



Integrating UEFI into Relax-and-Recover

by

Gratien D'haese

gratien.dhaese@it3.be

LOAD 2013 - Antwerpen, Belgium



Table of Content

- Who am I?
- Unified Extensible Firmware Interface (UEFI)
- Relax-and-Recover (rear)
- Hacking rear for UEFI integration



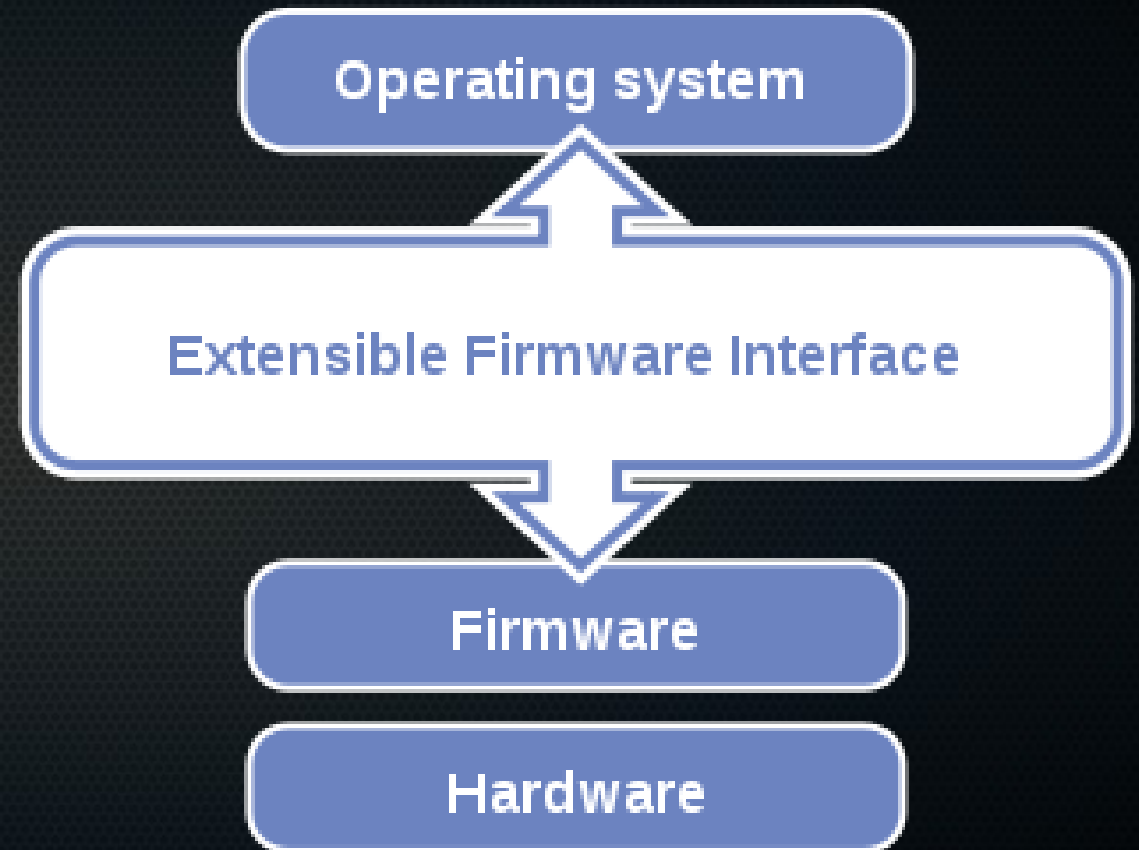
Who am I?

- Independent UNIX Consultant
- Over 25 years of experience with UNIX (using Linux since Dec 1991 version 0.1)
- Open source projects involved:
 - Relax-and-Recover
 - Make CD-ROM Recovery (development on hold)
 - WBEMextras (towards HP-UX HPSIM clients)
 - Ad-hoc Copy and Run (adhocr)
 - Config 2 HTML (cfg2html)



Unified Extensible Firmware Interface

- Unified Extensible Firmware Interface (UEFI) is meant as a replacement for the Basic Input/Output System (BIOS) firmware interface
- Initially (1998) designed by Intel for Itanium processor
- Since 2005 managed by the Unified EFI Forum (uefi.org)





Why UEFI?

