UEFI Secure Boot

Olaf Kirch
SUSE, Director SUSE Linux Enterprise
okir@suse.com
Agenda

• Introduction to Secure Boot
• Why this is a challenge for Open Source
• Secure Boot: SUSE® solution
Secure Boot in 60 Seconds
BIOS: Basic Input Output System

- Has been around for a while
- Will boot anything you give it
  - The Operating System you installed
  - A trojaned copy of your operating system
  - A virus written to the Master Boot Record of your disk
- Malware writers have become very good at concealing their steps
  - You can no longer trust that the OS you run is the OS you wanted to run
BIOS: What It's Supposed To Do

BIOS \rightarrow OS Loader \rightarrow OS Start
BIOS: What You Can Make It Do

All that's needed for the attacker is root access to your hard disk.
But... Does It Have To Be This Way?

- Wouldn't it be nice if you could trust your laptop to not boot a trojaned kernel?
  - This is what Secure Boot is supposed to help with
Enter UEFI

• UEFI is the *Unified Extensible Firmware Interface*
  - Based on an older standard called EFI
  - This will replace legacy BIOS

• Secure Boot
  - The purpose is to *prevent execution* of Malware OS
  - This is just one aspect of UEFI
  - Specified in version 2.3.1c of the standard

• This is very different from Trusted Computing
  - Which was about attestation of integrity
How Secure Boot Changes the Picture

Signature Verification with Trusted Key(s)
Trust Model

- Secure Boot comes with several databases of public keys embedded in the firmware
  - Anything the UEFI firmware will load *must* be signed by one of these keys
    - the signature is supposed to express a “trust” relationship
    - this includes the OS loader, UEFI modules, firmware updates
  - Anything without a valid signature is not trusted, and hence is never loaded

- There is also a database containing hashes of known malware code
  - This is *not* a virus scanner
Secure Boot Key Hierarchy

- Platform key owned by IHV
- Signs one or more Key Endorsement Keys (such as The Big Microsoft Key)
- Two key databases included in BIOS
- The user is not be allowed to add arbitrary keys. Firmware may allow adding keys signed by KEK
- Hardware vendor populates key databases
Physical Presence

• The specification allows a firmware to disable all these checks *if the user is physically present*
  – Which means it is permitted to offer a setup option to disable Secure Boot completely

• Which also means Secure Boot doesn't protect you from attackers with physical access to the machine
Opting Out?

- Most UEFI implementations on x86 offer an option to boot with “Secure Boot” disabled
- UEFI defines something called “Setup Mode” that allows you to set the PK and KEK
  - This is essentially the state before it leaves the factory
  - Would be a nice way for end users to customize which keys they trust
  - But we don't see many firmwares supporting Setup Mode
Summary

- Secure Boot effectively prevents execution of malware before the Operating System is loaded
  - An attacker with root privilege can still install a trojaned libc, bash, etc
  - Obviously, not a panacea

- Certainly painful from an Open Source perspective
What Does It Mean For Linux?
Is This For Real?

• Windows 8 hardware certification mandates support for Secure Boot
  - All desktop hardware now comes with UEFI Secure Boot enabled by default

• In the server market, we're seeing a slower rate of adoption
  - Not many operating system deployed in today's data centers knows how to deal with Secure Boot - yet
  - Some new server hardware already comes with UEFI, but has Secure Boot switched off by default
OS Vendors Have To Deal With It

- Machines with Secure Boot enabled will not boot a Linux kernel unless signed with a “trusted key”
- Linux is Doomed! Film at Eleven!
  - ... not quite yet :-)
- Still, it's an interesting mix of challenges
  - Technical
  - Legal
  - Political
Kernel Community

• If you're a kernel developer, you need to be able to build and run your own kernels
  - Which Secure Boot was designed to prevent
  - One solution is to disable secure boot

• Microsoft UEFI certification is no option for end users

• Needs a technical solution
GPLv3 and Secure Boot

• Our boot loader of choice (grub2) is covered by GPLv3. grub1 and elilo are “GPLv2 or later”

• Preloads + GPLv3: we must allow customers to install an alternative boot loader
  – Legal issues around the “Tivo Clause” in GPLv3

• The Windows Logo Programme Amendment for UEFI boot loader signing says you cannot sign GPL code
Political Issues

• “Evil Empire” rhetorics and conspiracy theories

• We are walking a tightrope in terms of community acceptance
  - Secure Boot is perceived as taking away the end-user's freedom to do with his Hardware whatever he wants
  - FSF calls it “Restricted Boot” rather than “Secure Boot” for this reason

• We cannot announce a solution that leaves the community out in the cold.
Embracing Secure Boot
Minimum Technical Prerequisites

• Boot loader must be signed with a key trusted by the firmware

• Kernel must be signed

• Boot loader must verify the kernel is signed with a trusted key
  - *This* key could be in the firmware's db, or embedded in the boot loader
Getting the Foot In the Door

