Using Arm Features for Security Analysis

Fengwei Zhang

张锋巍

COMPASS Lab, SUSTech

August 4, 2021

Overview of The Talk

- Background: Hardware features on Arm
- Ninja: Towards transparent tracing and debugging
- Investigator: Finding the root cause of concurrency bugs
- COMPASS lab

Overview of The Talk

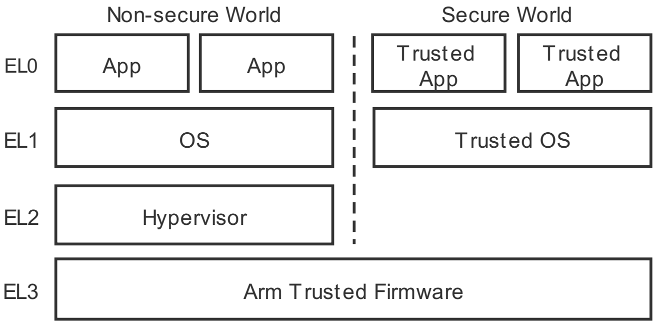
- Background: Hardware features on Arm
- Ninja: Towards transparent tracing and debugging
- Investigator: Finding the root cause of concurrency bugs
- COMPASS lab

ARM TrustZone

- **ARM TrustZone** divides the execution environment into a **secure** domain and a **non-secure** domain.
  - The RAM is partitioned to secure and non-secure regions.
  - The interrupts are assigned into the secure or non-secure group.
  - Hardware peripherals and secure-sensitive registers can be configured as secure access only.
- It is widely deployed in recent ARM processors.
- The OS runs in the **non-secure** domain, and only a few secure-sensitive payloads are executed in the **secure** domain.

Exception Levels

- Privileges in Armv8:
  - 3 Exception Levels
  - 2 Security Domains



Non-secure World

Secure World

EL0 App App Trusted App Trusted App

EL1 OS Trusted OS

EL2 Hypervisor

EL3 Arm Trusted Firmware

Embedded Trace Macrocell

Embedded Trace Macrocell (ETM) is a hardware component on Arm processors. It is able to tracing the instruction execution and memory access with negligible overhead.

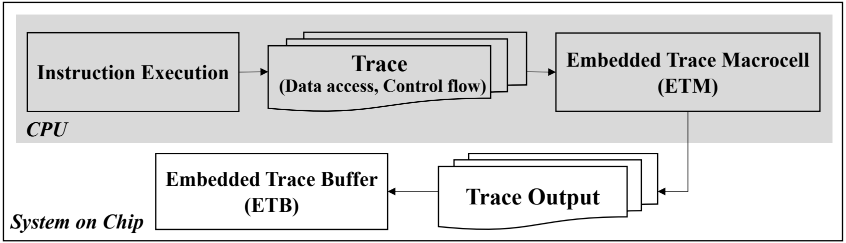


Figure: A general hardware model of ETM.

Trace-based Analysis Systems on Arm (ETM, etc)

- Ninja: Transparent Tracing and Debugging on Arm [USENIX Security'17, TIFS'19]
- HART: Hardware-assisted Kernel Module Tracing on Arm [ESORICS'20]
- Happer: Unpacking Android Apps via a Hardware-Assisted Approach [S&P'21]
- Alligator In Vest: Using Hardware Features for Failure Diagnosis on Arm

Trace-based Analysis Systems on Arm (ETM, etc)

- Ninja: Transparent Tracing and Debugging on Arm [USENIX Security’17, TIFS’19]
- HART: Hardware-assisted Kernel Module Tracing on Arm [ESORICS’20]
- Happer: Unpacking Android Apps via a Hardware-Assisted Approach [S&P’21]
- Alligator In Vest: Using Hardware Features for Failure Diagnosis on Arm

Overview of The Talk

- Background: Hardware features on Arm
- Ninja: Towards transparent tracing and debugging
- Investigator: Finding the root cause of concurrency bugs
- COMPASS lab

Transparency

What is transparent malware analysis?