- BIOS has its (aging) limitations
 - 16-bit processes,
 - max. 2.2TB,
 - 1 MB memory addressing,
 - 4 primary partition MBR
- UEFI offers 32-bit and 64-bit mode,
 - goes beyond the 2.2TB limit,
 - uses Globally Unique IDs (GUID) partition tables (GPT),
 - network authentication,
 - cryptography

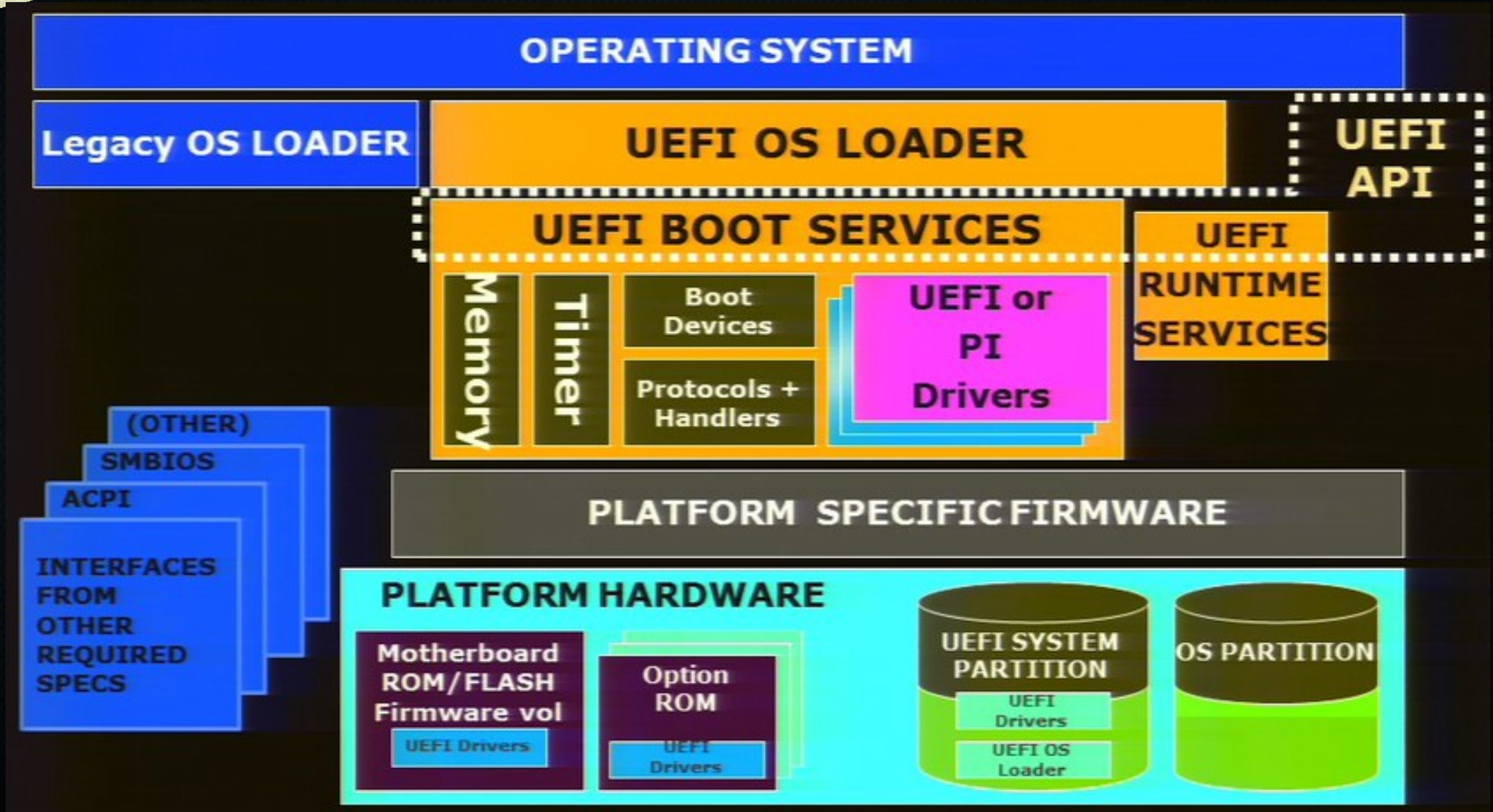


UEFI Pro's and Cons

- Pro's of UEFI:
 - Flexible and modular
 - Multiple OS loaders possible (no need to Chainload)
 - Written in C language (and it is free [BSD license])
 - IPv4 and IPv6 support
- Con's of UEFI:
 - Error prone
 - Lots of code



