Advanced Applications of DRTM with TrenchBoot SecureLaunch for Linux

Daniel P. Smith
Apertus Solutions, LLC
TrenchBoot Project

Agenda

- Introduction
- Early Launch Use Cases
- Late Launch Use Cases
- Futures
- Q&A

TrenchBoot Contributors

SecureLaunch Status

Timeline:

- The kern_info structure was merged in 5.5
- RFC version of SecureLaunch patch set submitted to LKML - March 2020
- RFC version of GRUB patch set sent to GRUB mailing list - May 2020
- SecureLaunch patch set submission to LKML - Sept 2020

See Project Info at the end for past presentations and ways to engage TrenchBoot community

Introduction

Common Understanding:

- *Launch Integrity is the responsibility a user or enterprise entrusts the system with to ensure the expected run time is launched*
- *Launch Integrity is often implemented as a Load Integrity solution*
- *The Launch Integrity trust chain are the TCB components conducting the transitive trust operations*
- *A Launch Integrity trust chain is susceptible to manipulation by all TCB components it loads or inherits*
- *Dynamic Launch enables early launch and late launch operational models*

Early Launch

SRTM Supplement

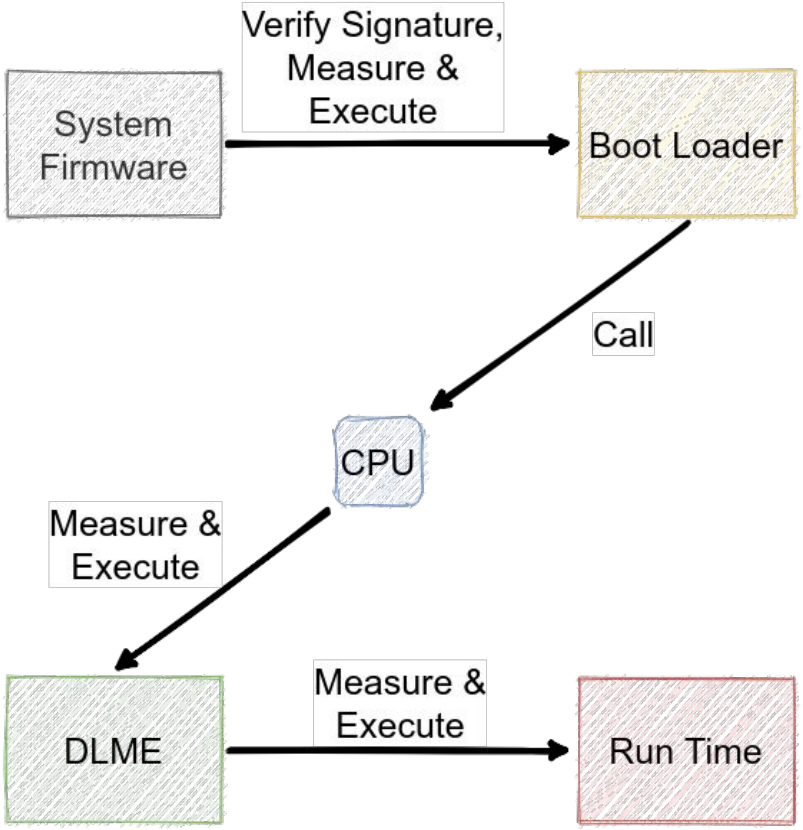
Purpose: To reset the Root of Trust for the launch integrity trust chain used to start the run time.

Reasons:

- Remove/replace the trust chain started by system firmware
- For UEFI, switch to a trust chain that is anchored with an RTM
- When system firmware RTV is not rooted in hardware, e.g. no BootGuard/Titan/DICE

Challenges:

- Increases the boot time
- RTM requires a separate assessment action



SecureLaunch Implementation

In SecureLaunch this is referred to as the First Launch use case and is the primary focus for initial implementation and release.

