Understanding the Security of ARM Debugging Features

Zhenyu Ning and Fengwei Zhang

COMPASS Lab
Wayne State University

May 21, 2019
Outline

- Introduction
- Obstacles in Traditional Debugging Model
- Nailgun Attack
- Mitigations
- Conclusion
Outline

- Introduction
- Obstacles for Traditional Debugging Model
- Nailgun Attack
- Mitigations
- Conclusion
Modern processors are equipped with hardware-based debugging features to facilitate on-chip debugging process.

- e.g. debug registers, debug exceptions and hardware-based trace.

- It normally requires JTAG [1] connection to make use of these features.
Traditional Debugging

Understanding the Security of ARM Debugging Features, S&P 19
Traditional Debugging

Debug Authentication
Debug Target (TARGET)
Debug Host (HOST)
JTAG Interface

What makes it secure?

Understanding the Security of ARM Debugging Features, S&P 19
Traditional Debugging

Debug Authentication

Debug Target (TARGET)

JTAG Interface

Debug Host (HOST)
Traditional Debugging

Debug Authentication

Debug Target (TARGET)

JTAG Interface

Debug Host (HOST)

What makes it secure?
Introduction

What makes it secure?

▶ Obstacle 1: Physical access.

▶ Obstacle 2: Debug authentication.
What makes it secure?

- **Obstacle 1**: Physical access.
- **Obstacle 2**: Debug authentication.

Do these obstacles work?
Introduction

Obstacles for Traditional Debugging Model

Nailgun Attack

Mitigations

Conclusion
Obstacles for Traditional Debugging Model

It is due to two general assumptions:

- **Obstacle 1**: Physical access.
- **Obstacle 2**: Debug authentication.

Does it really require physical access?
Inter-Processor Debugging

We can use one processor on the chip to debug another one on the same chip, and we refer it as inter-processor debugging.

- Memory-mapped debugging registers.
  - Introduced since ARMv7.

- No JTAG, No physical access.
Obstacles for Traditional Debugging Model

It is due to two general assumptions:

- **Obstacle 1**: Physical access.

- **Obstacle 2**: Debug authentication.

Does debug authentication work as expected?
ARM Debug Authentication

**TARGET**
*(Normal State)*

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Register 1</th>
<th>Register 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>x3,</td>
<td>#3</td>
</tr>
<tr>
<td>MOV</td>
<td>x4,</td>
<td>#4</td>
</tr>
<tr>
<td>MOV</td>
<td>x0,</td>
<td>x3</td>
</tr>
<tr>
<td>MOV</td>
<td>x1,</td>
<td>x4</td>
</tr>
<tr>
<td>LDR</td>
<td>pc,</td>
<td>[pc, #0x10]</td>
</tr>
</tbody>
</table>

TARGET is executing instructions pointed by pc
### ARM Debug Authentication

#### Non-invasive Debugging:
Monitoring without control

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>x3</td>
<td>#3</td>
</tr>
<tr>
<td>MOV</td>
<td>x4</td>
<td>#4</td>
</tr>
<tr>
<td>MOV</td>
<td>x0</td>
<td>x3</td>
</tr>
<tr>
<td>MOV</td>
<td>x1</td>
<td>x4</td>
</tr>
<tr>
<td>LDR</td>
<td>pc</td>
<td>[pc, #0x10]</td>
</tr>
</tbody>
</table>

**TARGET**
(Normal State)
**ARM Debug Authentication**

**TARGET**
(Debug State)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>x3,</td>
</tr>
<tr>
<td>MOV</td>
<td>x4,</td>
</tr>
<tr>
<td>MOV</td>
<td>x0,</td>
</tr>
<tr>
<td>MOV</td>
<td>x1,</td>
</tr>
<tr>
<td>LDR</td>
<td>pc,</td>
</tr>
<tr>
<td></td>
<td>[pc,</td>
</tr>
<tr>
<td></td>
<td>#-0x10]</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**Invasive Debugging:** Control and change status
ARM Debug Authentication

**TARGET**
(Normal State)

<table>
<thead>
<tr>
<th>pc</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>x3, #3</td>
</tr>
<tr>
<td>MOV</td>
<td>x4, #4</td>
</tr>
<tr>
<td>MOV</td>
<td>x0, x3</td>
</tr>
<tr>
<td>MOV</td>
<td>x1, x4</td>
</tr>
<tr>
<td>LDR</td>
<td>pc, [pc, #-0x10]</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**Debug Authentication Signal:** Whether debugging is allowed