UEFI Architecture





UEFI Services

- Two type of services:
 - Boot Services: interaction with the firmware of the motherboard
 - Console (text, graphical), block devices and other devices
 - Image loading (drivers, applications and OS loaders)
 - Runtime Services: start after the “boot services” and keep on running (while the OS is available)
 - Timer, date protocol
 - NVRAM access
 - Wakeup alarm
 - System reset



GUID Partition Table

- GPT (or GUID Partition Table) is part of UEFI specification
- Maximum 128 partitions per disk
- Maximum partition size is 9.4 ZB (assuming 512 byte blocks)
- Each partition has an GUID
- Using a primary and a backup table for redundancy



UEFI System Partition (ESP)

- Use GPT for UEFI boot (so use parted or gdisk)
- Partition size => 512 MB
- Partition type = VFAT
 - parted, or gdisk, FAT32 filesystem, boot flag enabled (under GPT this mean ESP!)
 - fdisk, partition type 0xef, format it as FAT32



UEFI Device Path

- Binary description of location:

blk3 :HardDisk - Alias (null)

PciRoot(0x0)/Pci(0x1,0x1)/Ata(Primary,Master,0x0)/HD(2,GPT,96D3A6AD-BB6F-48DB-8B43-050EA098AA98,0x100,0x31800)

blk4 :BlockDevice - Alias (null)

PciRoot(0x0)/Pci(0x1,0x1)/Ata(Primary,Master,0x0)

blk5 :BlockDevice - Alias (null)

PciRoot(0x0)/Pci(0x1,0x1)/Ata(Secondary,Master,0x0)

Shell> _

Initialize PCI root
Bridge

Initialize PCI
Device

Initialize ATA
Device

Initialize the Partition
Driver



UEFI Boot Manager

Boot Manager

Boot Option Menu

EFI DVD/CDROM

EFI DVD/CDROM 1

0

1

2

EFI Internal Shell

ubuntu 12.10

↑ and ↓ to change option, ENTER to select an option,
ESC to exit

Device Path :

PciRoot(0x0)/Pci(0xD,0x0)
/Ata(Primary,Master,0x0)/
HD(1,GPT,1CCBAED0-9D38-45
3F-8CAC-BA85B1BD1275,0x80
0,0x5F000)/\EFI\ubuntu\gr
ubx64.efi

OS Boot
Loader

↑↓=Move Highlight

<Enter>=Select Entry

Esc=Exit without Save



Running any kind of UEFI program

```
01/28/13  08:34p <DIR>          4,096  .
01/28/13  08:34p <DIR>           0  ..
01/28/13  08:34p <DIR>          4,096  Hello
          0 File(s)              0 bytes
          3 Dir(s)
```

```
fs0:\EFI> cd Hello
```

```
fs0:\EFI\Hello> ls
```

```
Directory of: fs0:\EFI\Hello
```

```
01/28/13  08:34p <DIR>          4,096  .
01/28/13  08:34p <DIR>          4,096  ..
01/28/13  08:34p          19,232  HelloWorld.efi
          1 File(s)        19,232 bytes
          2 Dir(s)
```

```
fs0:\EFI\Hello> HelloWorld.efi
```

```
UEFI Hello World!
```

```
fs0:\EFI\Hello> _
```

