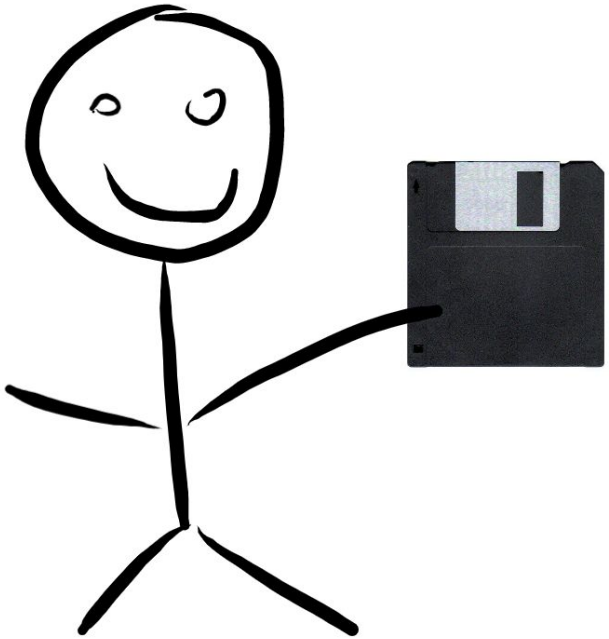
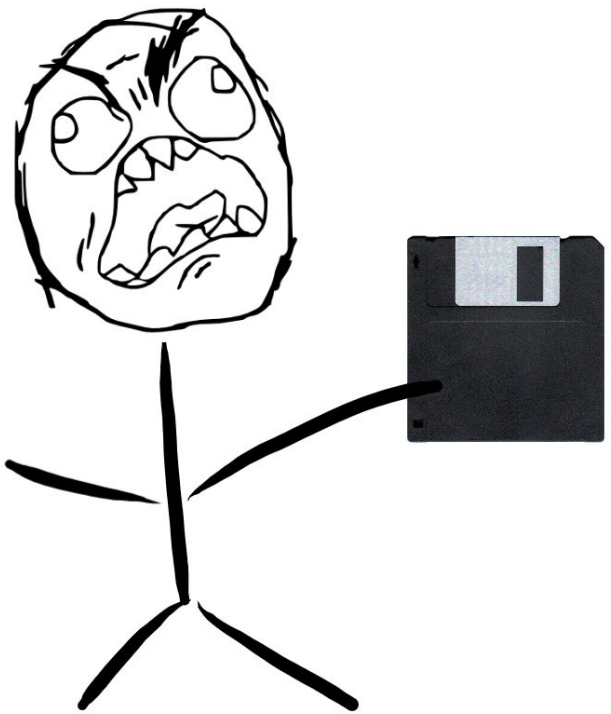
Hardening an Application-specific Linux

tycho@tycho.ws, tycander@cisco.com
github.com/tych0


Red Hat


ubuntu





How big of a problem is this?

- CVE-2019-14284: DoS, Kernel divide by zero
- CVE-2019-14283: infoleak, Out of bounds read
- CVE-2018-7755: KASLR leak

Solution

“Deleted code is debugged code” -- Jeff Sickel

s/CONFIG_BLK_DEV_FD/# CONFIG_BLK_DEV_FD is not set/

There are lots of these

CONFIG_SND
CONFIG_SOUND
CONFIG_WIRELESS
CONFIG_WLAN
CONFIG_STAGING
CONFIG_MACINTOSH_DRIVERS
CONFIG_BT
CONFIG_BLK_DEV_FD
CONFIG_YENTA
CONFIG_PCMCIA
CONFIG_CAN_DEV
CONFIG_CAN_VCAN
CONFIG_WIMAX

CONFIG_RFKILL
CONFIG_WAN
CONFIG_ISDN
CONFIG_*_LAPTOP
CONFIG_CIFS
CONFIG_(^ext4|xfs)_FS
CONFIG_DRM_NOUVEAU
CONFIG_DRM_RADEON
CONFIG_SUSPEND
CONFIG_HIBERNATE
CONFIG_IP_DCCP
CONFIG_IP_SCTP (might need for NFS, etc.)
CONFIG_FB_(^CMDLINE|VESA|EFI)

