Towards End-to-end TEE Verification with Keystone

Shweta Shinde
UC Berkeley
TEEs reduce the Trusted Computing Base

Operating Systems
10 MLOC

Hypervisor
150 KLOC

CVEs in Linux [CVE-DB]

CVEs in Xen [CVE-DB]

Vulnerabilities By Year

Web Server Enclave
Other Apps
RAM
Enclave Memory

Ring 3
OS / Hypervisor

Ring 0 - 2

Trustworthy Hardware

Trusted
Untrusted
We strive for lower TCB for TEEs
Makes them amenable to verification
What is the estimated Keystone TCB?

![Keystone TCB Diagram]

<table>
<thead>
<tr>
<th>Component</th>
<th>Runtime</th>
<th>SM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>1730</td>
<td>1000</td>
</tr>
<tr>
<td>Memory Isolation</td>
<td>—</td>
<td>530</td>
</tr>
<tr>
<td>Free Memory</td>
<td>310</td>
<td>—</td>
</tr>
<tr>
<td>Dynamic Memory</td>
<td>100</td>
<td>70</td>
</tr>
<tr>
<td>Edge-call Handling</td>
<td>300</td>
<td>30</td>
</tr>
<tr>
<td>Syscalls</td>
<td>190</td>
<td>—</td>
</tr>
<tr>
<td>libc Environment</td>
<td>50</td>
<td>—</td>
</tr>
<tr>
<td>IO Syscall Proxying</td>
<td>260</td>
<td>—</td>
</tr>
<tr>
<td>Cache Partitioning</td>
<td>—</td>
<td>300</td>
</tr>
</tbody>
</table>

Around 10 KLoC
Can we do an end-to-end verification of TEEs?

Some Open Challenges

- Verify-then-trust model
- What is the TCB of this stack?
- What are the specifications of each layer?
- Under which threat model do the properties hold true?
- Can the upper layer verify & trust the properties guaranteed by lower layers?
Ongoing Verification Efforts for TEEs

Kami: A Platform for High-Level Parametric Hardware Specification and Its Modular Verification

JOONWON CHOI, MURALIDARAN VIJAYARAGHAVAN, BENJAMIN SHERMAN, ADAM CHLIPALA, and ARVIND, MIT CSAIL, USA

The CHERI capability model: Revisiting RISC in an age of risk

Jonathan Woodruff† Robert N. M. Watson† David Chisnall† Simon W. Moore† Jonathan Anderson† Brooks Davis‡ Ben Laurie§ Peter G. Neumann‡ Robert Norton‡ Michael Roe‡
†University of Cambridge ‡SRI International §Google UK Ltd
firstname.lastname@cl.cam.ac.uk {neumann,brooks}@cs1.sri.com ben1@google.com

seL4: Formal Verification of an OS Kernel

Gerwin Klein1,2, Kevin Elphinstone1,2, Gernot Heiser1,2,3
June Andronick1,2, David Cock1, Philip Derrin1, Dhammika Eldadwe1,2,2 Kai Engelhardt1,2
Rafal Kolanski1,2, Michael Norrish1,4, Thomas Sewell1, Harvey Tuch1,2, Simon Winwood1,2
1NICTA, 2UNSW, 3Open Kernel Labs, 4ANU
ertos@nicta.com.au

Komodo: Using verification to disentangle secure-enclave hardware from software

Andrew Ferraiuolo Cornell University
Andrew Baumann Microsoft Research
Chris Hawblitzel Microsoft Research
Bryan Parno Carnegie Mellon University

A Formal Foundation for Secure Remote Execution of Enclaves

Pramod Subramanyan University of California, Berkeley
pramod@berkeley.edu
Rohit Sinha University of California, Berkeley
rsinha@eecs.berkeley.edu
Ilia Lebedev Massachusetts Institute of Technology
ilebedev@mit.edu
Srinivas Devadas Massachusetts Institute of Technology
devadas@mit.edu
Sanjit A. Seshia University of California, Berkeley
seshia@eecs.berkeley.edu