Work in progress:

- GRUB: Adding preamble logic to initiate a DL event
- Linux: Adding DL entry point and a kexec exit point
- U-root: An exemplar init that measures, preserves log, and kexec preparation

Measured Secure Boot

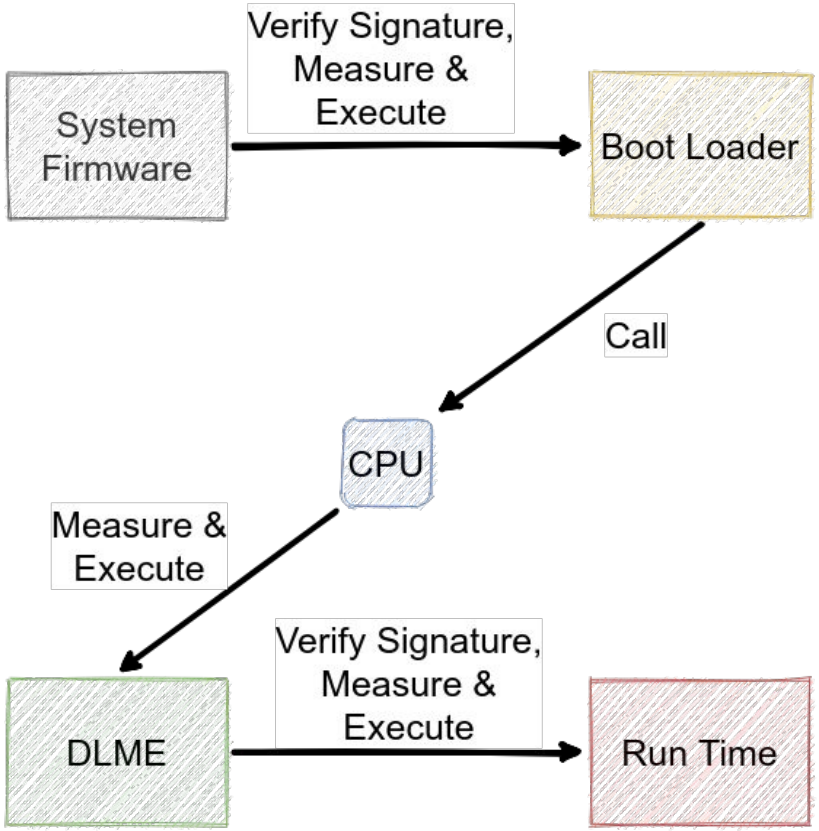
Purpose: Enable the existing Secure Boot plumbing while being rooted in hardware

Reasons:

- Removes complicated CRL revocation process
- Eliminate Microsoft from trust chain without OEM complications, e.g. firmware upgrade resetting Secure Boot keys
- DRTM PCRs are more predictable/less fragile than SRTM PCRs

Challenges:

- Increases the boot time
- RTM requires a separate assessment action



SecureLaunch Implementation

This is an extended version of the SRTM Supplement (First Launch) implementation.

Work In Progress:

- AMD: implementation has a MSB Key Hash field planned for in the SLB header

In Planning:

- Intel: it is still being worked where best it would be to store/pass the certificate hash to ensure it is included in the earliest possible DRTM measurement.
- Approach for activating Linux SecureBoot infrastructure from DLME will need to be determined

Late Launch

Secure Upgrade (Firmware and OS)