And lots of these

CONFIG_NET_VENDOR_(^CISCO|INTEL)
CONFIG_SCSI_(^MEGARAID)
CONFIG_LEDS*
CONFIG_MMC
CONFIG_USB* (modems, printers, etc.)
CONFIG_INPUT_* (IR remotes, etc.)
CONFIG_RC_CORE (more IR remotes)
CONFIG_MEMSTICK
CONFIG_BATTERY_*
CONFIG_CHARGER_*
CONFIG_CYCLADES
CONFIG_TYPHOON
CONFIG_X86_PLATFORM_DEVICES
CONFIG_X86_EXTENDED_PLATFORM
CONFIG_INFINIBAND
CONFIG_CDROM_PKTDVD
CONFIG_DNS_RESOLVER
CONFIG_IEEE802154
CONFIG_ATALK
CONFIG_MTD
CONFIG_PARPORT
CONFIG_SFI
CONFIG_ZONE_DMA
CONFIG_HID_* (minus whatever you need)
CONFIG_DRM
CONFIG_AGP

...and lots of these

CONFIG_SLIP
CONFIG_EEPROM_*
CONFIG_IPX
CONFIG_JME
CONFIG_NETCONSOLE
CONFIG_NETPOLL
CONFIG_AUXDISPLAY
CONFIG_UWB
CONFIG_SSB
CONFIG_B44
CONFIG_BCMA
CONFIG_KEYBOARD_*
CONFIG_MEDIA_SUPPORT

CONFIG_VORTEX
CONFIG_FIREWIRE
CONFIG_SENSORS*
CONFIG_HP_ILO (DELL_RBU, etc.)
CONFIG_I8K
CONFIG_SUNDANCE
CONFIG_TYPHOON
CONFIG_DCB
CONFIG_PHONET
CONFIG_ATALK
CONFIG_ATA_OVER_ETH
CONFIG_NET_DROP_MONITOR
CONFIG_ATA_SFF

Kernel config checking script

- Suggests particularly vulnerable things to disable
- Other options “tighten” things
- https://github.com/a13xp0p0v/kconfig-hardened-check
- CONFIG_STATIC_USERMODE_HELPER

Detour: The kernel asks userspace for stuff

- Hotplug events
- poweroff/reboot
- Core dumps
- Cgroup v1 has “notify_on_release”
- Module auto-loading
- https://github.com/tych0/huldufolk/blob/master/sample-usermode-helper.toml has a complete list

How does it ask userspace?

```
socket(PF_ALG, SOCK_SEQPACKET, 0); // can be unprivileged
```

net/ is missing the AF_ALG=38 protocol family, so
it does a
```
request_module("net-pf-%d", family);
```

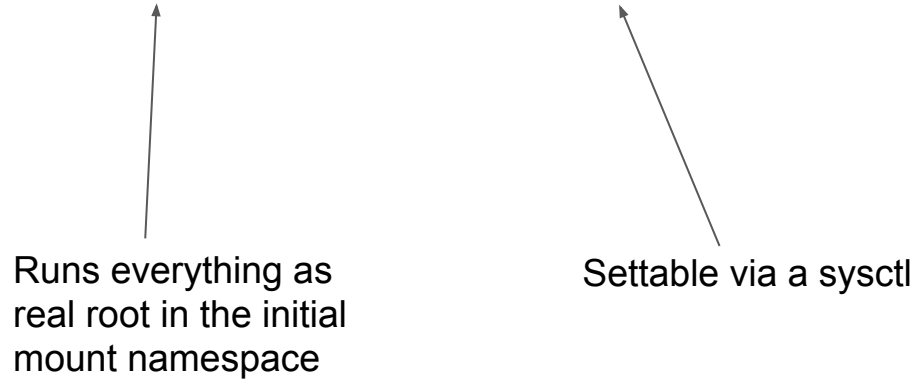
```
request_module() -> call_modprobe() ->
call_usermode_helper(modprobe_path, ...)
```

```
/sbin/modprobe -q -- net-pf-38
```

modprobe looks in modules.alias and finds:
alias net-pf-38 af_alg
and inserts af_alg.ko

Attack 1

```
call_usermode_helper(modprobe_path, ...)
```



Runs everything as real root in the initial mount namespace

Settable via a sysctl

=> A binary that functions like a setuid cat can be used to run arbitrary code.

