presented by

Self-signing The Linux Kernel (The Hobbyist Approach)

Zachary Bobroff – American Megatrends, Inc.

LINUXCON NORTH AMERICA

August 22, 2014

Agenda

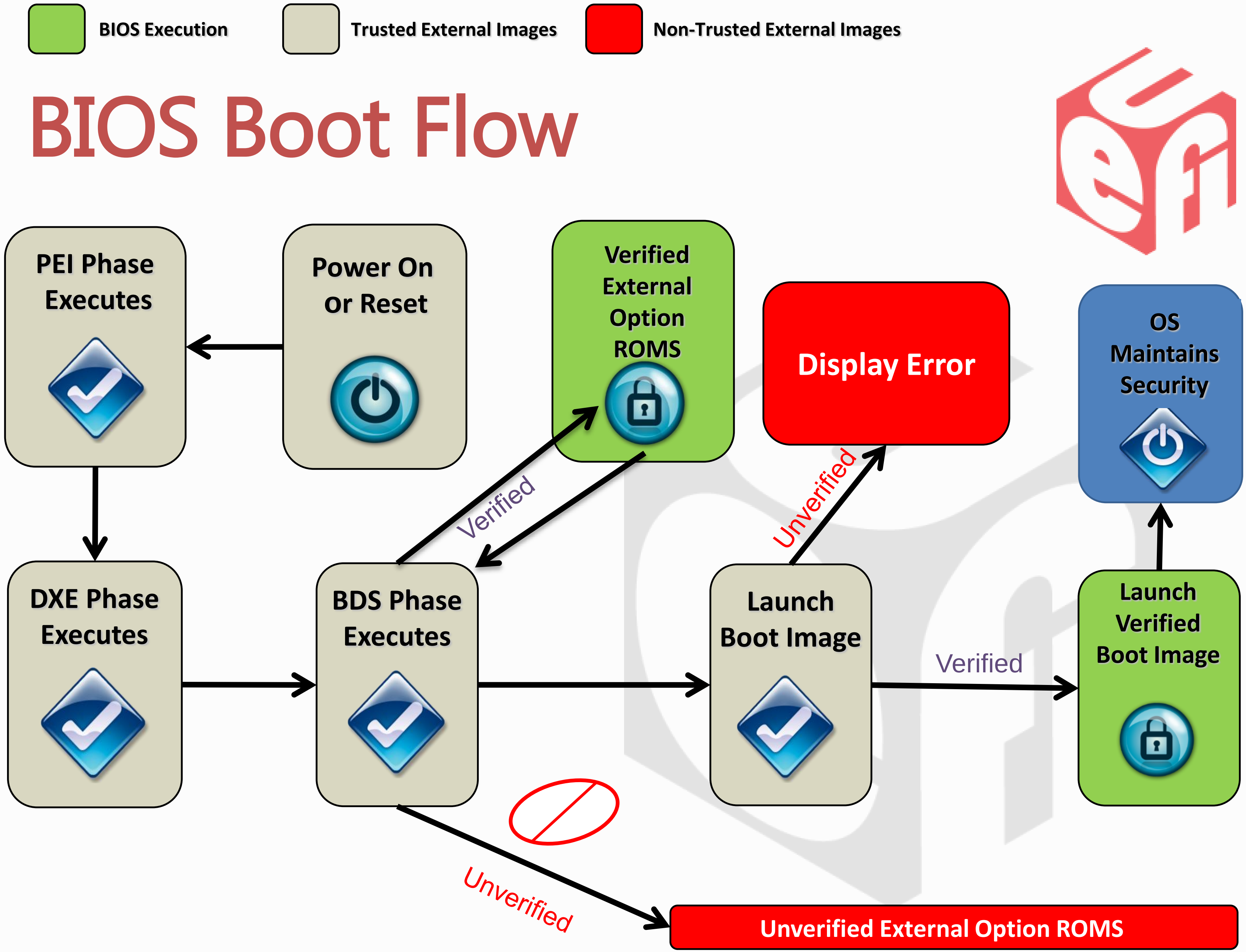
- Introduction
- Linux UEFI Secure Boot Overview
- Signing a Kernel
- Adding keys to the MoK DB
- Adding keys to the UEFI DB
- Call to Action

Introduction

Introduction

- UEFI Secure Boot
  - Introduced in UEFI version 2.3.1
  - Assure the System boot process does not run any malicious/unverified code
  - All external images to the BIOS must be signed and are verified against a signature database before execution
  - Implementation follows the chapter 27 of UEFI spec and uses RSA-2048 Keys, X509 certificates, SHA256 and PKCS#1 v1.5
- Secure Boot makes whole EFI FW (BIOS) a root of trust to an EFI OS
  - No in-line methods exists to bypass the image verification

BIOS Boot Flow



Secure Boot Purpose

- Everyone knows UEFI Secure Boot is about making sure only properly signed and verified images are executed
- The main overall reason for UEFI Secure Boot is to prevent any unauthorized software from being loaded in the pre-boot space
  - An attack in this pre-boot space can be referred to as a man in the middle attack or a root kit
  - Both attacks can be undetectable to the OS and pass bad information and has access to all system resources

Preventing this type of attack is important to all devices and systems!