Transparency

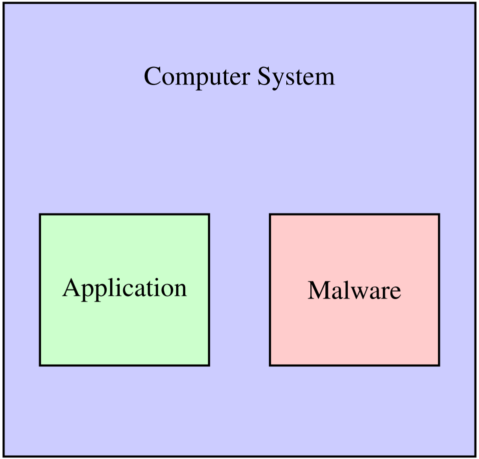
What is transparent malware analysis?

- Analyzing the malware without being aware.
- Transparent means that the malware cannot detect it.

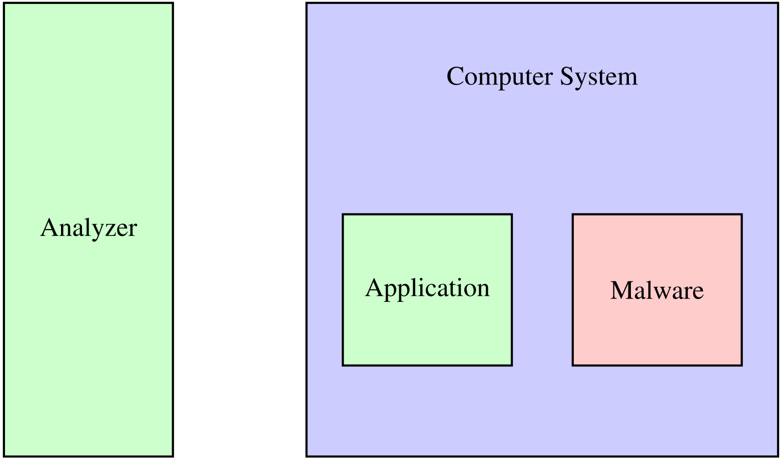
Transparency

Why transparency is important?

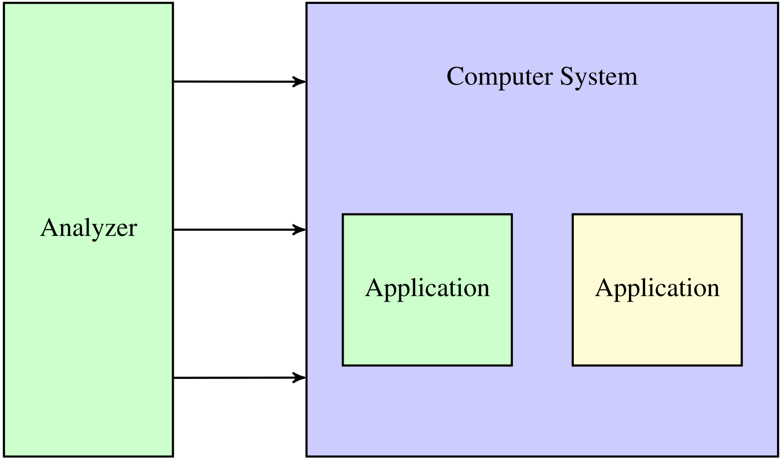
Evasive Malware



Evasive Malware



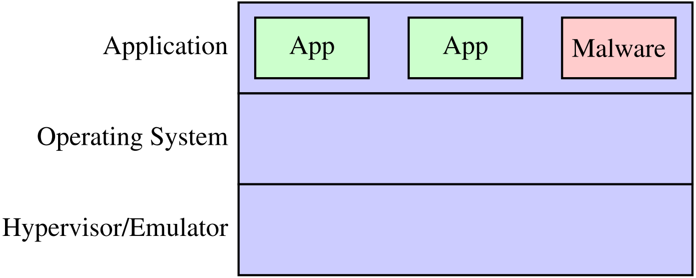
Evasive Malware



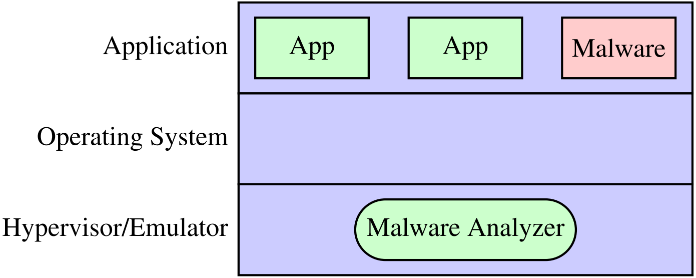
Malware Analysis

What is the current state of malware analysis systems?