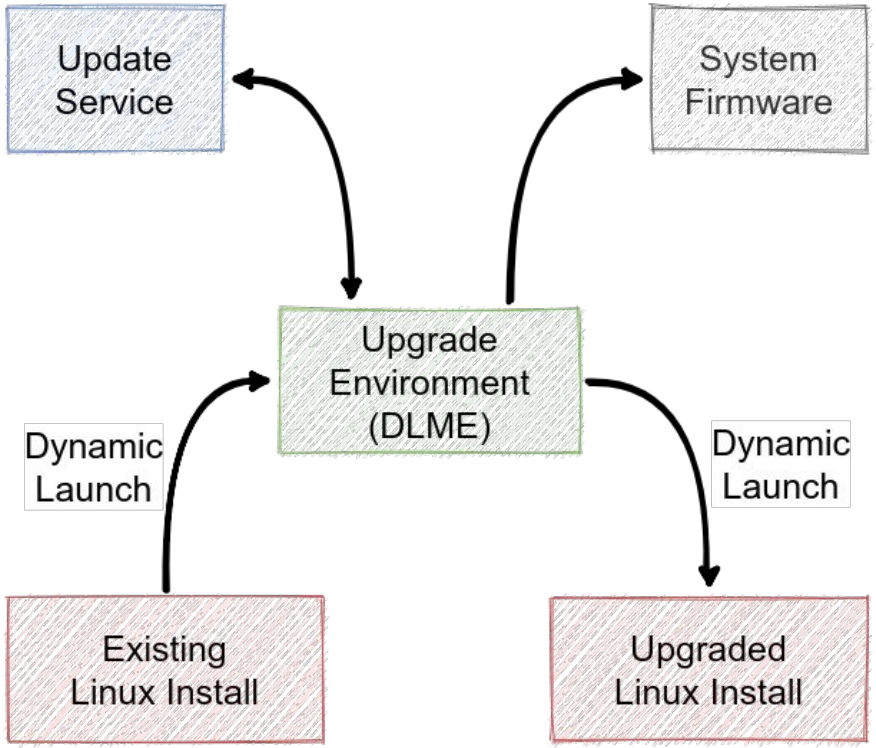
Purpose: To create a controlled and verifiable boundary to move from an unpatched state to a patched state.

Reasons:

- Machine reboot can be an expensive (timely) operation but require a fresh and/or short trust chain rooted in hardware
- Allows the ability to enforce only a controlled and verified runtime is used for firmware interactions

Challenges:

- Introduces overhead of external update service with additional overhead of attestation validation in the update service



SecureLaunch Implementation

For SecureLaunch this is referred to as the Relaunch use case and is the next one to be worked after the First Launch use case has been released.

In Planning:

- Linux: Implement a Dynamic Launch preamble kexec operation

Secure Wakeup

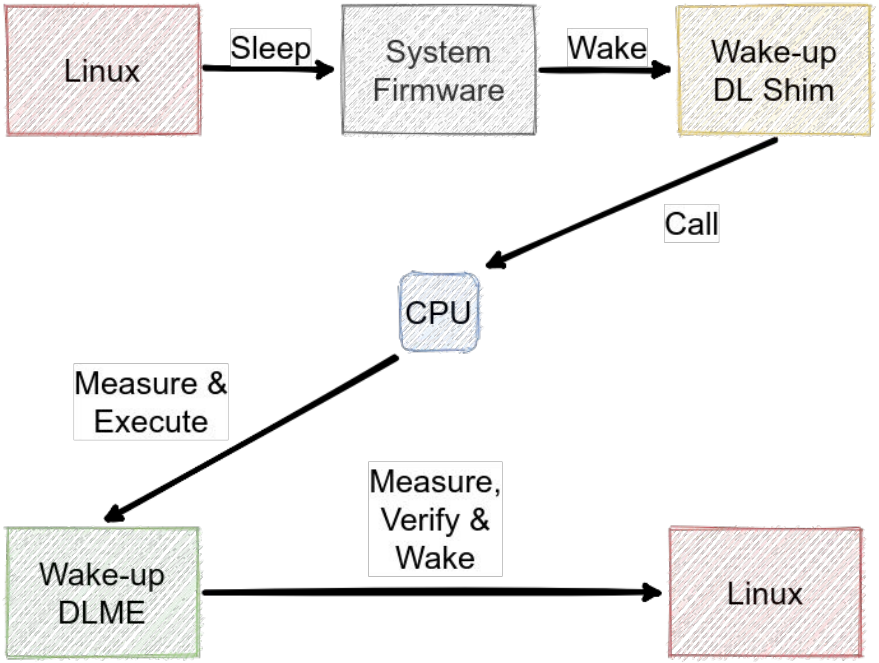
Purpose: To verify the integrity of the Linux kernel post sleep event.

Reasons:

- During sleep the kernel loses control of the platform
- DL Event puts hardware into a known controlled state

Challenges:

- Crafting an assessment action to validate the RTM chain



SecureLaunch Implementation

The Secure Wakeup use case is not currently on the roadmap for TrenchBoot, though some may find it an appealing application of Dynamic Launch.