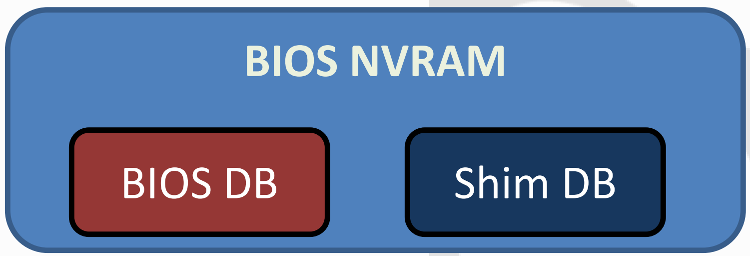
UEFI Secure Boot And Linux

Secure Boot And Linux

- Linux is traditionally booted using a bootloader like GRUB
  - Grub loads a kernel and ram disk into memory and launches the kernel
- Loading images into memory from GRUB is done through UEFI services
  - Each image will be properly validated by the UEFI services before passing control back to GRUB
  - Requires the proper signing of the kernel and ram disk!
- The Linux community at large has made use of a bootloader called Shim
  - https://github.com/mjg59/shim

Shim

- Shim is an EFI bootloader that provides an easier method for Linux to manage keys for its own signed images
- Shim provides a secondary key database that can be managed by the user
  - Not all OEMs provide easy ways to manage BIOS key databases



- Shim key DB is managed by:
  - Mok-util -- OS level application, indicates change request by user
  - MokManager -- EFI application, launched when change is requested and helps user make their requested change
  - All change requests require password based authentication

Booting Via Shim

- UEFI firmware gives control to Shim boot loader (signed by UEFI CA)
- Shim validates its own key DB for integrity
- Shim publishes its own security protocol
- Shim uses the BIOS DB or Shim DB to verify and launch GRUB2
- GRUB2 uses shim security protocol to verify and launch Linux kernel and ram disk
- Linux kernel is now responsible for maintaining system integrity

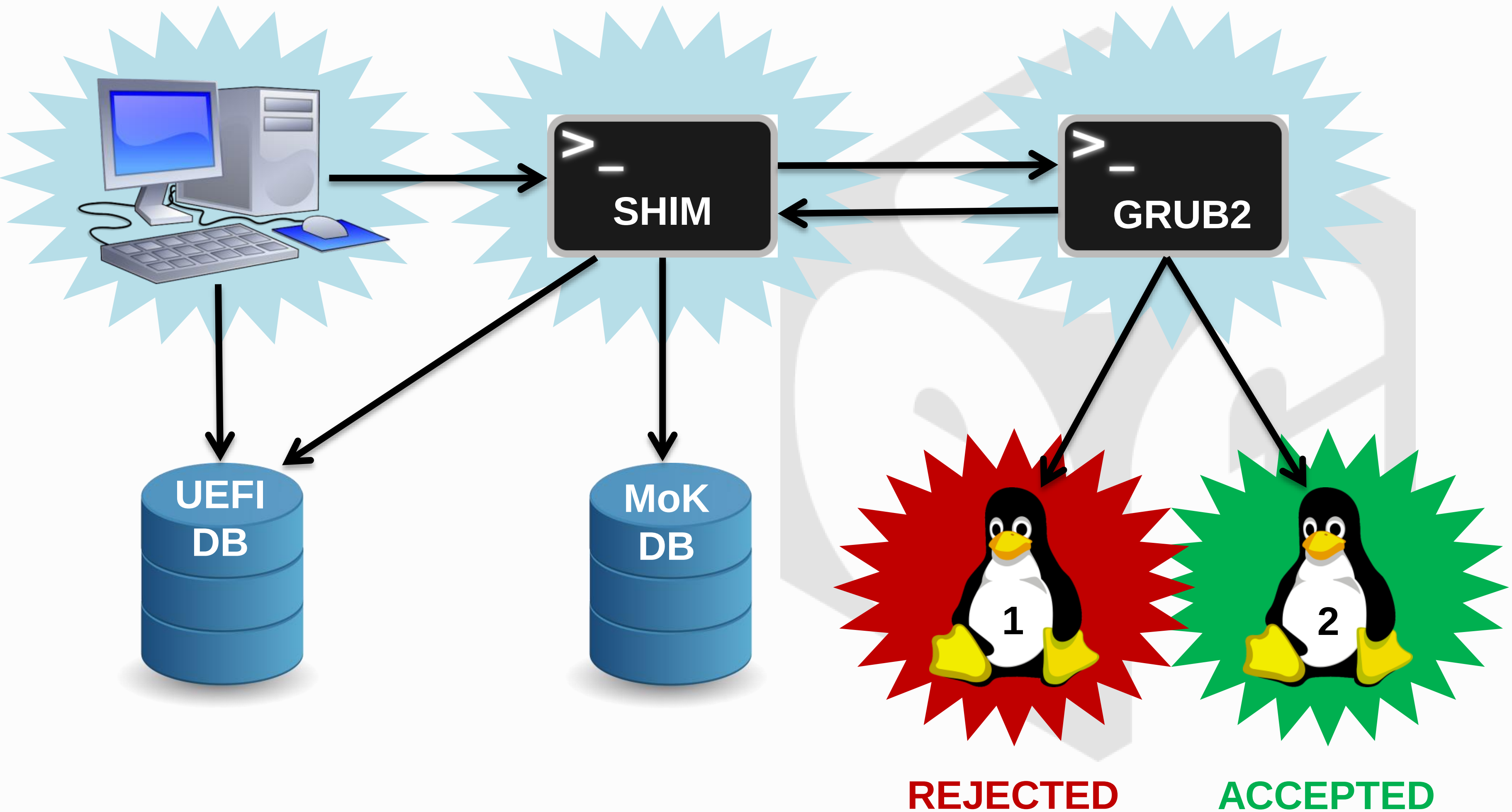
Any improperly signed image will result in a boot refusal and error screen