Attack 2

- Use `call_usermode_helper()` directly from shellcode
- https://googleprojectzero.blogspot.com/2018/09/ chooses to use `call_usermode_helper()` instead of changing memory protections
- Serves as an additional mechanism for an exploit to hand flow control back to userspace as in https://www.openwall.com/lists/oss-security/2017/02/04/1 or https://www.openwall.com/lists/oss-security/2016/12/07/3

The solution

- CONFIG_STATIC_USERMODE_HELPER=y
- Proxy all call_usermode_helper() requests through a hard coded path in userspace (set via CONFIG_STATIC_USERMODEHELPER_PATH="/sbin/usermode-helper")
- Userspace decides what is legitimate and what is not

What goes in /sbin/usermode-helper?

- First public implementation in LinuxKit: https://github.com/linuxkit/linuxkit/blob/master/pkg/init/usermode-helper.c
  - Not general purpose (disallows most helpers)
  - Written in (simple) C
- Enter https://github.com/tych0/huldufolk
  - Written in (<200 lines of) Rust
  - Config file for specifying what to allow

How does it work?

- Kernel does:
  `execv("/sbin/usermode-helper", (char *[]){"modprobe", NULL})`
- `usermode-helper` reads a hard coded config file path e.g. `/etc/usermode-helper.conf`
- Decides whether to allow the action based on args
- Re-execs the real binary if allowed

Sample config

```
# kernel/kmod.c
# set via sysctl
[[helpers]]
path = "/sbin/modprobe"
argc = 4
capabilities = "= cap_sys_module+eip"

# kernel/reboot.c
# set via a sysctl
[[helpers]]
path = “/sbin/poweroff”
```

```
# kernel/reboot.c
# Hard coded.
[[helpers]]
path = "/sbin/reboot"
argc = 1

# lib/kobject_uevent.c
# Default set by
# CONFIG_UEVENT_HELPER_PATH,
# controllable by sysctl.
[[helpers]]
path = "/sbin/hotplug"
argc = 2
```

Threat Model

- Attacker has control of RDI and RIP, so they can do `call_usermode_helper()`
- Other situations (attacker writes to `/proc/sys/kernel/modprobe`, `/etc/usermode-helper`, or `/sbin/usermode-helper`) not considered

TODOs

- argument filters (probably based on regexes?)
- setting No New Privileges?
- Namespaces?
- seccomp filters? Is there some nice language for specifying these in config files? Perhaps we want to do something else?

Pain points

- Need to change config file when you change sysctls or add custom cgroup release scripts
- Could read sysctls for these things maybe?

Code

- https://github.com/tych0/huldufolk
- Detailed writeup in README.md about threat model, etc.
- Full config for every usermode helper call in 5.0

Protecting secrets in the TPM

Problem Statement

- Store secrets in the TPM
- Restrict access to the secrets to authorized kernels
- Work on legacy BIOS as well as UEFI based systems
- Easy to manage, handle updates gracefully

Protecting TPM Secrets

- “Seal” data to a set of PCRs
  - A specific set of PCR values are used as a key to lock/unlock TPM secrets
- TPM protects PCRs from tampering

Setting TPM PCRs in Early Boot

- Secure boot measures system state into the PCRs
  - firmware / config
  - bootloader, etc.
  - Kernel
- When the components change, the PCR values change

UEFI Secure Boot

- UEFI verifies signature of everything it executes
  - static root of trust
  - public key embedded in firmware
  - Microsoft controls master keys
- PCR 7 measures the kernel’s igning authority
  - Stable across multiple kernels with the same signer

Intel Trusted Execution Technology (TXT)

- Hardware and firmware creates a dynamic root of trust
  - “SINIT ACM”
- TXT “measured launch environment” verifies and bootstraps the kernel