• As an operating system vendor, we have two ways of getting our boot loader accepted by Secure Boot enabled firmwares
Option A: Be Microsoft

• Work with all IHVs on the planet to get our public key into the firmware database
  - That's a serious scalability issue
Option B: Work With Microsoft

- Use the Microsoft Windows Logo Programme to certify your UEFI loader
  - Microsoft will sign the code with their key

- What this implies:
  - We continue to boot on all Windows certified HW out there
  - You have to sign some serious legal stuff
  - They're asking *lots* of detailed questions on the code we submit
Secure Boot: SUSE® Solution
The MOK: Machine Owner Key

• Return key management back to the user and sysadmin
  - Unconditionally, regardless of what options are missing in the firmware
  - Uniformly, with the same interface and same key formats on all systems
  - Proof of trust by physical presence
  - Implemented by Matthew Garrett, Peter Jones (RH, shim), and Gary Lin (SUSE, MOK)
Secure Boot With MOK

AT: Authenticated
BS: Boot Service

UEFI → load → shim → load → grub2 → load → kernel

AT Var → db
BS Var → MOKList
User → db
User → kernel
Multi-boot With MOK

- **shim**
  - SUSE UEFI key
  - BS Var
  - MOKList
    - User

- **grub2**
  - SUSE
  - User
  - grub2

- **kernel**
  - SUSE
  - User

SUSE boot path
custom boot path

load

load
Enrolling a MOK key

1. request

2. detect

3. enrol

4. enroll

5. reboot

Linux

mokutil --import

Password Hash

User

Reboot

UEFI

shim

MokManager

reboot custom boot path

MOKNew

RT Var

BS Var

MOKList

Password Hash

User

User

User
What Should Be Signed?

• UEFI mandates only EFI binaries to be signed
  – that protects against bootloader attacks only
  – not much value for a lot of hassle
• The value is in a kernel that can be trusted to be intact
  – hence kernel signing
  – modules need to be signed, too
  – and some functionality disabled: direct /dev/mem writes, iopl()
Module Verification With MOK

shim -> Kernel

MOKList

BS Var

MOKListRT

RT Var

copy -> Kernel

Kernel -> verify

driver 1 sig

driver 2 sig

driver 3 sig

...

driver n sig
MOK and Modules

• MOK allows for easy installation of 3rd party modules
• An addition to the MOKList is requested upon installation of a module package
• The user then approves the addition on a subsequent reboot
  - yes, a reboot is needed to prove intent and physical presence of the machine owner
  - this is somewhat less convenient than using KEKs and AVs, but, and that's most important, works in all cases, even when you can't install your own KEK in the firmware
Restrictions in Secure Boot Mode

• No direct access to IO port, must use kernel interface
  – KMS drivers are required for graphics card

• No direct access to memory
  – No /dev/mem, no /dev/rmem

• Not possible to load unsigned 3\textsuperscript{rd} party modules
Further Limitations

- Hibernation
  - aka S4 or Suspend-to-disk
  - stores RAM image on disk
  - image can be manipulated, or even created by malware
  - it's a hole in the security model
  - solution: HMAC Signatures
  - implementation by Joey Lee (SUSE) in progress

- kexec and kdump
  - allow for executing an entirely new OS
  - must do signature verification
  - in-kernel implementation by Vivek Goyal (Red Hat) in progress
Secure Boot in SUSE Linux Enterprise
Current State of Support

• Enabled in
  - SUSE Linux Enterprise 11 SP3
  - SUSE Linux Enterprise 12
  - openSUSE 12.3 and forward

• SLE and openSUSE use different keys and hence different shim loaders
Installation on a Secure Boot System

- DVD image should boot nicely with Secure Boot enabled
- Copy your ISO image to USB stick if your firmware is funky
  - isohybrid –uefi <ISO-image>
- Secure Boot support will be automatically enabled by installer.
- Your boot manager will have an entry named
- “SUSE Linux Enterprise 11 SP3”
Enrolling MOK

- Needed to allow 3rd party module (or kernel) to be accepted by shim / Secure Boot
- Use "mokutil --import <key.der>" then reboot
- You'll need to type a password for MOK List (or use –root-pw to use root password. Installer will do that automatically if 3rd party module are enrolled at install time)
- Reboot and accept changes in shim
Summary

• Secure Boot lets you lock down your systems more tightly
  - trade offs in convenience, limitations like kexec/kdump and hibernation

• MOK handling allows flexibility for testing, upgrading and 3rd party support

• SUSE integration for lower effort handling
Unpublished Work of SUSE. All Rights Reserved.
This work is an unpublished work and contains confidential, proprietary and trade secret information of SUSE.
Access to this work is restricted to SUSE employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of SUSE.
Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer
This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.