SECURE IDENTIFICATION of ACTIVELY EXECUTED CODE on a GENERIC TRUSTED COMPONENT

BRUNO VAVALA
CMU / UL

Nuno Neves
UL

Peter Steenkiste
CMU

IEEE/IFIP Conference on Dependable Systems & Networks (DSN’16)
Trusted Executions: trends & tradeoffs
Anatomy of a Trusted Execution

untrusted third party
client

Execute
Anatomy of a Trusted Execution

1. **Load & Identify**
2. **Execute**
3. **Attest**

- Secure environment
- Untrusted third party
- Client

**Outsource**

**Verify**
Anatomy of a Trusted Execution

- Code must have an ID (i.e. hash)
- Trusted hardware computes ID
- Hardware attests the ID
- Client trusts hardware & ID
**Generic TCC Interface**

- **execute**
  - code loading + identification + isolated execution
- **attest**
  - TCC-signed code identity and I/O data
- **verify**

Implementable with:
- Intel TXT + TPM
- Hypervisor-based TCC
- Intel SGX
- ...
Trends

- **TOCTOU**
  Problem: static measurements do not reflect later changes.
  [2005-]

- **Dynamic Root of Trust**: build a new robust and verifiable chain of trust on demand.
  [2008]

- **Fast Trusted Computing**: combine slow trusted chips with software on main CPU.
  [2010-]

- **Large Trusted Executions**: implement large services in the trusted environment.
  [2011-]

---

- **Reduced TCB**
- **On-demand execution**
- **Improved efficiency**
- **Richer services in TCB**
- **+security +efficiency**
Security/Efficiency Tradeoff for Large-Scale Services

- Identify-once-execute-once
- Identify-once-execute-forever

Security: high, low
Efficiency: high, low
Trusted Executions: trends & tradeoffs

Identifying Actively Executed Code

Practical Analysis

Conclusions

Problem definition
/* SQLite code */
int main () {
    switch(op) {
        case SELECT:
            do_select();
        case DELETE:
            do_delete();
        case INSERT:
            do_insert();
        case FOOBAR:
            do_foobar();
    }
}

source code

binary

code identity
Execution Verification
Identified ≠ Executed

identified binary code

> 

actually executed binary code
Desirable Properties

- Identifying what is “actually” executed
- TCC agnostic execution
- Keeping efficient client-side verification
Trusted Executions: trends & tradeoffs

Problem definition

Practical Analysis

Conclusions

Identifying Actively Executed Code
• OS and services are untrusted
• Client knows service identity and TCC certificate
Enriching the Interface

- **execute**
  - code loading + identification + isolated execution
- **attest**
  - TCC-signed code identity and I/O data
- **auth-put**
  - secure storage for a specific recipient
    (TCC authenticates the sender)
- **auth-get**
  - secure storage from a specific sender
    (TCC authenticates the recipient)
- **verify**

Implementable with:
- Intel TXT + TPM
- Hypervisor-based TCC
- Intel SGX
- ...
One ID Per Code Module

/* SQLite code */
int main () {
    switch(op) {
        case SELECT:
            do_select();
            break;
        case DELETE:
            do_delete();
            break;
        case INSERT:
            do_insert();
            break;
        case FOOBAR:
            do_foobar();
            break;
    }
}
Execution Protocol

- Execution flows: A-to-B, A-to-C
- If C must be executed, then B is not loaded
Execution Protocol

- code base
- input, ID’s Tab
- A
- B
- C
- OS
- TCC
- hardware
Execution Protocol

code base

A

B

C

input, ID’s Tab

A

B

C

OS

TCC

hardware
Execution Protocol

- code base
- hardware
- A
- B
- C
- OS
- TCC

int. results, ID’s Tab

Bruno Vavala - IEEE/IFIP DSN'16
Execution Protocol

A \quad B \quad C

1. id(A)
2. id(B)
3. id(C)

int. results, ID’s Tab

identity table

OS

TCC

hardware

code base
Execution Protocol

- **code base**
  - A
  - B
  - C

- **OS**
  - B
  - C

- **TCC**
  - A

- **hardware**

- int. results, ID’s Tab

A → C
Execution Protocol

code base

A —> C
int. results, ID’s Tab

A B C

OS

TCC

hardware
Execution Protocol

code base

int. results, ID’s Tab

hardware
Execution Protocol

code base

A
B
C

A
B

OS

TCC

hardware

identity table

1. id(A)
2. id(B)
3. id(C)

identity table

int. results, ID’s Tab

secure channel
Execution Protocol

code base

A
B
C

A
B

OS

TCC

hardware

int. results, ID’s Tab

A
B
C
Execution Protocol

code base

A

B

C

A

B

OS

TCC

hardware

attested

output

ID’s Tab
Execution Protocol

code base

A

B

C

output
ID’s Tab

A

B

C

OS

TCC

hardware
Execution Protocol

A B C

OS

TCC

hardware

output ID’s Tab

attested
Execution Protocol
1. Client/Verifier does not verify intermediate results
   - Results are secured locally

2. Client does not verify execution flow
   - Verification of last module & ID’s Table implies correct execution flow

3. Build mutually authenticated secure channels
   - Using TCC-based secure storage

4. Fast (zero round) identity-based key sharing
   - Construction: 1 hash using sender-receiver identity pairs
     (see paper for details)

5. Avoid hash loops in general executions
   - Detach identity from code module using the ID’s Table
5. Avoiding Hash Loops

- General execution may have loops

code base
5. Avoiding Hash Loops

- General execution may have loops

![Diagram showing code base and problem (hash loop)]
5. Avoiding Hash Loops

- General execution may have loops

**code base**

**problem**
(hash loop)

**solution**
(ID’s in input table)

identity table

1. id(A)
2. id(B)
3. id(C)
Practical Analysis
Practical Analysis

- Hypervisor-based TCC Implementation
- Protocol applied to a real-world service (SQLite)
- End-To-End experiments on server cluster
SQLite (full implementation) is ~1MB

5-10x reduction of used code for single operation
(PAL = Piece of Application Logic)
• Same critical path, different code identification

• Monolithic SQLite is up 46% slower (w/ attestation)
Multi-PAL SQLite up to 2x faster (w/o attestation)
Trusted Executions: trends & tradeoffs

Problem definition
Identifying Actively Executed Code

Practical Analysis

Conclusions
Conclusions

- **Code identification** has security/efficiency tradeoffs

- **Identification** of just **actively executed** code can:
  - provide **fresher integrity guarantees**
  - **improved resource usage & performance**
  - be done **retrofitting existing trusted components**
THANKS

BRUNO VAVALA
CMU/UL

Nuno Neves
UL

Peter Steenkiste
CMU

IEEE/IFIP Conference on Dependable Systems & Networks (DSN’16)
BACKUP SLIDES
Identity Dependent keys
- Sender specifies recipient’s identity to TCC
- Recipient specifies sender’s identity to TCC
- Very efficient construction (one hash per-key)