A Design and Verification Methodology for Secure Isolated Regions

Rohit Sinha1 Manuel Costa2 Akash Lai3 Nuno P. Lopes2
Sriram Rajamani1 Sanjit A. Seshia1 Kapil Vaswani2
1University of California at Berkeley, USA 2Microsoft Research, UK 3Microsoft Research, India
{rsinha,ssesha}@eecs.berkeley.edu {manuel,akashl,nlopes,sriram,kapilv}@microsoft.com
Keystone Design Makes Verification Easier

• The modular design allows to reduce TCB per use-case
  • Each component has well-defined APIs and properties by design

• Existing components can be independently verified
  • Benefit from existing verification efforts in the TEE space
  • Easy to verify new components before integration
Keystone Design Makes Verification Easier

- Three on-going efforts which demonstrate this:
  - **Extending Trusted Abstract Platform (TAP) to Keystone:** Formalization of idealized enclave platforms along with a parameterized adversary
  - **BesFS:** Mechanized Proof of a Iago-Safe Filesystem for Enclaves
  - **Friscy:** Formal verification PMP implementation and use in the security monitor

---

PMP Implementation & Use

**FRISCY:** Formal verification of the RISC-V Rocket Chip implementation of PMP and the use of PMP in the security monitor

Cameron Rasmussen, Kevin Cheang, Kevin Lauer and Dayeol Lee
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, USA
crasmussen,kcheang,laefer,dayeol}@berkeley.edu

---

Iago-Safe Filesystem API

**BesFS:** Mechanized Proof of an Iago-Safe Filesystem for Enclaves

Shweta Shinde*
University of California, Berkeley
shwetas@berkeley.edu

Aquinas Hobor
National University of Singapore & Yale-NUS College
hobor@comp.nus.edu.sg

Abhik Roychoudhury
National University of Singapore
abhik@comp.nus.edu.sg

Pinghai Yuan
National University of Singapore
yuanping@comp.nus.edu.sg

Prateek Saxena
National University of Singapore
prateek@comp.nus.edu.sg
Effort #1 BesFS: Mechanized Proof of a Iago-Safe Filesystem for Enclaves

- **Iago Attacks**: “Manipulate system call return values for arbitrary computation at the malicious kernel’s behest” [ASPLOS’13]
Iago Attacks via the Filesystem API

• (A1) File Content Manipulation
  • Maps same page to multiple files/ parts of a file

• (A2) Paths & File Descriptor Mismatch
  • Returns wrong file id, opens wrong file

• (A3) Size Mismatch
  • Under or over write/reads

• (A4) Error Code Manipulation
  • Returns success without doing the operation

Why do we need a verified file system interface?
Because checks are incomplete
BesFS State Properties