Isolated Execution (Flicker)

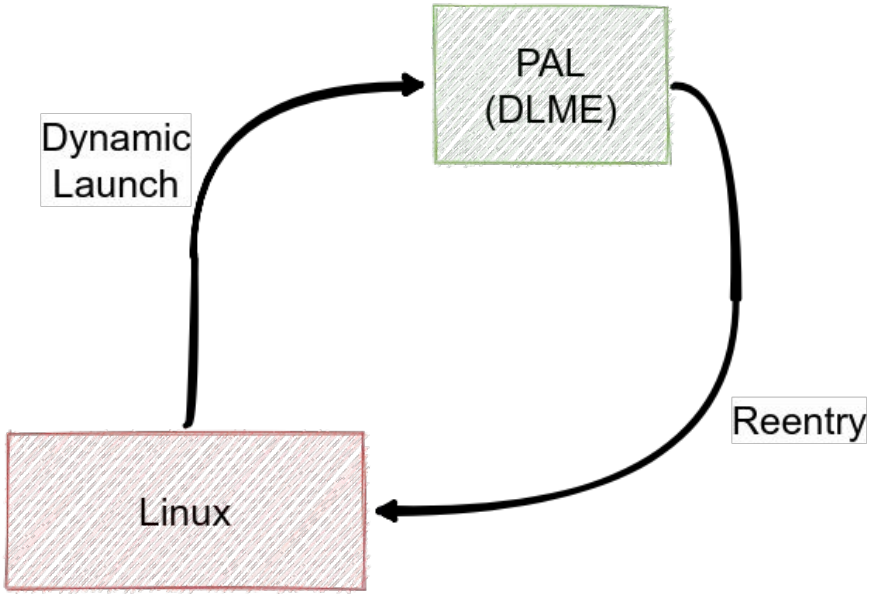
Purpose: "*An infrastructure for executing security sensitive code in complete isolation while trusting as few as 250 lines of additional code.*"

Reason:

- Verified, isolated execution of code without threat of kernel or user-space interference

Challenges:

- Is a disruptive event that causes unexpected system state change for the Linux kernel



SecureLaunch Implementation

It is possible that a Kernel Runtime Integrity could be ran in an Isolated Execution implementation to introspect the running instance of Linux. This would be done when it is acceptable to cause a significant pause in system execution to validate that there is no presence of a kernel rootkit before allowing the system to then execute a sensitive operation.

Roadmap:

- The TrenchBoot Project for the near term is planning for a Xen hypervisor based implementation
- Contribution running Linux Kernel Runtime Guard, or something similar, as a DLME would be welcomed

Future Capability Development

While it is possible to implement a portion of the use cases presented today, others depend on capabilities that require further development or even invention. Here are capabilities that would either enable or enrich the use cases.

- A public database of known good measurements and optional attestation service
  - A possible solution would be to collaborate with LVFS (fwupd.org)
- An external runnable Runtime Integrity capability for the Linux kernel
  - The LKRG project is laying a solid foundation for such a capability
- A cross-vendor virtual Dynamic Launch instruction for hypervisor
  - There are qemu implementations of SKINIT and SENTER that exist

Project Info

Website: https://trenchboot.org

Github: https://github.com/TrenchBoot

Slack: #trenchboot on https://osfw.slack.com

Mailing List:
https://groups.google.com/g/trenchboot-devel

Past Presentations:
https://github.com/TrenchBoot/documentation/tree/master/presentations

Questions??

Backup Slides

Trust Computing Concepts

Trust: Empowering an entity with the responsibility to perform an action on your behalf

Trusted: The entity or state of being that has been empowered with a responsibility

Trustworthy: Confidence in the trust imparted to a trusted entity

Trusted Computing Base: All entities that either are responsible for the security properties or is outside of the purview of these entities

Transitive Trust: The empowering of an entity with a trust by a trusted entity that may or may not be the original entity that delegated the responsibility

Trust Chain: The set of entities specifically involved in propagating trust through a transitive trust

Trust Computing Concepts (cont’d)

Load Integrity: Checking the integrity of an entity when loaded into memory

Runtime Integrity: Checking the integrity of an entity after execution has began

Early Launch: Using Dynamic Launch as a supplement to the Static Launch (UEFI Phases: SEC - TSL)

Late Launch: Using Dynamic Launch after the target run time (OS) has started