Solving the problem

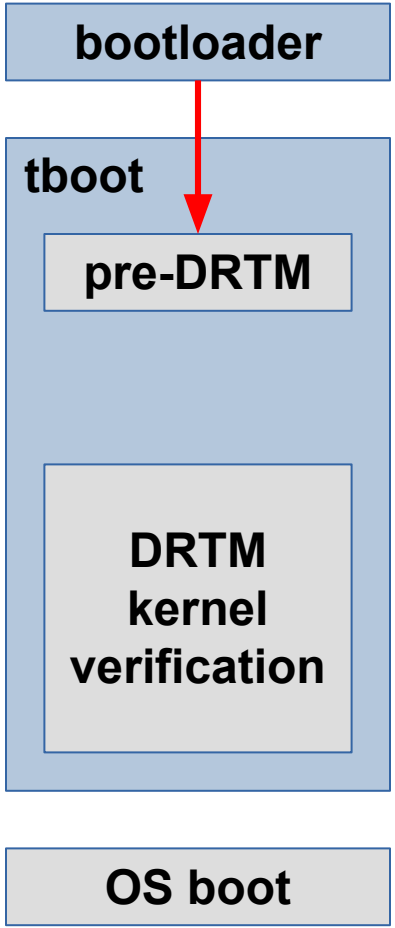
- UEFI secure boot
  - Stable PCR
  - Only works on UEFI systems
- Intel TXT
  - Unstable PCRs
  - Works on all systems with TXT (which are more than UEFI)

Lessons Learned

- Signature based PCRs are stable assuming the same signing authority
- TXT verification of the kernel/initrd/command line happens in the “tboot” bootloader

Proposed solution

- Extend tboot to support signature verification using PECOFF
  - Same format as UEFI
  - Add signing authority to the tboot policy
- No changes required to SINIT ACM required



