

UEFI Bootkits
UEFI Bootkits

UEFI DXE Core / Dispatcher

UEFI OS Loaders

OS Kernel / Drivers

System Firmware (SEC/PEI)

Hardware
- I/O
- Memory
- Network
- Graphics

UEFI Boot Loader

UEFI Boot Loader

Bootx64.efi

Bootmgfw.efi

UEFI OROM

UEFI OROM

DXE Driver

DXE Driver

HDD

Malware

Malware
Adding/Replacing DXE Driver

Stored on Fixed Drive
Not embedded in Firmware Volume (FV) in ROM
Modified DriverOrder + Driver#### EFI variables
UEFI Bootkits

- System Firmware (SEC/PEI)
- UEFI DXE Core / Dispatcher
- OS Kernel / Drivers
- UEFI OS Loaders
- Malware
- UEFI Boot Loader
  - Bootx64.efi
  - Bootmgfw.efi
- Hardware
  - I/O
  - Memory
  - Network
  - Graphics

HDD
Patching UEFI “Option ROM”

UEFI DXE Driver in Add-On Card (Network, Storage..)
Non-Embedded in FV in ROM

Mac EFI Rootkits by @snare, Black Hat USA 2012
Replacing OS Loaders (winload.elf, winresume.elf)

Patching GUID Partition Table (GPT)
By booting this system up you agree to have **no** expectation of **privacy** in any communications or data, transiting or stored on this system. Any communications or data may be **monitored**, **intercepted**, **recorded** and may be **disclosed** for any purpose.

press any key to continue...
Replacing legacy OS boot with UEFI boot will also replace legacy M/VBR bootkits with UEFI bootkits. So can bootkit problem be fixed?
Windows 8
Secure Boot
System Firmware and NVRAM Are in ROM
UEFI Firmware Relies on Secure Update
DXE Verifies Non-Embedded UEFI OROMs
Signed BIOS Update

- UEFI OROM
- UEFI OROM
- DXE Driver
- DXE Driver

UEFI DXE Core / Dispatcher

System Firmware (SEC/PEI)

Hardware
- I/O
- Memory
- Network
- Graphics

DXE Verifies Non-Embedded DXE Drivers
DXE Core Verifies UEFI Applications
Signed BIOS Update

UEFI OROM
UEFI App
DXE Driver
UEFI Boot Loader
Bootx64.efi Bootmgfw.efi

UEFI DXE Core / Dispatcher
System Firmware (SEC/PEI)

Hardware
I/O Memory Network Graphics

DXE Core Verifies UEFI Boot Loader(s)
Signed BIOS Update

- UEFI Secure Boot
- UEFI DXE Core / Dispatcher
- System Firmware (SEC/PEI)
- Hardware:
  - I/O
  - Memory
  - Network
  - Graphics

- UEFI OROM
- UEFI App
- DXE Driver
- UEFI Boot Loader:
  - Bootx64.efi
  - Bootmgfw.efi
UEFI Boot Loader Verifies OS Loader
OS Loader Verifies OS Kernel

- Signed BIOS Update
- OS Kernel / Early Launch Anti-Malware (ELAM)
- UEFI OS Loaders (winload.efi, winresume.efi)
- UEFI DXE Core / Dispatcher
- UEFI OROM
- UEFI App
- DXE Driver
- UEFI Boot Loader
- Bootx64.efi
- Bootmgfw.efi
- System Firmware (SEC/PEI)
- Hardware
  - I/O
  - Memory
  - Network
  - Graphics

UEFI Secure Boot
Signed BIOS Update

OS Kernel / Early Launch Anti-Malware (ELAM)

UEFI OS Loaders (winload.efi, winresume.efi)

System Firmware (SEC/PEI)

UEFI DXE Core / Dispatcher

Hardware
- I/O
- Memory
- Network
- Graphics

OS Kernel Verifies OS Device Drivers
Windows 8 Secure Boot

Signed BIOS Update

- OS Driver
- OS Kernel / Early Launch Anti-Malware (ELAM)
- UEFI OS Loaders (winload.efi, winresume.efi)

UEFI OROM

- UEFI App
- DXE Driver
- UEFI Boot Loader (Bootx64.efi, Bootmgfw.efi)

UEFI DXE Core / Dispatcher

System Firmware (SEC/PEI)

Hardware
- I/O
- Memory
- Network
- Graphics

Windows 8 Secure Boot
**Platform Key (PK)**
- Verifies KEKs
- Platform Vendor’s Cert

**Key Exchange Keys (KEKs)**
- Verify db and dbx
- Earlier rev’s: verifies image signatures

**Authorized Database (db)**

**Forbidden Database (dbx)**
- X509 Certificates, image SHA1/SHA256 hashes of allowed and revoked images
- Earlier rev’s: RSA-2048 public keys, PKCS#7 Signatures
Secure Boot Key Protections