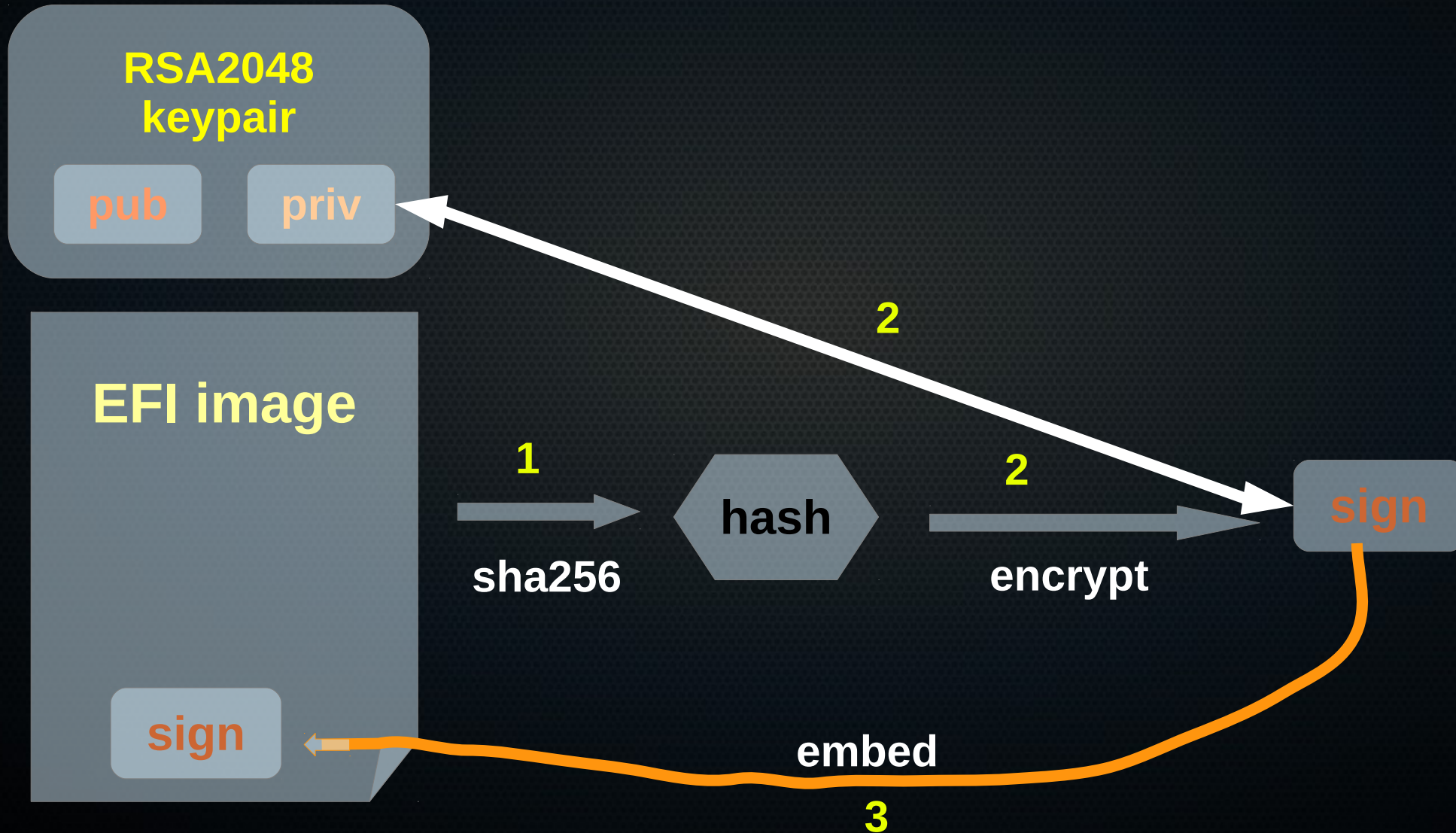


UEFI Secure boot

- Securing the bootstrapping process means establishing a chain of trust
- The “platform” is the root of this chain of trust (=Platform Key)
- Each OS bootloader, kernel has there own set of Key Exchange keys (KEKs)
- A computer is either in “setup” mode (secure boot disabled) or “user” mode (secured)
- Be aware the PK and KEKs are locked into the firmware (eprom) and are not transportable (cloning with secure boot enabled is not possible)