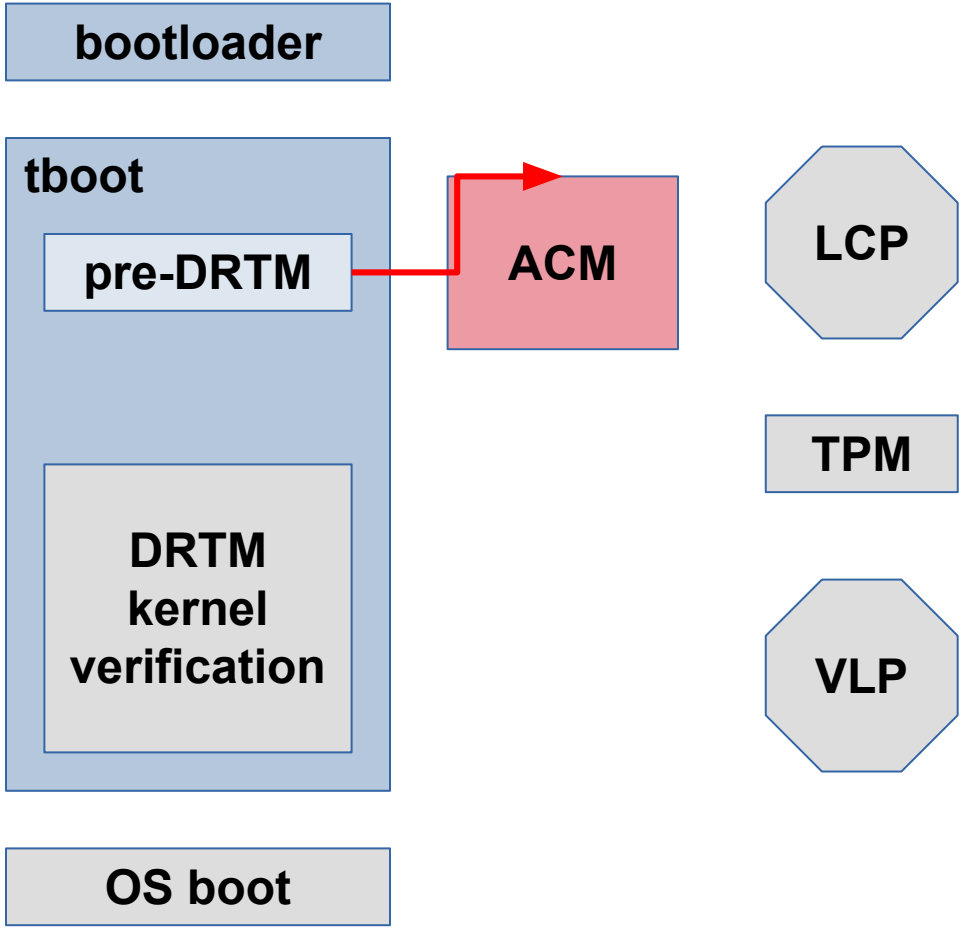
ACM

LCP

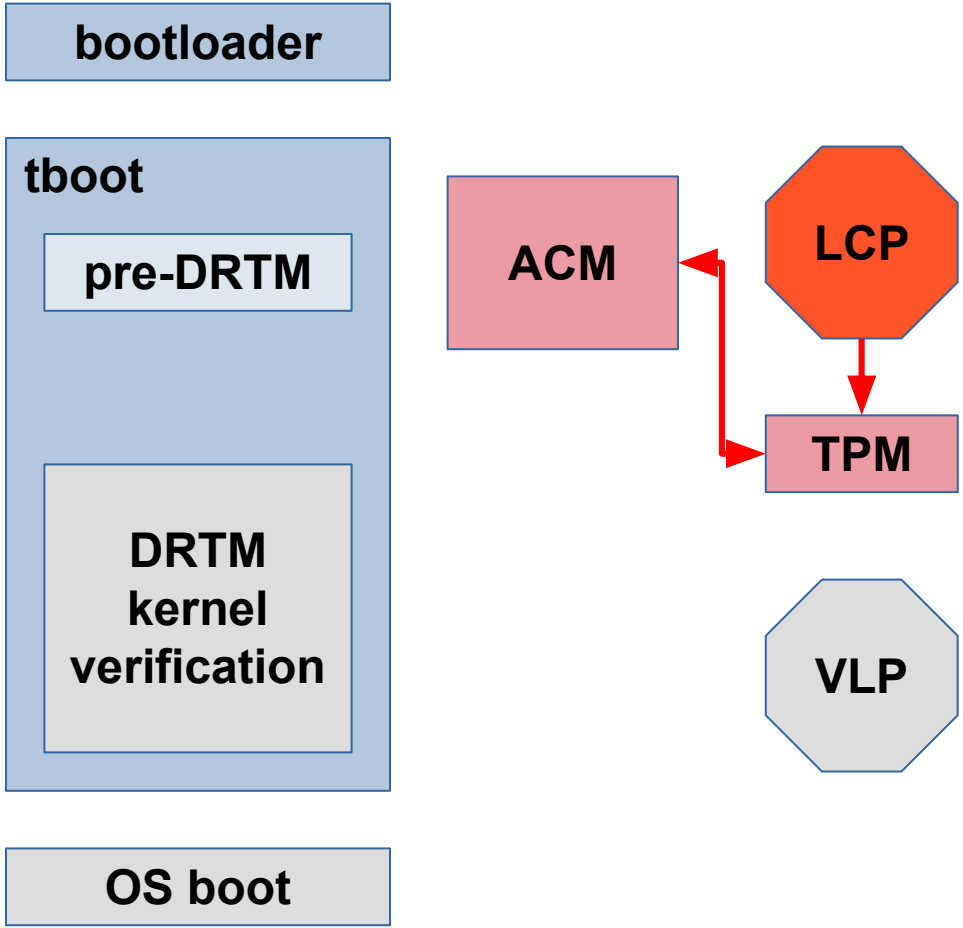
TPM

VLP

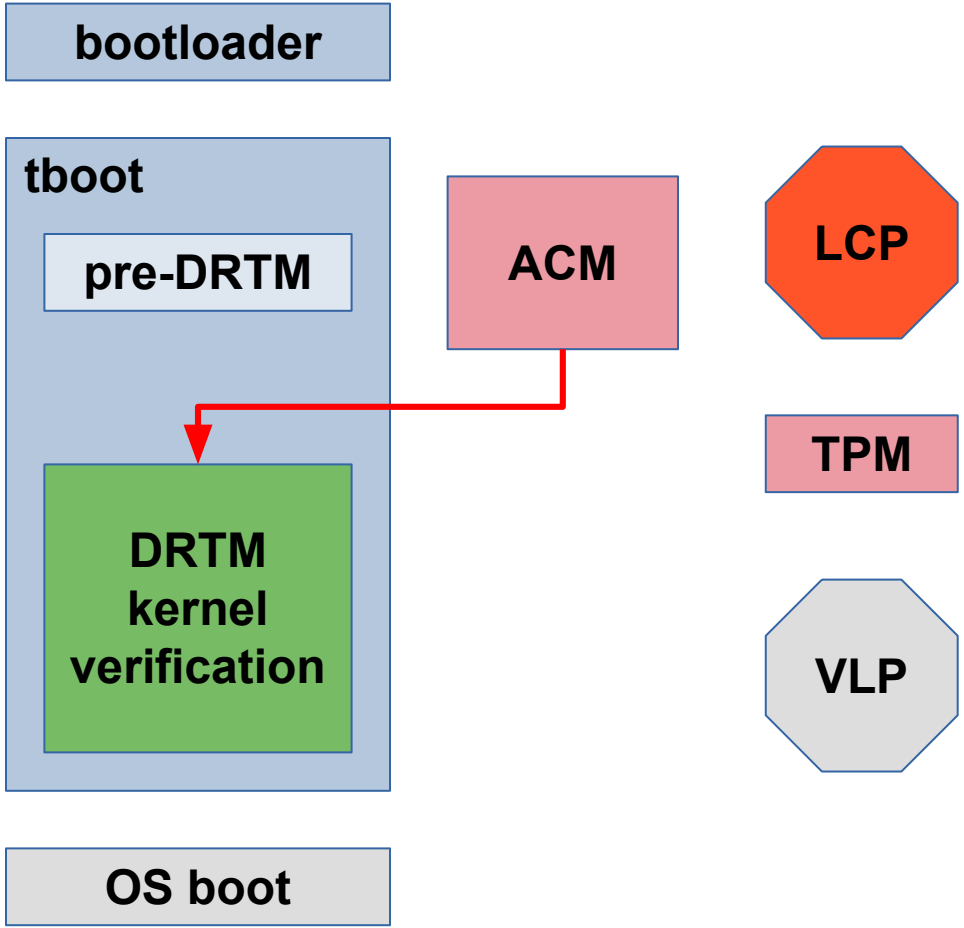
- bootloader boots tboot
- tboot performs TXT sanity checks