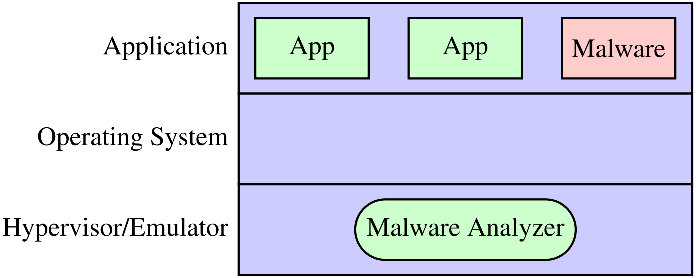
Malware Analysis



Malware Analysis

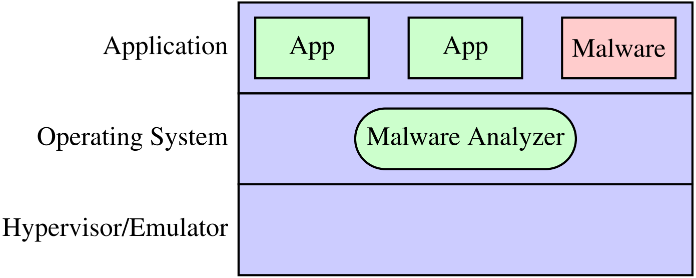


Malware Analysis

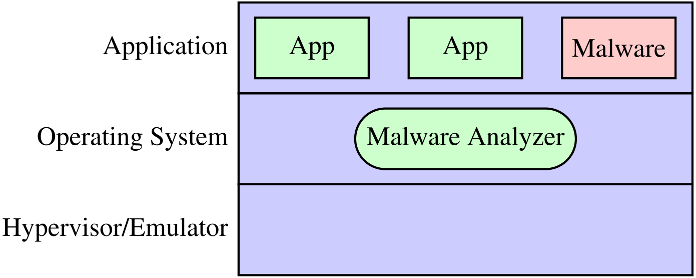


- Unarmed to anti-virtualization or anti-emulation techniques.
- Large performance overhead.

Malware Analysis



Malware Analysis



- Unable to handle malware with high privilege (e.g., rootkits).

Transparency Requirements

What makes a transparent malware analysis system?

Transparency Requirements

- An **Environment** that provides the access to the states of the target malware.

- An **Analyzer** which is responsible for the further analysis of the states.

Transparency Requirements

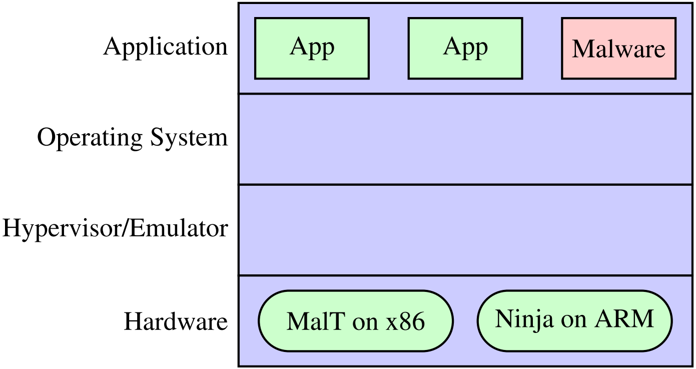
- An **Environment** that provides the access to the states of the target malware.
  - It is isolated from the target malware.
  - It exists on an off-the-shelf (OTS) bare-metal platform.
- An **Analyzer** which is responsible for the further analysis of the states.

Transparency Requirements

- An **Environment** that provides the access to the states of the target malware.
  - It is isolated from the target malware.
  - It exists on an off-the-shelf (OTS) bare-metal platform.
- An **Analyzer** which is responsible for the further analysis of the states.
  - It should not leave any detectable footprints to the outside of the environment.