Debug Disabled
ARM Debug Authentication

TARGET
(Normal State)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>x3, #3</td>
</tr>
<tr>
<td>MOV</td>
<td>x4, #4</td>
</tr>
<tr>
<td>MOV</td>
<td>x0, x3</td>
</tr>
<tr>
<td>MOV</td>
<td>x1, x4</td>
</tr>
<tr>
<td>LDR</td>
<td>pc, [pc, #-0x10]</td>
</tr>
</tbody>
</table>

Four signals for: Secure/Non-secure, Invasive/Non-invasive

Debug Disabled
ARM licenses technology to the SoC Vendors.
- e.g., ARM architectures and Cortex processors

Defines the debug authentication signals.
ARM Ecosystem

- The SoC Vendors develop chips for the OEMs.
  - e.g., Qualcomm Snapdragon SoCs
- Implement the debug authentication signals.
ARM Ecosystem

- The OEMs produce devices for the users.
  - e.g., Samsung Galaxy Series and Huawei Mate Series

- Configure the debug authentication signals.
ARM Ecosystem

- Finally, the User can enjoy the released devices.
  - Tablets, smartphones, and other devices
- Learn the status debug authentication signals.
Debug Authentication Signals

- What is the status of the signals in real-world device?
- How to manage the signals in real-world device?
## Debug Authentication Signals

### Table: Debug Authentication Signals on Real Devices.

<table>
<thead>
<tr>
<th>Category</th>
<th>Platform / Device</th>
<th>DBGEN</th>
<th>NIDEN</th>
<th>SPIDEN</th>
<th>SPNIDEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Boards</td>
<td>ARM Juno r1 Board</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>NXP i.MX53 QSB</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>IoT Devices</td>
<td>Raspberry PI 3 B+</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cloud Platforms</td>
<td>64-bit ARM miniNode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Packet Type 2A Server</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Scaleway ARM C1 Server</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mobile Devices</td>
<td>Google Nexus 6</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>Samsung Galaxy Note 2</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>Huawei Mate 7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Motorola E4 Plus</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Xiaomi Redmi 6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
# Debug Authentication Signals

Table: Debug Authentication Signals on Real Devices.

<table>
<thead>
<tr>
<th>Category</th>
<th>Platform / Device</th>
<th>Debug Authentication Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DBGEN</td>
</tr>
<tr>
<td>Development Boards</td>
<td>ARM Juno r1 Board</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>NXP i.MX53 QSB</td>
<td>✗</td>
</tr>
<tr>
<td>IoT Devices</td>
<td>Raspberry PI 3 B+</td>
<td>✓</td>
</tr>
<tr>
<td>Cloud Platforms</td>
<td>64-bit ARM miniNode</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Packet Type 2A Server</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Scaleway ARM C1 Server</td>
<td>✓</td>
</tr>
<tr>
<td>Mobile Devices</td>
<td>Google Nexus 6</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>Samsung Galaxy Note 2</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Huawei Mate 7</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Motorola E4 Plus</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Xiaomi Redmi 6</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Debug Authentication Signals

**Table:** Debug Authentication Signals on Real Devices.

<table>
<thead>
<tr>
<th>Category</th>
<th>Platform / Device</th>
<th>DBGEN</th>
<th>NIDEN</th>
<th>SPIDEN</th>
<th>SPNIDEN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Development Boards</strong></td>
<td>ARM Juno r1 Board</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>NXP i.MX53 QSB</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td><strong>IoT Devices</strong></td>
<td>Raspberry PI 3 B+</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td><strong>Cloud Platforms</strong></td>
<td>64-bit ARM miniNode</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Packet Type 2A Server</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Scaleway ARM C1 Server</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td><strong>Mobile Devices</strong></td>
<td>Google Nexus 6</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>Samsung Galaxy Note 2</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>Huawei Mate 7</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Motorola E4 Plus</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Xiaomi Redmi 6</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Debug Authentication Signals

How to manage the signals in real-world device?

- For both development boards with manual, we cannot fully control the debug authentication signals.
  - Signals in i.MX53 QSB can be enabled by JTAG.
  - The DBGEN and NIDEN in ARM Juno board cannot be disabled.

- In some mobile phones, we find that the signals are controlled by One-Time Programmable (OTP) fuse.

For all the other devices, nothing is publicly available.
Obstacles for Traditional Debugging Model

To summarize,

▶ We don’t need physical access to debug a processor.