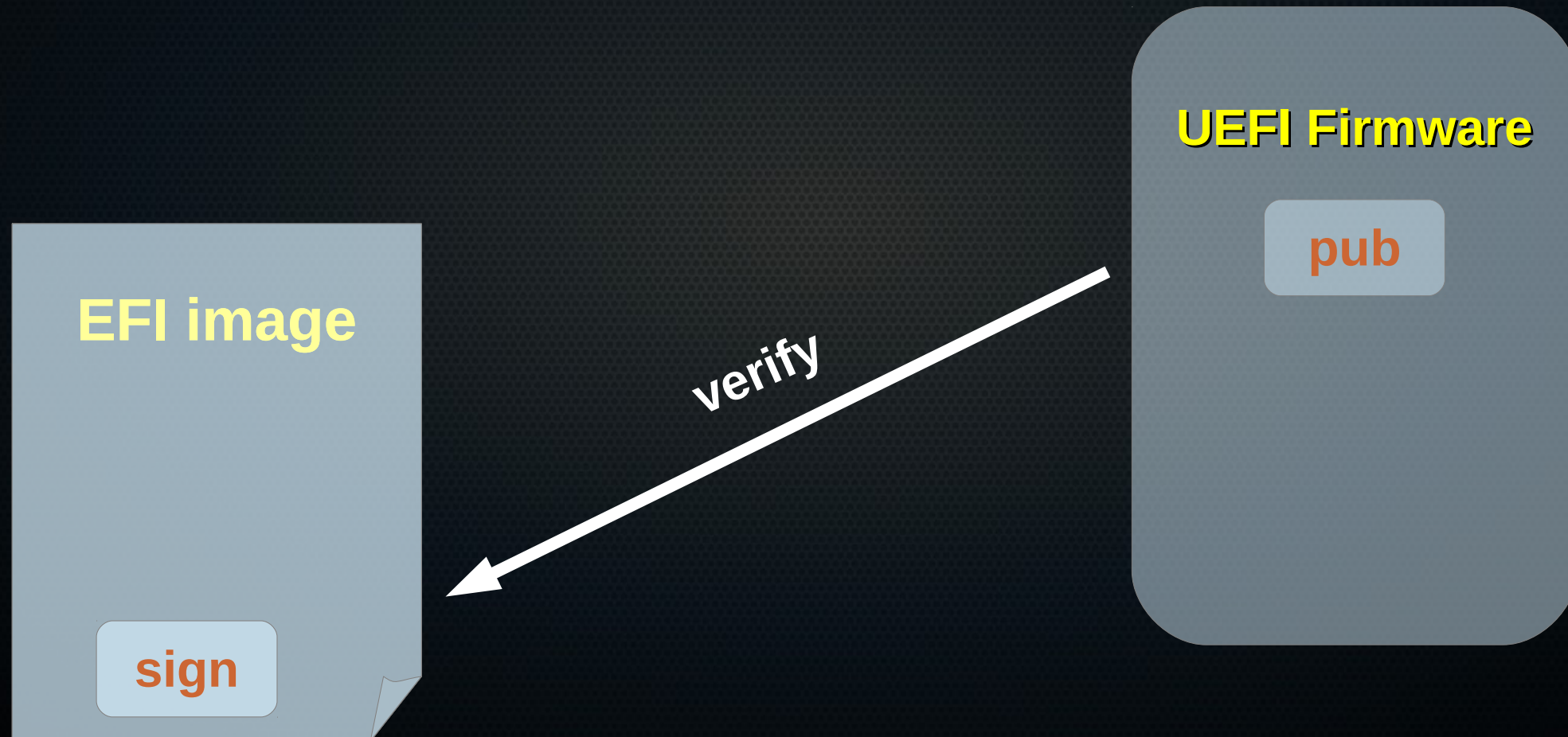


How to sign an EFI image?