**Non-Volatile (NV)**
- Stored in SPI Flash based NVRAM

**Boot Service (BS)**
- Accessible to DXE drivers / Boot Loaders at boot time

**Run-Time (RT)**
- Accessible to the OS through run-time UEFI SetVariable/GetVariable API

**Time-Based Authenticated Write Access**
- Signed with time-stamp (anti-replay)
- PK cert verifies PK/KEK update
- KEK verifies db/dbx update
- certdb verifies general authenticated EFI variable updates
Certificate:

Data:

Version: 3 (0x2)
Signature Algorithm: sha256WithRSAEncryption
Issuer: CN=ASUSTeK MotherBoard PK Certificate
Validity
Not Before: Dec 26 23:34:50 2011 GMT
Not After : Dec 26 23:34:49 2031 GMT
Subject: CN=ASUSTeK MotherBoard PK Certificate
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
Public-Key: (2048 bit)
Modulus:
...
Exponent: 65537 (0x10001)
X509v3 extensions:
2.5.29.1:
?=@:/0-1+0)U."ASUSTeK MotherBoard PK
Certificate..SA...H6...i.
Signature Algorithm: sha256WithRSAEncryption
...
-----BEGIN CERTIFICATE-----
MIIDRjCCAi6gAwIBAgIQU0HgFcQ6+KhINrml/2kUiDANBgkqhkiG9w0BAQsFADAk
...
-----END CERTIFICATE-----
SecureBoot
- Enables/disables image signature checks

SetupMode
- PK is installed (USER_MODE) or not (SETUP_MODE)
- SETUP_MODE allows updating KEK/db(x), self-signed PK

CustomMode
- Modifiable by physically present user
- Allows updating KEK/db/dbx/PK even when PK is installed

SecureBootEnable
- Global non-volatile Secure Boot Enable
- Modifiable by physically present user
PK variable exists in NVRAM?
- Yes. Set SetupMode to USER_MODE
- No. Set SetupMode to SETUP_MODE

SecureBootEnable variable exists in NVRAM?
- Yes
  - SecureBootEnable is SECURE_BOOT_ENABLE and SetupMode is USER_MODE? Set SecureBoot to ENABLE
  - Else? Set SecureBoot to DISABLE
- No
  - SetupMode is USER_MODE? Set SecureBoot to ENABLE
  - SetupMode is SETUP_MODE? Set SecureBoot to DISABLE
DxeImageVerificationLib defines policies applied to different types of images and on security violation

IMAGE_FROM_FV (ALWAYS_EXECUTE), IMAGE_FROM_FIXED_MEDIA, IMAGE_FROM_REMOVABLE_MEDIA, IMAGE_FROM_OPTION_ROM

ALWAYS_EXECUTE, NEVER_EXECUTE,
ALLOW_EXECUTE_ON_SECURITY_VIOLATION
DEFER_EXECUTE_ON_SECURITY_VIOLATION
DENY_EXECUTE_ON_SECURITY_VIOLATION
QUERY_USER_ON_SECURITY_VIOLATION

Image Verification Policies
Let’s have a look at the Secure Boot image verification process

SecurityPkg\Library\DxeImageVerificationLib

Image Verification Policy?

- **(IMAGE_FROM_FV)** • ALWAYS_EXECUTE? **EFI_SUCCESS**

- **NEVER_EXECUTE?** • **EFI_ACCESS_DENIED**
• SecureBoot EFI variable doesn’t exist or equals to SECURE_BOOT_MODE_DISABLE? **EFI_SUCCESS**

• File is not valid PE/COFF image? **EFI_ACCESS_DENIED**

• SecureBootEnable NV EFI variable doesn’t exist or equals to SECURE_BOOT_DISABLE? **EFI_SUCCESS**

• SetupMode NV EFI variable doesn’t exist or equals to SETUP_MODE? **EFI_SUCCESS**
Image signed?

• No
  – Image SHA256 hash in dbx? **EFI_ACCESS_DENIED**
  – Image SHA256 hash in db? **EFI_SUCCESS**

• Yes
  For each signature in PE file:
  – Signature verified by root/intermediate cert in dbx? **EFI_ACCESS_DENIED**
  – Image hash in dbx? **EFI_ACCESS_DENIED**
  For each signature in PE file:
  – Signature verified by root/intermediate cert in db? **EFI_SUCCESS**
  – Image hash in db? **EFI_SUCCESS**

• **EFI_ACCESS_DENIED**
Secure Boot in Action

Secure Boot Violation

Invalid signature detected. Check Secure Boot Policy in Setup

OK
Windows 8/UEFI Secure Boot is pretty important protection from boot malware!
Attacking Windows 8 Secure Boot
Just turn it off in the BIOS setup screen ;)}
Just to be clear, the issues are in the implementation of Secure Boot and required UEFI firmware protections on certain platforms.
We think Windows 8 Secure Boot looks like this

