December 3 - 6, 2018
Santa Clara Convention Center
CA, USA

REVOLUTIONIZING
THE COMPUTING
LANDSCAPE AND
BEYOND.

https://tmt.knect365.com/risc-v-summit

@risc_v
SECURE BOOTSTRAPPING OF TRUSTED SOFTWARE IN RISC-V

Ilia Lebedev
Graduate Student
Massachusetts Institute of Technology

https://tmt.knect365.com/risc-v-summit
Secure Bootstrapping of Trusted Software in RISC-V

Ilia Lebedev, Kyle Hogan, Jules Drean, Srini Devadas - MIT

This work was partially funded by Delta Electronics, Analog Devices, DARPA & SPAWAR (N66001-15-C-4066), and DARPA SSITH (HR001118C0018).
We often have no idea whether trust is justified

I expect a specific software environment that I trust; I call it Steve.

Hi, are you Steve?

I don’t know if I can trust you :'(

Ok, sure, I’m Skeeve. Steeve? Steve.
Secure Bootstrapping of Trusted Software in RISC-V

I trust a binary, which I call Steve

I also trust the manufacturer of this hardware

Untrusted environment, software adversary

Oh hi! I am Steve and I am securely loaded on trusted hardware

- https://medium.com/@ilia.lebedev/secure-boot-2d6e319b6978

( this talk lives here )
Consider trust: actors in a system (1/2)

- How is each trusted?
- What specific guarantees can be made?

- Hardware
  - Root of trust software
  - Post-boot software (inc. third party modules)
  - user-provided software (scripts in a web page)

- anonymous remote users
- Authenticated remote users
- Users with physical access

How is each trusted?
What specific guarantees can be made?
Guarantee **integrity** for loading a **boot image** in an **untrusted setting**, given pre-existing trust in HW, manufacturer, desired boot image.
Threat Model

Unconditional trust:
- Manufacturer’s keys, trusted HW
- Desired boot image (key privacy)
- ... and you! (the first party)

Adversary: boot a different, malicious binary, interact with it
... but not tamper with the the hardware
... or extract its secret key
... or deny service
1: \textbf{M} is a trustworthy CA

Manufacturer securely generates and stores $SK_M$

\{ $SK_M$, $PK_M$ \}

Publish $PK_M$

\textit{ed25519 cryptosystem}\footnote{RSA is also a reasonable choice}

We will document our chain of trust here:
2: M builds a trustworthy D with keys

M builds a trustworthy D with keys

\{SK_M, PK_M\} → Publish PK_M

D has some properties:
- Trustworthy keys
- No back doors
- A trusted ROM
- First instruction integrity

*D exposes SK_D via a special interface, and is hidden from normal software.
3: M certifies PK$_D$

- M publishes their public key PK$_M$.
- M signs their public key with their private key: $\text{Sign}(\text{PK}_D, \text{SK}_M)$.
- M publishes their endorsement of PK$_D$: Cert$_D$.

Certificate Authority

- Proves* that a device with access to SK$_D$ is D manufactured by M.

*trust assumptions...
4: D executes from a root of trust at reset

D must guarantee first instruction integrity

1). Erase (or re-key) memory … secrets from previous boot
    erasing GBs of DRAM is very slow!

2). Load an untrusted binary … assuming it has certain structure
    (size, reserved space for keys, etc.)
Hash the binary $S$ in memory $\to H_S$

**Certificate Authority**

**Manufacturer**

**Processor D**

**Software S**

**Root of trust measures** $S \to H_S$

- **Hash** the binary $S$ in memory $\to H_S$

- **Certificate Authority** issues a certificate for $S$

- **Manufacturer** signs the certificate and sends it to the **Processor D**

- **Processor D** verifies the certificate and stores it in a secure area

- **Software S** includes the certificate in its execution environment

- **Root of trust** is established through the certificate

Note: can shrink the ROM by hashing and verifying the bootloader in RAM
6: Root of trust endows $S$ with unique keys