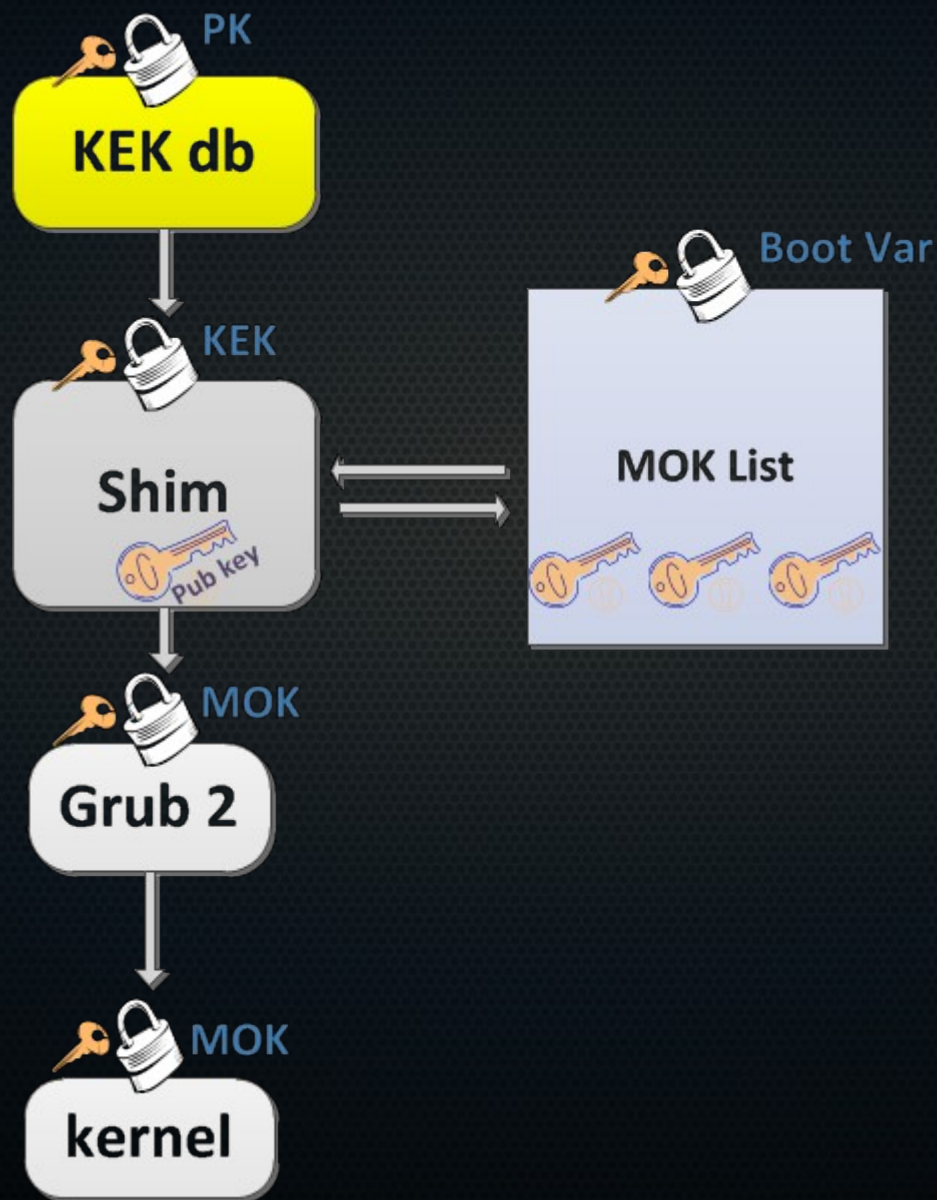


Secure booting of an EFI image





Secure boot & Linux





Dealing with Secure Boot

- Three options (disable it, pre-signed boot loader, or use your own keys)
- Pre-signed boot loader: shim
 - Secure boot keys (registered PKs)
 - Shim keys (not registered with firmware)
 - Machine Owner Keys (MOKs) to sign your own EFI's
- Create your own MOKs is easy:
 - `openssl req -new -x509 -newkey rsa:2048 -keyout MOK.key -out MOK.crt -nodes -days 3650 -subj "/CN=gratien/"`
 - `openssl x509 -in MOK.crt -out MOK.cer -outform DER`