▶ The debug authentication also allows us to debug the processor.
Outline

- Introduction
- Obstacles for Traditional Debugging Model
- Nailgun Attack
- Mitigations
- Conclusion
Nailgun Attack

Memory-mapped Interface

Debug Target (TARGET)

Debug Host (HOST)
Nailgun Attack

Understanding the Security of ARM Debugging Features, S&P 19
Nailgun Attack

A Multi-processor SoC System

High-privilege Resource
(Secure RAM/Register/Peripheral)

TARGET

HOST

Low-privilege Resource
(Non-Secure RAM/Register/Peripheral)

An example SoC system:

- Two processors as HOST and TARGET, respectively.
- Low-privilege and High-privilege resource.
Nailgun Attack

A Multi-processor SoC System

High-privilege Resource
(Secure RAM/Register/Peripheral)

Low-privilege Resource
(Non-Secure RAM/Register/Peripheral)

TARGET
HOST

- Low-privilege refers to non-secure kernel-level privilege
- High-privilege refers to any other higher privilege
Nailgun Attack

A Multi-processor SoC System

- High-privilege Resource (Secure RAM/Register/Peripheral)
- Low-privilege Resource (Non-Secure RAM/Register/Peripheral)

TARGET (Normal State) (Low Privilege)
HOST (Normal State) (Low Privilege)

Both processors are only access low-privilege resource.
- Normal state
- Low-privilege mode

Understanding the Security of ARM Debugging Features, S&P 19
HOST sends a **Debug Request** to TARGET,

- TARGET checks its authentication signal.
- Privilege of HOST is ignored.
TARGET turns to **Debug State** according to the request.

- Low-privilege mode
- No access to high-privilege resource
HOST sends a **Privilege Escalation Request** to TARGET,

- e.g., executing DCPS series instructions.
- The instructions can be executed at any privilege level.
Nailgun Attack

A Multi-processor SoC System

TARGET
(Debug State)
(High Privilege)

HOST
(Normal State)
(Low Privilege)

High-privilege Resource
(Secure RAM/Register/Peripheral)

Low-privilege Resource
(Non-Secure RAM/Register/Peripheral)

TARGET turns to **High-privilege Mode** according to the request.

- Debug state, high-privilege mode
- Gained access to high-privilege resource
HOST sends a **Resource Access Request** to TARGET,

- e.g., accessing secure RAM/register/peripheral.
- Privilege of HOST is ignored.
Nailgun Attack

A Multi-processor SoC System

TARGET
(Old State)
(High Privilege)

HOST
(Normal State)
(Low Privilege)

High-privilege Resource
(Secure RAM/Register/Peripheral)

Low-privilege Resource
(Non-Secure RAM/Register/Peripheral)

Debug Response

TARGET return the result to HOST,
➤ i.e., content of the high-privilege resource.
➤ Privilege of HOST is ignored.
HOST gains access to the high-privilege resource while running in,

- Normal state
- Low-privilege mode
Nailgun Attack

Nailgun: Break the privilege isolation of ARM platform.

- Achieve access to high-privilege resource via misusing the ARM debugging features.

- Can be used to craft different attacks.
  - Inferring encryption keys
  - Arbitrary payload execution in TrustZone
Nailgun Attack

Fingerprint extraction in commercial mobile phone.

- Device: Huawei Mate 7 (MT-L09)
- Firmware: MT7-L09V100R001C00B121SP05
- Fingerprint sensor: FPC1020
Nailgun Attack

- By reverse engineering, we learn the address to store fingerprint data.

- With Nailgun, we extract the fingerprint data from secure world with a non-secure kernel module.

- Finally, the fingerprint image is reconstructed from the data with help of the publicly available sensor manual.
Nailgun Attack

- The right part of the image is blurred for privacy concerns.
- Source code: https://compass.cs.wayne.edu/nailgun/
Outline

▶ Introduction
▶ Obstacles for Traditional Debugging Model
▶ Nailgun Attack
▶ Mitigations
▶ Conclusion
Mitigations

Simply disable the signals?
Mitigations

Simply disable the authentication signals?

▶ Existing tools rely on the debug authentication signals.
  - e.g., [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

▶ Unavailable management mechanisms.

▶ OTP feature, cost, and maintenance.
Mitigations

We suggest a comprehensive defense across different roles in the ARM ecosystem.