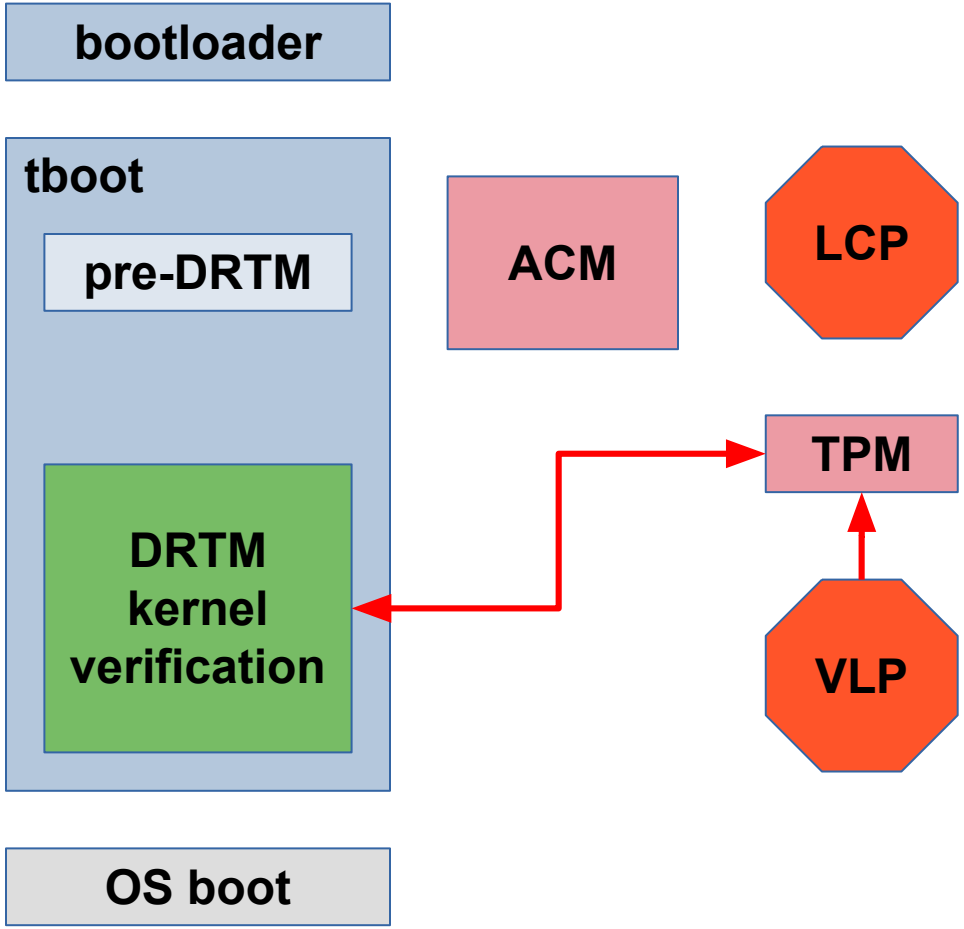
- tboot issues special CPU instruction to start TXT process
- SINIT Authenticated Code Module (ACM) establishes a dynamic root of trust