Windows 8 Secure Boot
(Microsoft)

Or more like this

Windows 8 Secure Boot
UEFI Secure Boot
(UEFI)
How exciting! … But still not close
The Reality Is Much More Exciting
Windows 8 Secure Boot is only secure when ALL platform/BIOS vendors do a couple of things correctly
- Allow signed UEFI firmware updates only
- Protect UEFI firmware in SPI flash from direct modification
- Protect firmware update components (inside SMM or DXE on reboot)
- Program SPI controller and flash descriptor securely
- Protect SecureBootEnable/CustomMode/PK/KEK/db(x) in NVRAM
- Implement VariableAuthenticated in SMM and physical presence checks
- Protect SetVariable runtime API
- Securely disable Compatibility Support Module (CSM), unsigned legacy Option ROMs and MBR boot loaders
- Configure secure image verification policies (no ALLOW_EXECUTE)
- Build platform firmware using latest UEFI/EDK sources
- Correctly implement signature verification and crypto functionality
- And don’t introduce a single bug in all of this...
Windows Hardware Certification Requirements: Client and Server Systems

3 When Secure Boot is Enabled, CSM must NOT be loaded
7 Secure Boot must be rooted in a protected or ROM-based Public Key
8 Secure firmware update process
9 Signed Firmware Code Integrity Check
14 No in-line mechanism is provided whereby a user can bypass Secure Boot failures and boot anyway
...

Windows 8 Secure Boot Requirements
What If UEFI BIOS Updates Are Not Signed?
When UEFI Firmware Updates Are Not Signed
No luck 😞

UEFI firmware update capsules are signed

RSA-PSS 2048 / SHA-256 / e=F₄

Wait, let’s check one little thing...
Can We Write to UEFI Firmware in ROM?
So UEFI firmware updates are signed but firmware is directly writeable in SPI flash? So is NVRAM with EFI variables. Hmm... What could go wrong?

**Hint:** Malware could patch DXE Image Verification driver in ROM or it could change persistent Secure Boot keys/configuration in NVRAM.
When Firmware Is Not Protected in ROM

Signed BIOS Update

Hardware
- I/O
- Memory
- Network
- Graphics

OS Driver

OS Exploit

OS Kernel

UEFI OS Loaders

DXE Driver

UEFI Boot Loader
- Bootx64.efi
- Bootmgfw.efi

UEFI DXE Core / Dispatcher

System Firmware (SEC/PEI)
Malware Modifies UEFI Firmware in ROM
(directly programming SPI controller)
Signed BIOS Update

DXE Driver

UEFI DXE Core / Dispatcher

System Firmware (SEC/PEI)

Hardware
- I/O
- Memory
- Network
- Graphics

OS Driver

UEFI OS Loaders

OS Kernel

Install UEFI Bootkit

OS Exploit

OS Driver

DXE Driver

UEFI Bootkit

Signed BIOS Update

Then Installs UEFI Bootkit
Firmware Doesn’t Enforce Secure Boot
Signed BIOS Update

UEFI Bootkit Now Patches OS Loaders/Kernel
**Exploit Strategies**

1. **Patch DXE ImageVerificationLib driver code**
   - Differ from one platform/vendor to another
   - Different versions of EDK and BIOS Cores

2. **Replace/add hash or Cert in db**
   - Bootkit hash is now allowed
   - Generic exploit, independent of the platform/vendor
   - Can be found by inspecting “db” in ROM

3. **Replace/add RootCert in KEK or PK with your own**
   - Bootkit signature is now valid
Clear `SecureBootEnable` variable

- Despite UEFI defines “SecureBootEnable” EFI variable, platform vendors store Secure Boot Enable in platform-specific places.
- Format of EFI NVRAM and EFI variable in ROM is platform/vendor specific.
- May require modification in multiple places in NVRAM ➔ parsing of platform specific NVRAM format.
- Replacing entire NVRAM or even entire BIOS region to SB=off state is simpler but takes a while.
Parsing Proprietary EFI NVRAM
Corrupt Platform Key EFI variable in NVRAM

- Name ("PK") or Vendor GUID (8BE4DF61-93CA-11D2-AA0D-00E098032B8C)

- Recall that AutenticatedVariableService DXE driver enters Secure Boot SETUP_MODE when correct “PK” EFI variable cannot be located in EFI NVRAM

- Main volatile SecureBoot variable is then set to DISABLE

- ImageVerificationLib then assumes Secure Boot is off and skips Secure Boot checks

- Generic exploit, independent of the platform/vendor

- 1 bit modification!