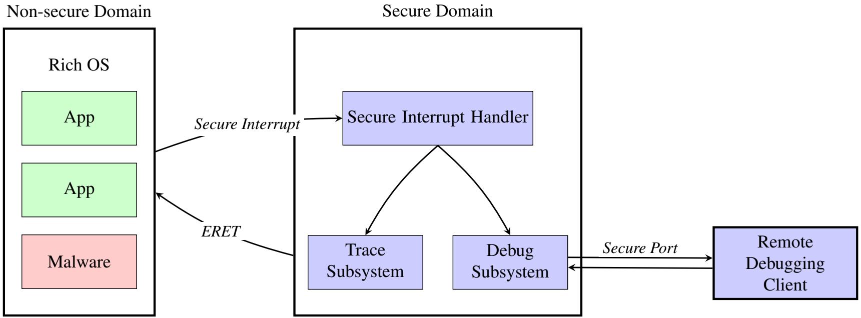
Towards Transparent Malware Analysis

- MalT on x86 Architecture [S&P'15, TDSC'18]
- Ninja on Arm Architecture [USENIX Security'17, TIFS'19]



Transparent Malware Analysis on Arm: Ninja

- Ninja [USENIX Security'17, TIFS'19]: Towards transparent tracing and debugging
- It uses TrustZone as the isolated execution environment.
- The debug subsystem is similar to MalT while the trace subsystem is immune to timing attacks.



Ninja — Performance

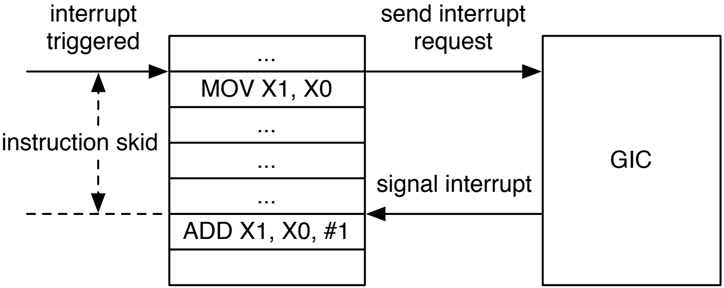
- Testbed Specification
  - ARM Juno v1 development board
  - A dual-core 800 MHZ Cortex-A57 cluster and a quad-core 700 MHZ Cortex-A53 cluster
  - ARM Trusted Firmware (ATF) v1.1 and Android 5.1.1

Table: Performance Scores Evaluated by CF-Bench

	Native Scores		Java Scores		Overall Scores	
	Mean	Slowdown	Mean	Slowdown	Mean	Slowdown
Tracing Disabled	25380		18758		21407	
Instruction Tracing	25364	1x	18673	1x	21349	1x
System Call Tracing	25360	1x	18664	1x	21342	1x
Android API Tracing	6452	4x	122	154x	2654	8x

Ninja — Minors

- Two-way semantic gaps.
  - Gap between secure domain and normal domain
  - Gap in Android Java virtual machine
- Instruction skid in interrupt



Overview of The Talk

- Background: Hardware features on Arm
- Ninja: Towards transparent tracing and debugging
- Investigator: Finding the root cause of concurrency bugs
- COMPASS lab

Failure diagnosis

Why is failure diagnosis important in production environment?

Failure diagnosis

Why is failure diagnosis important in *production environment*?

- Short release cycles make in-house testing unlikely to reveal all bugs.

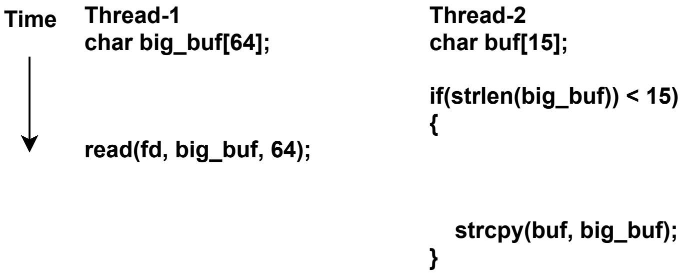
- It is difficult for developers to debug failures in production environment due to limited information (e.g., crashed memory dump).

Concurrency bug diagnosis

What do we require to know to find the root cause of concurrency bug?