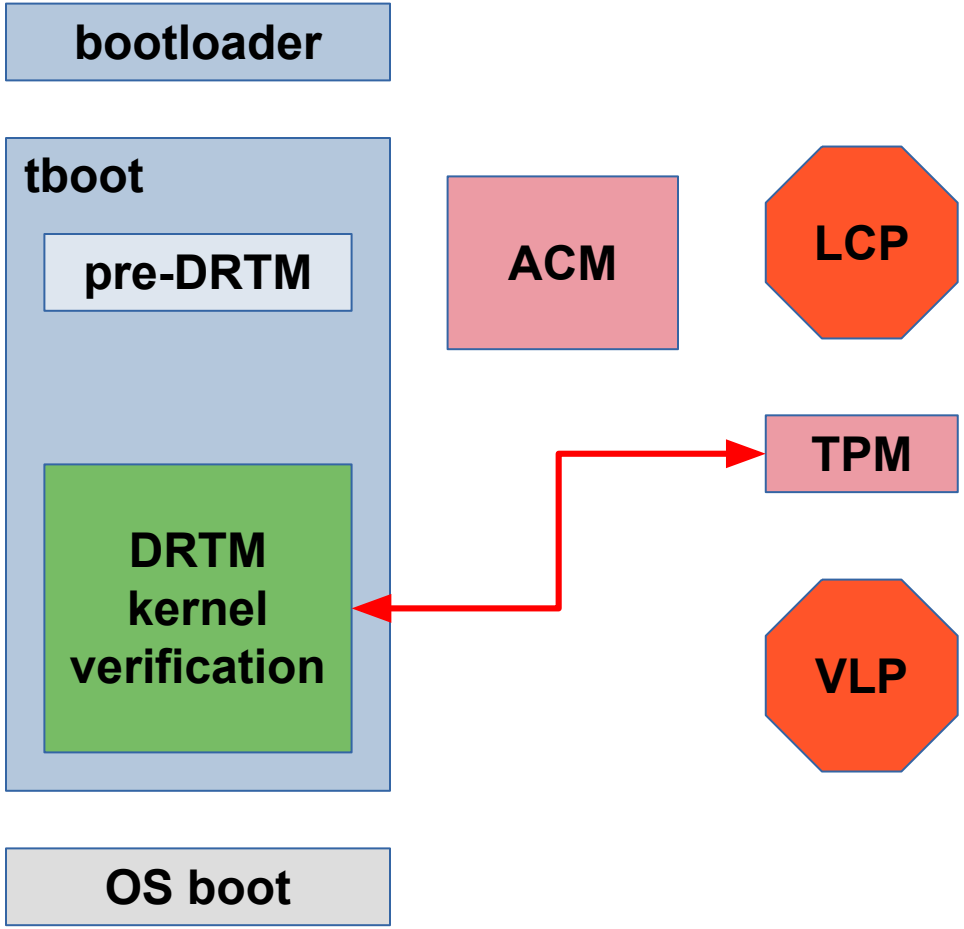
- ACM examines the Launch Control Policy (LCP) rooted in the TPM
- ACM enforces the LCP
  - validates firmware
  - validates tboot