Exploit Strategies
Windows 8 HW Certification Requires Platforms to Protect UEFI Firmware and NVRAM with Secure Boot keys!

7. Mandatory. **Secure Boot must be rooted in a protected or ROM-based Public Key.** Secure Boot must be rooted in an RSA public key with a modulus size of at least 2048 bits, and either be based in unalterable ROM or otherwise protected from alteration by a secure firmware update process, as defined below.
Secure Boot Is Enabled
Corrupting Platform Key in NVRAM
Platform Key Is De-Installed
Для загрузки необходим номер вашей кредитной карты на securecreditcardz.ru
Back to Setup Mode ➔ Secure Boot Is Off
Demo 1

Attacking Windows 8 Secure Boot on ASUS VivoBook Q200E
When UEFI firmware is not adequately protected (in ROM or during update), subverting UEFI Secure Boot is not the only thing to worry about!
S-CRTM and TPM based Measured Boot including Full-Disk Encryption solutions relying on the TPM can also be subverted

Evil Maid Just Got Angrier
BIOS Chronomancy by John Butterworth, Corey Kallenberg, Xeno Kovah
Or you can get infected with UEFI BIOS or SMM malware
a.k.a. “extremely persistent malware” © .gov

Persistent BIOS Infection by Anibal Sacco, Alfredo Ortega
Hardware Backdooring is Practical by Jonathan Brossard
The Real SMM Rootkit by core collapse
SMM Rootkits by Shawn Embleton, Sherri Sparks, Cliff Zou
Huh! It requires kernel exploit?
Why Not Just Directly Modify Secure Boot Keys from the OS? There’s an API for that

```python
# chipsec_util.py uefi writevar PK 8BE4DF61-93CA-11D2-AA0D-00E098032B8C PK_forged.bin
SetFirmwareEnvironmentVariable failed
[Error 5] Access is denied.
```

```
C:\Users\bh2013\Desktop\bh2013\chipsec-1.0\python chipsec_util.py uefi writevar PK 8BE4DF61-93CA-11D2-AA0D-00E098032B8C PK_forged.bin
CHIPSEC] Writing EFI variable Name='PK' GUID={8BE4DF61-93CA-11D2-AA0D-00E098032B8C} from 'PK_forged.bin' via Variable API.
Writing EFI variable:
Name : PK
GUID : 8BE4DF61-93CA-11D2-AA0D-00E098032B8C
[CHIPSEC] [Error 5] Access is denied.
```
Remember Secure Boot Key variables are “Authenticated Write Access”

You have to sign EFI variable and have corresponding X509 Cert in NVRAM (PK/KEK/certdb)
Is it possible to bypass Windows 8 Secure Boot and install UEFI bootkit by remote user mode exploit?

Coordinated disclosure of multiple vulnerabilities to affected BIOS and platform vendors is ongoing but we can offer a demo
Demo 2

Attacking Windows 8 Secure Boot from user-mode
Now what?
Only signed updates should be allowed
- Signed UEFI Capsule based update via S3/reset
- Run-time update from within SMM only

Protect UEFI firmware in ROM
- Use BIOS Control to enable write protection of the entire BIOS region in SPI flash
- Use Protected Range registers to write-protect ranges of SPI flash

Protect EFI variable store (NVRAM) in ROM
- NVRAM contains Secure Boot keys and NV configuration

Measuring Secure Boot configuration into PCR[7] may prevent or complicate certain exploits
- Facilitates detection via remote attestation
- Windows BitLocker may seal encryption keys to PCR[7]
- Microsoft Hardware Certification Requirements
- **Make sure platform has Windows 8 Logo**
  - Such platform has to adhere to the security requirements in `System.Fundamentals.Firmware.UEFISecureBoot`

- **Check if UEFI firmware updates are signed**
  - Corrupt firmware update binary and feed it to the BIOS update utility

- **Ask if and how platform adheres to the BIOS Protection Guidelines (NIST SP 800-147)**
Black Hat organizers and review committee

**Bruce Monroe**, John Loucaides, Ben Parmeter, Paul Alappat; Microsoft’s MSRC and Secure Boot team for coordinating vendor disclosures

Nicholas Adams, Kirk Brannock, doughty, Dhinesh Manoharan, Brian Payne, Mickey Shkatov, Vincent Zimmer, Monty Wiseman for help & support of this work

Greetz to Misha, @j4istal, @apebit, @drraid, sharkey, secoeites, @tobyhush, @matrosoyv
• BIOS SECURITY by John Butterworth, Corey Kallenberg and Xeno Kovah

• UART THOU MAD? by Toby Kohlenberg and Mickey Shkatov

• ANDROID: ONE ROOT TO OWN THEM ALL by Jeff Forristal

Don’t Miss These Talks!
Thank You!
@c7zero
@Abazhanyuk
andrew.furtak@gmail.com