Example Boot Failure Screen

```
error: /vmlinuz-3.16.0-rc6 has invalid signature.
error: you need to load the kernel first.

Press any key to continue..._
```

Shim Bootflow



Signing Linux Kernel

Tools Used In Signing

- OpenSSL
- Kernel from distro or www.kernel.org
- Scripts for signing done by AMI
- SUSE well documents the process of signing here
- Fedora specific tools
  - Certutil provided by Fedora
  - PESign provided by Fedora
- Ubuntu specific tools
  - SBSign provided by Ubuntu

Steps Taken Before Signing

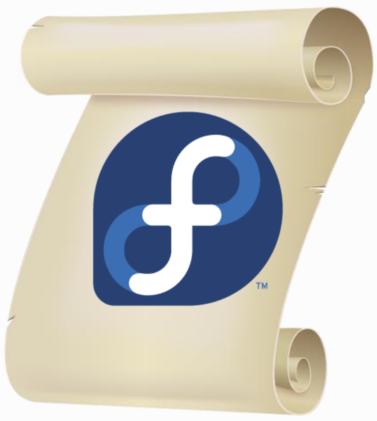
- Download proper kernel image from either www.kernel.org or from distribution
- Build the kernel to whatever configuration is desired
- Build and install the kernel modules
- Install the kernel modules

Fedora Specific Script 1 Steps

- Generate new key
- Export key in proper format
- Import certificate into NSS certificate DB
- Convert certificate into DER format
  - Common format both BIOS and Shim DB use
- Invoke mok-util requesting insertion of key on next reboot
  - Key is also copied to EFI partition for use by mok-util

Fedora Specific Script 2 Steps

- Signs the kernel using pesign utility
- Verifies kernel signature exists



Ubuntu Specific Script 1 Steps

- Generate a key
- Export key in proper format
- Invoke mok-util requesting insertion of key on next reboot
  - Key is also copied to EFI partition for use by mok-util

Ubuntu Specific Script 2 Steps

- Signs the kernel using sbsign utility

Signature

Common After Signing Steps

- Copy signed kernel to the EFI boot partition
- Modify GRUB2 configuration to allow booting to newly built and custom signed kernel

Method 1

Adding Keys To The MoK DB

Generate A Signing Certificate

Generate a signing certificate that will be used to sign your Custom Kernel, and generate the associated DER formatted certificate.

- Instructions can be found on the OpenSuse Wiki under the “OpenSuse:UEFI” article.

Add The Keys Into The Mok Database

- Keys imported into the Mok database must be in DER format
  - Mokutil –import <importcertificate.cer>
- On reboot follow MokManager instructions to add certificate to the MOK DB

Method 2

Adding Keys To The UEFI DB

Adding Keys To The UEFI DB (1)

- Boot into the BIOS Setup
- On the Security Page of the BIOS setup, enter into the Secure Boot Menu
- Change the Secure Boot mode from “Standard” to “Custom”, then enter into the Key Management sub menu
- In the Authorized Signatures, submenu, select “Append Key”

Adding Keys To The UEFI DB (2)

- Select Load key from external media
- Find the device type and navigate to the certificate
- Select Public Key Certificate for the import File Format
- Confirm the Update of ‘db” with the certificate

MoK Vs UEFI DB (1)

- Since Mok only functions within the Shim environment, it will not effect UEFI bootable external media (extra security)
- MoK DB is OS interactive and could be more susceptible to Malware
- Keep generated keys secure in either case
  - Virus can find the keys on the system and attempt to sign the virus code

Demonstration

- Demonstration of:
  - Generating a key
  - Signing a Linux kernel
  - Adding it to GRUB2 as a boot option

Demonstration

- Demonstration of:
  - Adding keys for new kernel to UEFI DB

Demonstration

- Demonstration of:
  - Adding keys for new kernel to MoK DB

Call to Action

Call to Action

- Investigate if UEFI Secure Boot would work in your environment
  - Secure Boot is designed to work well with any UEFI OS!
- Try signing your own kernel and booting it with Secure Boot on and off
  - Secure any keys used in signing!
- If process could be simplified become an active member of UEFI.org and offer your opinion

For more information on the Unified EFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by

American Megatrends