Concurrency bug diagnosis

Buffer overflow due to a data race



Time

Thread-1

```
char big_buf[64];



read(fd, big_buf, 64);
```

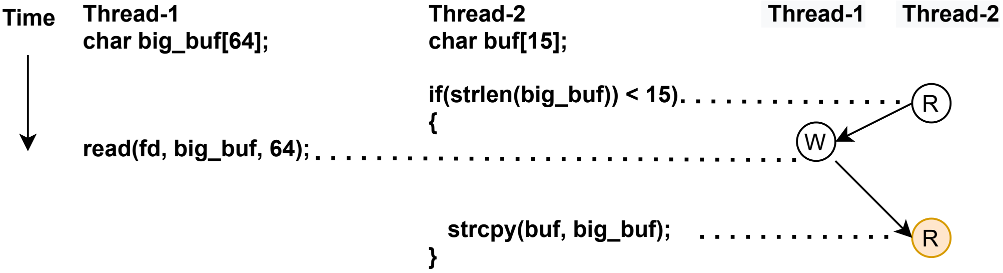
Thread-2

```
char buf[15];

if(strlen(big_buf)) < 15)
{



   strcpy(buf, big_buf);
}
```

Concurrency bug diagnosis

Buffer overflow due to a data race

Atomicity Violation



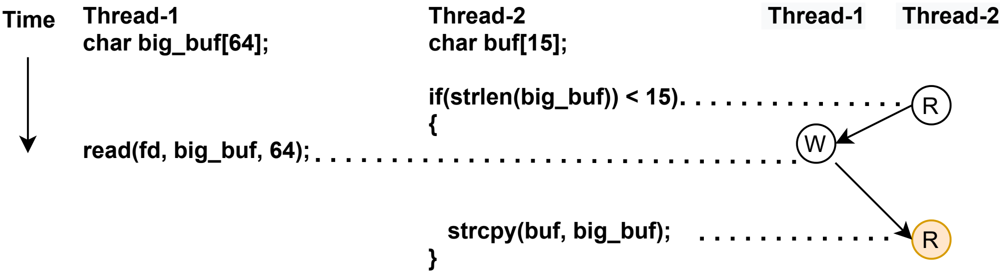
```
Time   Thread-1                       Thread-2
       char big_buf[64];              char buf[15];

  |                                   if(strlen(big_buf)) < 15). . . . . . . .
  |                                   {
  v    read(fd, big_buf, 64);. . . . . . . . . . . . . . . . . . . . . . . . .

                                        strcpy(buf, big_buf);  . . . . . . . .
                                      }
```

- Concurrency bug diagnosis requires knowing the order of data race between thread1 and thread2.

Concurrency bug diagnosis

Buffer overflow due to a data race

Atomicity Violation



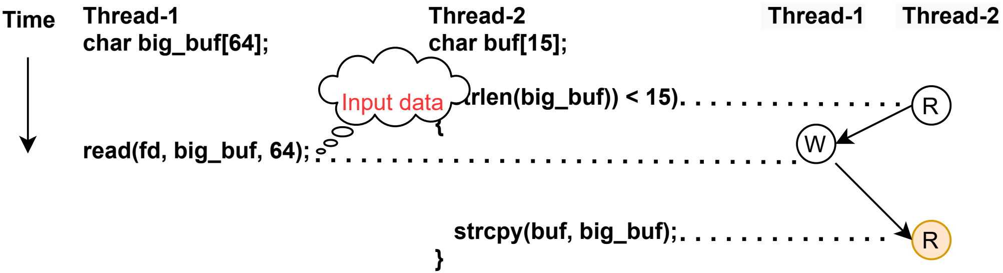
```
Time   Thread-1                      Thread-2
       char big_buf[64];             char buf[15];

                                     if(strlen(big_buf)) < 15). . . . . . . . . . . . . .
                                     {
       read(fd, big_buf, 64); . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

                                       strcpy(buf, big_buf);  . . . . . . . . . . . . .
                                     }
```

- What else is important for this bug diagnosis?

Concurrency bug diagnosis

Buffer overflow due to a data race

Atomicity Violation



```
char big_buf[64];

read(fd, big_buf, 64);
```

```
char buf[15];
trlen(big_buf)) < 15)
{
   strcpy(buf, big_buf);
}
```

- The input is also important for reproducing the bug of buffer overflow.

Challenges of failure diagnosis

Finding the root cause of production failures is important but hard.