Compile your own EFI executable

- Install some pre-requisites: gnu-efi, efibootmgr, openssl-devel, binutils-devel, libuuid-devel, iasl, git
- `$ git clone git://github.com/tianocore/edk2.git`
- `$ cd edk2`
- `$. edksetup.sh`
- `$ vi Conf/target.txt`
- `$ build`
- See <http://sourceforge.net/apps/mediawiki/tianocore>



Relax-and-Recover (rear)

- Bare Metal Disaster Recovery solution for Linux
- Supported on practical all Linux distributions (ia32, x64, ia64, ppc). Written in “bash”
- Easy to deploy (set it up once, test it and forget about it)
- Recovery with rear takes care of everything (except what you excluded in the 'mkbackup' process)
- Rear is very modular and can easily be extended with a few simple scripts



Rear under the hood

- Rear builds a rescue image from the “live” system
 - A bootable image (ISO, PXE, OBDR, USB)
 - An archive of the current data (NFS, CIFS, USB, TAPE, RSYNC)
 - Or, without an archive if a (commercial) external backup program is used which takes care of full backup
- Restored system is an accurate copy
- Cloning is possible (within limits): P2P, P2V, V2P and V2V
- Recovery process is fully automated



Rear Challenges

- Different system storage and network configuration
 - Rear has built-in disk migration assistance
 - At boot time you can define a new IP address or use DHCP instead instead of original (saved) IP address
- Incompatibilities between firmware/software/hardware
 - Rear has most kernel modules automatically added
 - Rear understand most popular Linux distributions
 - Rear works together with most popular commercial external backups programs (TSM, NBU, DP, Bacula)
- Developers and documentation



Integration of UEFI into rear (ia64)

- Integrity platform (ia64) UEFI support was added long time ago
- Using the UEFI standard v1 or v2 (**no secure boot**)
- What do we need to integrate?
 - **/boot/efi** : mounted as vfat
 - **/boot/efi/efi/*/elilo.efi** : boot loader (same for different flavors of Linux)
 - **CONSOLE="console=tty1 console=ttyS1"** : mandatory
 - No need to be grubby after recovery as /boot/ef/* is all you need
 - Create a bootable CDROM which is recognized by UEFI



Integration of UEFI into rear (x86_64)

- What do we need for UEFI support on Linux?
 - Bootable disk with GPT partition table (parted /dev/disk p)
 - /boot/efi mount point (vfat)
 - Linux Kernel Config should contain CONFIG_EFI=y
 - UEFI Runtime Variables/Services Support - 'efivars' kernel module
 - Check /sys/firmware/efi/vars/ directory
 - Efibootmgr to manipulate boot entries, order of booting
 - Create a bootable UEFI capable ISO image




Start from the sources

- `$ git clone git@github.com:rear/rear.git`
- `# yum|zypper install rpm-build lsb mingetty`
- `$ make rpm`
- `$ sudo rpm -ivh rear-1.14-1.git201303211657.noarch.rpm`
- `$ sudo su`
- Rear is at your service (`/etc/rear/local.conf`, `/usr/share/rear/*`)
- Edit `/etc/rear/local.conf` (`BACKUP=NETFS,OUTPUT=ISO`)



Preparation work

- To manipulate disk devices with GPT label we need
 - Be sure this system uses UEFI
 - Parted (./conf/Linux-i386.conf:parted) 
 - Gdisk (GPT fdisk utility – not mandatory, but nice to have)
 - A mounted /boot/efi file system (type vfat)
 - The efivars kernel module
 - Efibootmgr utility
 - Which boot manager is used (grub, elilo, gummiboot, shim,...)
 - Secure boot used? Recovered system might be unbootable!



Writing your own rear scripts

- Good to know – everything is a script, even config files
- Does rear has an API? Yes, check out our functions:
`grep '()' /usr/share/rear/lib/*functions.sh`
- Rear works with workflows – see other presentations on the basics
- Where to drop your script? Use 'rear -s mkbackup' to see all existing scripts and order of execution



Workflow dependent (-s option)

- mkrescue – mkbackup
 - prep
 - layout/save
 - rescue
 - build
 - pack
 - output
 - backup
 - recover
 - prep
 - layout/prepare & recreate
 - restore
 - finalize
 - wrapup
- See 'rear/doc/user-guide/' for more details**



What to do in preparation phase?