• All the file and directory paths are unique, there are no circular paths in the file system
  \[ \text{dom}(N) = \mathcal{P} \]
  \[ \forall (p, p') \in \mathcal{P} \times \mathcal{P}, p \neq p' \Rightarrow N(p)_{id} \neq N(p')_{id} \quad (\text{SP1}) \]
• All open file IDs have to be registered in O
  \[ \forall o \in \mathcal{O}, \exists p \text{ s.t. } p \in \mathcal{P} \wedge N(p)_{id} = o_{id} \quad (\text{SP2}) \]
• All open file IDs have unique entries
  \[ \forall (o, o') \in \mathcal{O} \times \mathcal{O}, o_{id} = o'_{id} \Rightarrow o = o' \quad (\text{SP3}) \]
• No overlaps between addresses & one-to-one mapping from virtual address to content
  \[ \forall p \in \mathcal{P}, o \in \mathcal{O}, N(p)_{id} = o_{id} \Rightarrow o_{\text{Cursor}} < N(p)_{\text{Size}} \quad (\text{SP4}) \]
• Current cursor position can only take values between 0 and EOF
  \[ \forall f, \forall o \text{ s.t. } p \in \mathcal{P} \wedge f = N(p)_{id} \wedge o < N(p)_{\text{Size}} \]
  \[ \Rightarrow M(f, o) \neq \perp \quad (\text{SP5}) \]
<table>
<thead>
<tr>
<th>TP_i</th>
<th>BesFS Interface</th>
<th>Pre-condition ( \text{Pre}(s) )</th>
<th>Transition Relation ( \tau_i(s, S') )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP_1</td>
<td>fs_close (h : Id, e : Error)</td>
<td>( \exists o, o_{14} = h \land o \in O )</td>
<td>( S' = S[O/O - {o}] ) ( \land ) ( e = ESucc )</td>
</tr>
<tr>
<td>TP_2</td>
<td>fs_open (h : Id, e : Error)</td>
<td>( \exists p \in P )</td>
<td>( S' = S[O/O + {N(p)<em>{id} \neq o</em>{14}}] ) ( \land ) ( h = N(p)_{id} )</td>
</tr>
<tr>
<td>TP_3</td>
<td>fs_mkdir (p : Path, r : Perm, e : Error)</td>
<td>( \exists p \in P \land \land N(Paren(p)) = \text{True} )</td>
<td>( S' = S[P/P + {p}, \land N(N({p}_{id}) \rightarrow (h, r, 0)) ) ( \land ) ( e = ESucc )</td>
</tr>
<tr>
<td>TP_4</td>
<td>fs_create (p : Path, r : Perm, e : Error)</td>
<td>( \exists p \in P \land \land N(Paren(p)) = \text{True} )</td>
<td>( S' = S[P/P + {p}, \land N(N({p}_{id}) \rightarrow (h, r, 0)) ) ( \land ) ( e = ESucc )</td>
</tr>
<tr>
<td>TP_5</td>
<td>fs_remove (p : Path, e : Error)</td>
<td>( \exists p \in P \land \land N(Paren(p)) = \text{True} )</td>
<td>( S' = S[P/P - {p}] ) ( \land ) ( e = ESucc )</td>
</tr>
<tr>
<td>TP_6</td>
<td>fs_rmdir (p : Path, e : Error)</td>
<td>( \exists p \in P \land \land \forall q \in P, Parent(q) \neq \land N(Paren(p)) = \text{True} )</td>
<td>( S' = S[P/P - {p}] ) ( \land ) ( e = ESucc )</td>
</tr>
<tr>
<td>TP_7</td>
<td>fs_stat (h : Id, r : Perm, n : String, l : N, e : Error)</td>
<td>( \exists o, o_{14} = h \land o \in O \land \land N(Paren(p)) = \text{True} )</td>
<td>( S' = S[O/O - {o}] ) ( \land ) ( e = ESucc \land l = N(p)<em>{size} \land n = N(p)</em>{name} )</td>
</tr>
<tr>
<td>TP_8</td>
<td>fs_readlink (p : Path, e : Error)</td>
<td>( \exists p \in P )</td>
<td>( S' = S[O/O - {o}] ) ( \land ) ( e = ESucc \land n \in l, p + n \in P )</td>
</tr>
<tr>
<td>TP_9</td>
<td>fs_chmod (p : Path, r : Perm, e : Error)</td>
<td>( \exists p \in P )</td>
<td>( S' = S[O/O - {o}] ) ( \land ) ( e = ESucc \land \land M(h, o_{read}), \ldots, M(h, o_{write}) )</td>
</tr>
<tr>
<td>TP_10</td>
<td>fs_inode (h : Id, l : N, e : Error)</td>
<td>( \exists o, o_{14} = h \land o \in O \land \land N(Paren(p)) = \text{True} )</td>
<td>( S' = S[O/O - {o}] ) ( \land ) ( e = ESucc \land b = M(h, o_{read}), \ldots, M(h, o_{write}) )</td>
</tr>
<tr>
<td>TP_11</td>
<td>fs_write (h : Id, l : N, b : [Byte], e : Error)</td>
<td>( \exists o, o_{14} = h \land o \in O \land \land N(Paren(p)) = \text{True} )</td>
<td>( S' = S[O/O - {o}] ) ( \land ) ( e = ESucc \land b = M(h, o_{write}), \ldots, M(h, o_{write}) )</td>
</tr>
<tr>
<td>TP_12</td>
<td>fs_truncate (h : Id, l : N, e : Error)</td>
<td>( \exists o, o_{14} = h \land o \in O \land \land N(Paren(p)) = \text{True} )</td>
<td>( S' = S[O/O - {o}] ) ( \land ) ( e = ESucc \land \land M(h, o_{write}), \ldots, M(h, o_{write}) )</td>
</tr>
</tbody>
</table>
BesFS Proof