*D has access to $SK_D$, puts it on the stack.

$$KDF\left\{H_S, SK_D\right\} \rightarrow \left\{SK_S, PK_S\right\}$$

**Certificate Authority**

Manufacturer

Processor D

Software S

Note: can store encrypted $SK_S$ in public, use SHA3($SK_D$) to encrypt/decrypt
7: **Root of trust** certifies \( \{H_S, PK_S\} \)

*SK* \(_D\) is on the stack.

\[ \text{Sign}(\{H_S, PK_S\}, SK_D) \rightarrow cert_S \]

**Certificate Authority**

- **Manufacturer**
- **Processor D**
- **Software S**

\( 1024B \)

- stack
  - \( 8B \) \( \text{sizeof(S)} \)
  - \( 64B \) \( H_S \)
  - \( 64B \) \( SK_S \)
  - \( 32B \) \( PK_S \)
  - \( 64B \) \( cert_D \)
  - \( 32B \) \( PK_D \)
  - \( 32B \) \( PK_M \)
8: **Root of trust cleans up and boots**

**Secrets are on the stack**

.. and in *μarchitectures*

.. and $SK_D$ must be hidden

---

**Certificate Authority**

**Manufacturer**

**Processor D**

**Software S**

---

Secrets are on the stack

.. and in *μarchitectures*

.. and $SK_D$ must be hidden
9: **S** performs attestation (1/4)

**Trusted first party**

- **PK**
- **Cert**
- **Hash**
- **SK**

... has a priori trust in **PK**

**S on D**, trusted to maintain secrecy of **SK**

Yes! Here is a **fresh** proof that I am **Steve** running on device built by 🤖

Hi, are you **Steve**?

Ok, thanks
9: **S** performs attestation (2/4)

An attestation only proves the **sender has access to** a genuine system.

A **man-in-the-middle adversary** (even a local one) may pass attestation
9: **S performs attestation (3/4)**

**Trusted first party**

**Key agreement** to defeat a man-in-middle; encrypt attestation with $K$ and include $K$ in the attestation

The adversary cannot compute $K$

( did $S$ and the verifier compute the same $K$? )

**S on $D$, trusted to maintain secrecy of $SK_s$**

Other key agreement protocols exit too (e.g. Diffie Hellman)
We did it!

Yes! Here is a **fresh** proof **computed by me** that I am **Steve** running on device built by 🥳

Hi, are you Steve?

(¬¬)σ

\(\checkmark(\cdot \omega \cdot o)\)

Trusted first party

S on D

Ok, thanks

Read the [Medium article](https://example.com) for extensions to this boot protocol to make it more flexible
Booting with **Multiple Harts**

**Core 0**
- 0x1000
- Is this core 0?
- do bootloader things
- Send IPI to cores
- Yeah!
- Boot address

**Core not 0**
- 0x1000
- Is this core 0?
- Enable IPI
- WFI
- No :(
- Boot address

+20 ASM instructions

+7 ASM instructions

(time)

(barrier via interrupts)
Trust S w/o 1st party: gatekeeping (1/2)

**IDEA 1:** White-list the allowed S in the root of trust.

The space of software that can boot with an integrity guarantee from the RoT is pre-defined and immutable.
IDEA 2: Delegate white-listing of $S$ to a certificate.

(Panic if the Cert$_S$ is invalid)
If a **small TCB** is the goal...

Not judging, but feature-packed bootloaders are huge:

- U-BOOT: ~2,000,000 LOC
- GRUB: ~326,000 LOC
- GRUB Legacy: ~43,000 LOC
The things we’ve swept **under the rug**...

- **How can we trust** $S$ **is keep** $SK_s$ **secret?**
  
  Sanctum/Keystone, or at least a PMP rule

- **Where does** $SK_D$ **come from, exactly?**

- **What is our trust in the Manufacturer?**

  PUF-backed keys
THANK YOU

https://tmt.knect365.com/risc-v-summit