- In the 'prep' phase we do basic checks and copy the binaries
 - In case of UEFI: `prep/default/31_include_uefi_tools.sh`
 - `modprobe -q efivars || return`
 - Define `SYSFS_DIR_EFI_VARS=`
 - `mount | grep -q efivarfs && SYSFS_DIR_EFI_VARS=/sys/firmware/efi/efivars`
 - `[[! -d /boot/efi]] && return`
 - `USING_UEFI_BOOTLOADER=1`
 - `PROGS=("${PROGS[@]}" parted gdisk efibootmgr uefivars dosfsck dosfstools)`
 - `MODULES=("${MODULES[@]}" efivars)`



What to do in layout phase?

- Save layout: should already be in place
- Needed to fix a “space” bug in 'part' line (see issue #212)
 - Parted: 3 44.6GB 44.8GB 210MB fat16 EFI System Partition boot
- See </var/lib/rear/layout/disklayout.conf> :
 - `part /dev/sda 209715200 44557139968`
`EFI0x20System0x20Partition boot /dev/sda3`
 - `fs /dev/sda3 /boot/efi vfat uuid= label=FEDORAEFI`
`options=rw,relatime,fmask=0077,dmask=0077,codepage=437,iocchar`
`set=ascii,shortname=winnt,errors=remount-ro`



What to do in rescue phase?

- Script `rescue/default/85_save_sysfs_uefi_vars.sh` saves the UEFI variables
- Foresee an escape: `(($USING_UEFI_BOOTLOADER)) || return`
- Capture output : `efibootmgr > $EFIBOOTMGR_OUTPUT`
- Save the current set of UEFI variables in file `$VAR_DIR/recovery/uefi-variables`
- Find and define the current UEFI_BOOTLOADER
- Save UEFI_BOOTLOADER to `rescue.conf` file (used in recover mode)

```
# cat $ROOT_FS/etc/rear/rescue.conf
USE_DHCLIENT=y
DHCLIENT_BIN=dhclient
DHCLIENT6_BIN=
NETFS_KEEP_OLD_BACKUP_COPY=""
NETFS_PREFIX="freedom"
USING_UEFI_BOOTLOADER=1
UEFI_BOOTLOADER="/boot/efi/EFI/fedora/shim.efi"
```




How to test your scripts

- Testing is a big challenge (without having to run a complete mkrescue or mkbackup)...

```
SYSFS_DIR_EFI_VARS="/sys/firmware/efi/efivars"
```

```
./usr/share/rear/lib/uefi-functions.sh
```

```
USING_UEFI_BOOTLOADER=1
```

```
UEFI_BOOTLOADER=
```

```
LogPrint="echo"
```

```
BugError="echo"
```

```
TMP_DIR=/tmp
```

```
VAR_DIR=/var/lib/rear
```

```
#--above lines should be removed once script is finished--#
```

```
# a simplified efivars replacement
```

```
(( $USING_UEFI_BOOTLOADER )) || return
```



What to do in the output phase?

- `OUTPUT=ISO` => `output/ISO/Linux-i386/*`
- So far only isolinux was the only CD-ROM/ISO boot-able way
- Should become dual boot-able (isolinux and UEFI)
- Need to prepare an efiboot image
- Need to populate an ISO filesystem that separates UEFI and isolinux
- Write a dual boot ISO image with mkisofs using the ISO filesystem (dropped the `${ISO_FILES[@]}` array)



Recover time – finalize phase

- Plain old BIOS systems use grub, grub2, ...
- UEFI systems have a modified grub[2]-efi, but depend on **efibootmgr** to manipulate the EFI boot
- Required a new script to run the efibootmgr command:
 - finalize/Linux-i386/23_run_efibootmgr.sh
 - finalize/Linux-i386/22_install_grub2.sh will not run if UEFI is used



ISO booting on UEFI system

- Booting the ISO image:

```
GNU GRUB version 2.00

Relax and Recover (no Secure Boot)
Relax and Recover (Secure Boot)
Reboot
Exit to EFI Shell

Use the ▲ and ▼ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the commands
before booting or 'c' for a command-line. ESC to return
```



demo