• **State Transition Safety.** Given a good state $S$ satisfying pre-conditions $\text{pre}_i$, then if we execute $f_i$ to reach state $S'$, then $S'$ is always a good state and relation between $S$ and $S'$ is valid according to the transition relation $\tau_i$:

$$\forall S, S', i. \quad S \models SP1\text{--}SP5 \land \text{pre}_i(S) \land S \xrightarrow{f_i} S' \Rightarrow \tau_i(S, S') \land S' \models SP1\text{--}SP5$$

• **Sequential Composition Safety.** Given a good initial state $S_0$ subject to a sequence of transitions $\tau_{m1}, \ldots, \tau_{mn}$ always produces a good final state $S_n$

$$S_0 \models SP1\text{--}SP5 \land S_0 \xrightarrow{f_{m1}} S_1 \land S_1 \xrightarrow{f_{m2}} S_2 \land \cdots \land S_n \xrightarrow{f_{mn}} S_n$$

$$\Rightarrow \land \tau_{m1}(S_0, S_1) \land \tau_{m2}(S_1, S_2) \land \cdots \land \tau_{mn}(S_{n-1}, S_n) \land S_n \models SP1\text{--}SP5$$
BesFS Summary of Results

• **TCB:** 3676 LOC Coq, 1.5K LOC in C

• **Do Proofs Help in Eliminating Bugs?**

  – Seek Specification Bug
    • if pos < size

  – Write Implementation Bug
    • Variable scope overlaps

  – Intel SGX SDK Bugs
    • enclave’s stack is corruption for large sizes

  – Error Code Bugs
    • 7 distinct functions where error codes were incorrect
Effort #2 Extending Trusted Abstract Platform (TAP) to Keystone

- Prove **secure remote execution** for Keystone security monitor
  - SRE decomposed into separate proofs on integrity, confidentiality, and measurement
  - Proofs verified under various parameterizations of the adversary
Effort #3 FRISCY: Formal Verification of PMP Implementation & Use in the Security Monitor

• Goal: Verify the Keystone memory isolation implementation in the SM
  • Extract the PMP Model from Rocket Chip SOC implementation
  • Prove PMP properties and use these properties to further check SM implementation
Can we do an end-to-end verification of TEEs?

A long way to go until all the layers of the TEE are fully verified.

Some open challenges:
- Under which threat model do the properties hold true?
- Can the upper layer verify & trust the properties guaranteed by lower layers?
End-to-end Verified TEE Ecosystem

- Eapp Logic
- Untrusted and enclave APIs
- SM, RT, SDK libraries
- Attestation
- Secure Boot
- PMP spec and usage
- RISC-V chip components
- SOC design & manufacturing

- A well-defined adversary model, specifications of each layer, properties expected at the layer interfaces
- Standard way of composing and customizing components while retaining verification guarantees

Let’s work together towards this grand vision!
Thank You!

Shweta Shinde
shwetas@berkeley.edu