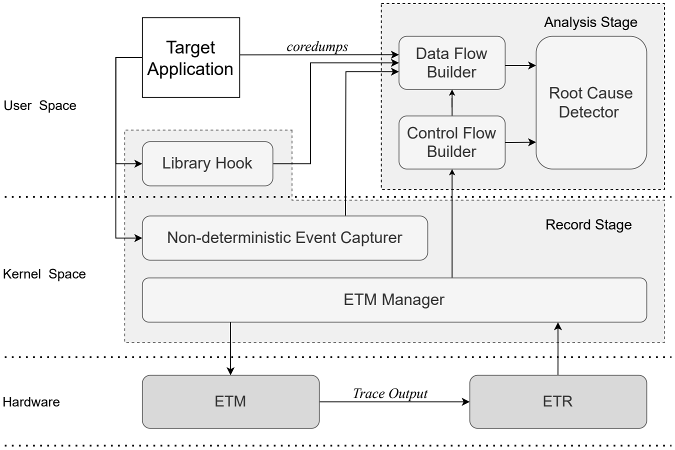
- Exposing bugs in production may be invasive and impractical.
- Tracing fine-grained interleavings of data race incurs high overhead.
- Non-deterministic events such as unforeseen inputs for reproducing bugs may not be available.

Failure Diagnosis

- Investigator on Arm Architecture.

Failure Diagnosis on Arm: Investigator

- Alligator In Vest: Using Hardware Features for Failure Diagnosis



Failure Diagnosis on Arm: Investigator

- Online recording stage
  - Leverage ETM to trace the control flow with timestamps
  - Use a lightweight event capturer to collect non-deterministic events
  - Low runtime overhead

- Offline analysis stage
  - Recover data flow from information collected in recording stage
  - Adaptively improve data flow recovery in analysis stage
  - Indentify root cause using reconstructed control and data flow

Investigator — Key Techniques

- Tackle trace loss
  - Use ETR for ETM trace buffer (up to 4GB)
  - Timely interrupts raised by PMU to save trace to persistent storage without losing the trace output
- A lightweight event capturer
  - Handle the effect of syscall with low overhead
  - Handle library functions avoiding a lot of engineering for applications

Investigator — Key Techniques

- Hardware-assisted adaptive data collection
  - Use hardware watchpoints and breakpoints to achieve high accuracy of data recovery

Investigator — Effectiveness

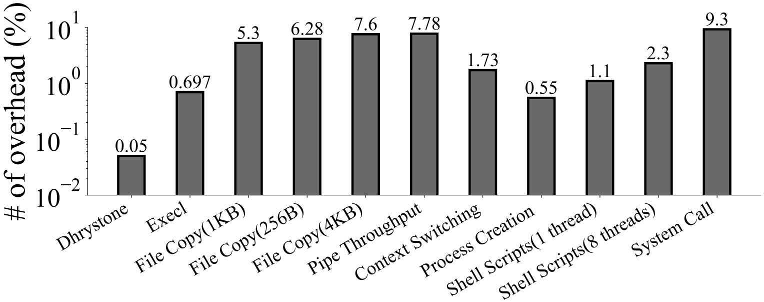
- Testbed Specification
  - ARM Juno v2 development board
  - Real-world C/C++ programs
- Investigator correctly find the root cause of concurrency bugs and sequential bugs.

Table: Partial bugs Evaluated by Investigator

Program-BugID	**bug type**	**LOC**	**Symptom**
SQLite-1672	DL	80K	deadlock
memcached-127	SAV	18K	race condition fault
Python-35185	SAV	1256K	race condition fault
Python-31530	MAV	1256K	segmentation fault
aget-N/A	MAV	2.5K	assertion failure
pbzip2-N/A	OV	2K	use-after-free
curl-965	SEQ	160K	unhandled input pattern
cppcheck-2782	SEQ	120K	unhandled input pattern
cppcheck-3238	SEQ	138K	NULL pointer dereference

Investigator — Efficiency

- Investigator incurs up to 3.88% runtime performance overhead on average in Unxibench benchmark.



Overview of The Talk

- Background: Hardware features on Arm
- Ninja: Towards transparent tracing and debugging
- Investigator: Finding the root cause of concurrency bugs
- COMPASS lab

$cat COMPASS

COMPASS Lab

COMPuter And System Security Lab

- 计算机系统安全实验室

COMPASS Research Interests:

- Hardware-assisted Security
- Transparent Malware Analysis
- Transportation Security

- TEE on Arm/x86/RISC-V
- Arm Debugging Security
- Plausible Deniability encryption

$more COMPASS

$echo Thank you!

Questions?

zhangfw@sustech.edu.cn

http://compass.sustech.edu.cn/
http://cse.sustech.edu.cn/faculty/~zhangfw/