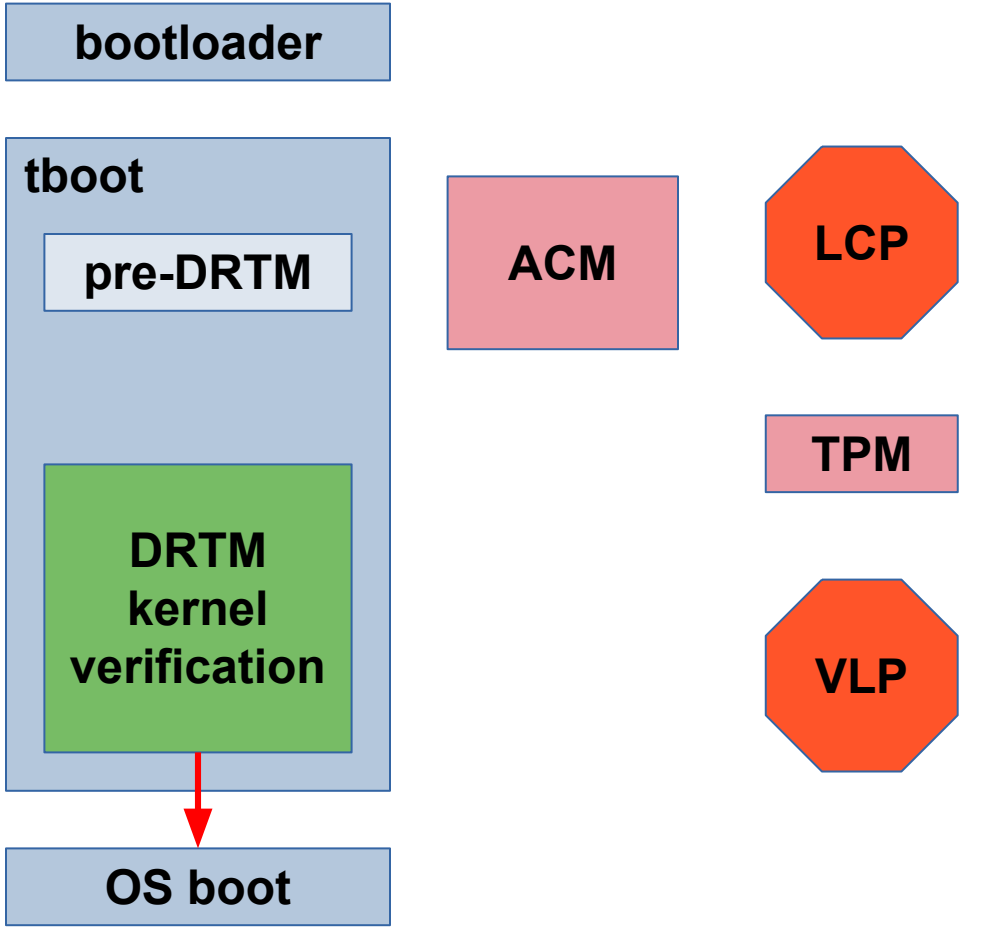
- ACM returns execution to the “measured launch environment” (tboot)
- tboot continues to execute in a protected environment



- tboot examines the Verified Launch Policy (VLP) rooted in the TPM
- tboot verifies the kernel, initrd, and cmdline
  - currently using hash values
  - adding support for



- tboot extends TPM PCRs
  - kernel signing authority certificate digest
  - kernel, initrd, and cmdline digests
- TPM TXT PCRs are protected against tampering outside the



- tboot boots the OS using the measured kernel, initrd, cmdline
- TPM rooted secrets are unlocked if the tboot PCR values match the sealing values

Open Issues

- No verification of the initrd or kernel command line
  - Problem for UEFI too
  - May be able to use UEFI workarounds
  - Existing digest verification OK

Links

- Code: https://sourceforge.net/p/tboot/mailman/tboot-devel/?style=threaded&viewmonth=201909
- tboot mailing list thread: https://github.com/pcmoore/misc-tboot
- Paul’s talk: https://www.youtube.com/watch?v=Qbjz_5jUE9o
- Paul’s slides: https://static.sched.com/hosted_files/lssna19/17/lss-securing_tpm_with_txt-pmoore-201909-r2.pdf

Спасибо

tycho@tycho.ws, tycander@cisco.com
http://github.com/tych0