- For ARM, additional restriction in inter-processor debugging model.
- For SoC vendors, refined signal management and hardware-assisted access control to debug components.
- For OEMs and cloud providers, software-based access control.
Outline

▶ Introduction
▶ Obstacles for Traditional Debugging Model
▶ Nailgun Attack
▶ Mitigations
▶ Conclusion
Conclusion

- We present a study on the security of hardware debugging features on ARM platform.
- It shows that the "known-safe" or "assumed-safe" component in the legacy systems turns to be vulnerable while advanced systems are deployed.
- We suggest a comprehensive rethink on the security of legacy mechanisms.
References


Thank you!

Questions?

zhenyu.ning@wayne.edu

http://compass.cs.wayne.edu
Backup Slides
Nailgun in different ARM architecture

- **64-bit ARMv8 architecture**: ARM Juno r1 board.
  - Embedded Cross Trigger (ECT) for debug request.
  - Binary instruction to Instruction Transfer Register (ITR).

- **32-bit ARMv8 architecture**: Raspberry PI Model 3 B+.
  - Embedded Cross Trigger (ECT) for debug request.
  - First and last half of binary instruction should be reversed in ITR.

- **ARMv7 architecture**: Huawei Mate 7.
  - Use Debug Run Control Register for debug request.
  - Binary instruction to Instruction Transfer Register (ITR).
Instruction Execution in Debug State

TARGET
(Normal State)

<table>
<thead>
<tr>
<th>pc</th>
<th>MOV</th>
<th>x3,  #3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOV</td>
<td>x4,  #4</td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>x0,  x3</td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>x1,  x4</td>
</tr>
<tr>
<td></td>
<td>LDR</td>
<td>pc, [pc, #-0x10]</td>
</tr>
</tbody>
</table>

In normal state, TARGET is executing instructions pointed by pc
Instruction Execution in Debug State

In debug state, TARGET stops executing the instruction at pc.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Register/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>x3, #3</td>
</tr>
<tr>
<td>MOV</td>
<td>x4, #4</td>
</tr>
<tr>
<td>MOV</td>
<td>x0, x3</td>
</tr>
<tr>
<td>MOV</td>
<td>x1, x4</td>
</tr>
<tr>
<td>LDR</td>
<td>pc, [pc, #-0x10]</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
### Instruction Execution in Debug State

**TARGET**  
(Debug State)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>x3,</td>
<td>#3</td>
</tr>
<tr>
<td>MOV</td>
<td>x4,</td>
<td>#4</td>
</tr>
<tr>
<td>MOV</td>
<td>x0,</td>
<td>x3</td>
</tr>
<tr>
<td>MOV</td>
<td>x1,</td>
<td>x4</td>
</tr>
<tr>
<td>LDR</td>
<td>pc,</td>
<td>[pc, #-0x10]</td>
</tr>
</tbody>
</table>

**ITR**  
<table>
<thead>
<tr>
<th>Binary Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV x4, #0</td>
</tr>
</tbody>
</table>

In debug state, write binary instruction to ITR for execution.
Instruction Execution in Debug State

In debug state, write binary instruction to ITR for execution.

TARGET
(Debug State)

<table>
<thead>
<tr>
<th>MOV</th>
<th>x3, #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>x4, #4</td>
</tr>
<tr>
<td>MOV</td>
<td>x0, x3</td>
</tr>
<tr>
<td>MOV</td>
<td>x1, x4</td>
</tr>
<tr>
<td>LDR</td>
<td>pc, [pc, #-0x10]</td>
</tr>
</tbody>
</table>

ITR
(Binary Instruction)

MOV   x4, #0

0xB20003E4

In debug state, write binary instruction to ITR for execution.
Instruction Execution in Debug State

In debug state, write binary instruction to ITR for execution

<table>
<thead>
<tr>
<th>pc</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>x3, #3</td>
</tr>
<tr>
<td>MOV</td>
<td>x4, #4</td>
</tr>
<tr>
<td>MOV</td>
<td>x0, x3</td>
</tr>
<tr>
<td>MOV</td>
<td>x1, x4</td>
</tr>
<tr>
<td>LDR</td>
<td>pc, [pc, #-0x10]</td>
</tr>
</tbody>
</table>

**TARGET**
(Debug State)

**ITR**

```
0xB20003E4
MOV x4, #0
```

```
0xB20003E4
```
Disclosure

- March 2018: Preliminary findings are reported to ARM.
- August 2018: Report to ARM with enriched result.
- August 2018: Report our findings to related OEMs.
- October 2018: Issue is reported to MITRE.
- February 2019: PoCs and demos are released.
- April 2019: CVE-2018-18068 is released.