

# Understanding the Security of ARM Debugging Features

**Zhenyu Ning** and Fengwei Zhang

COMPASS Lab  
Wayne State University

May 21, 2019

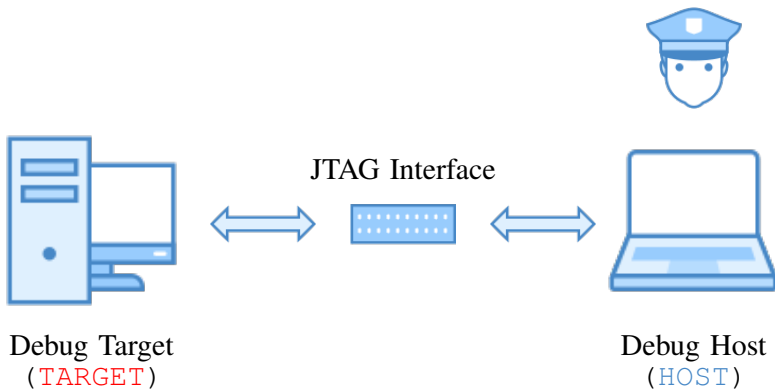
- ▶ Introduction
- ▶ Obstacles in Traditional Debugging Model
- ▶ Nailgun Attack
- ▶ Mitigations
- ▶ Conclusion

- ▶ [Introduction](#)
- ▶ Obstacles for Traditional Debugging Model
- ▶ Nailgun Attack
- ▶ Mitigations
- ▶ Conclusion

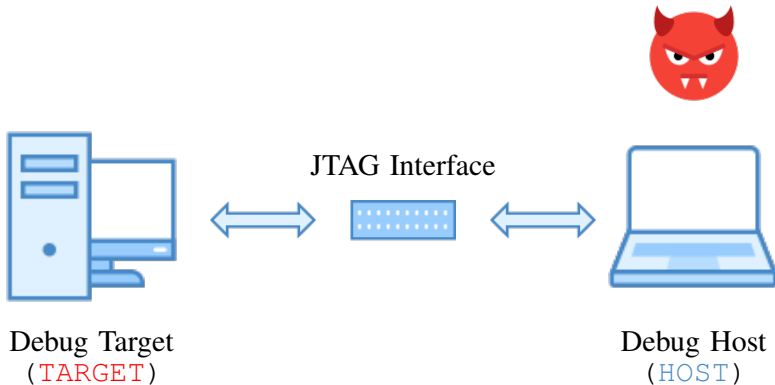
Modern processors are equipped with hardware-based debugging features to facilitate on-chip debugging process.

- e.g. debug registers, debug exceptions and hardware-based trace.
- It normally requires JTAG [1] connection to make use of these features.

# Traditional Debugging



# Traditional Debugging



# Traditional Debugging

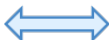


Debug  
Authentication



Debug Target  
(**TARGET**)

JTAG Interface



Debug Host  
(**HOST**)

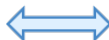


Debug  
Authentication



Debug Target  
(**TARGET**)

JTAG Interface



Debug Host  
(**HOST**)

## What makes it **secure**?



## 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?

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 is executing instructions pointed by pc

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



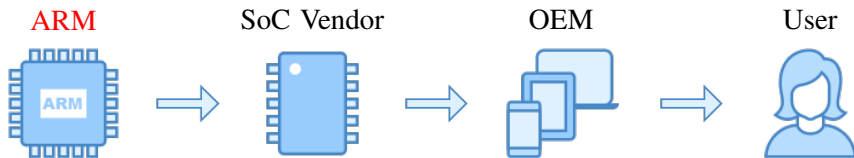
**Invasive Debugging:** Control and change status

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

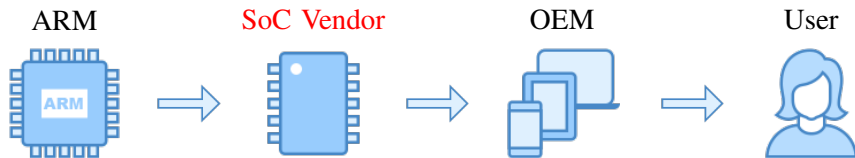
# ARM Debug Authentication



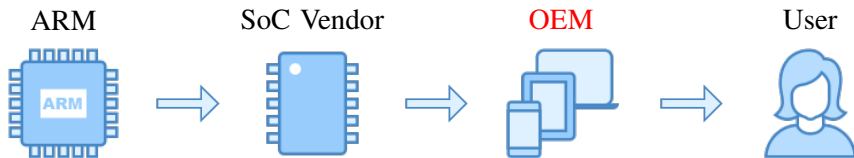
Four signals for: Secure/Non-secure, Invasive/Non-invasive



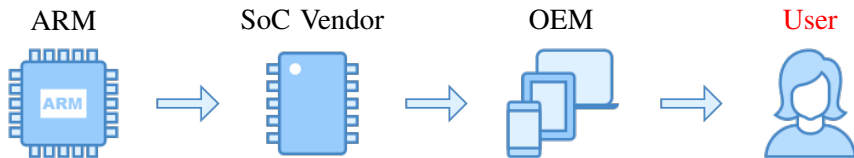
- ▶ ARM licenses technology to the SoC Vendors.
  - e.g., ARM architectures and Cortex processors
- ▶ Defines the debug authentication signals.



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



- ▶ The OEMs produce devices for the users.
  - e.g., Samsung Galaxy Series and Huawei Mate Series
- ▶ Configure the debug authentication signals.



- ▶ 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.

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

# Debug Authentication Signals

Table: Debug Authentication Signals on Real Devices.

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

# Debug Authentication Signals

Table: Debug Authentication Signals on Real Devices.

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

## 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.

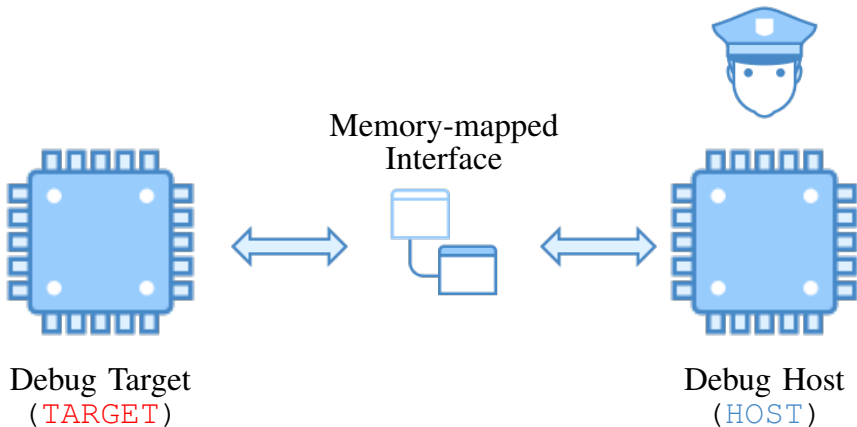
# Obstacles for Traditional Debugging Model

To summarize,

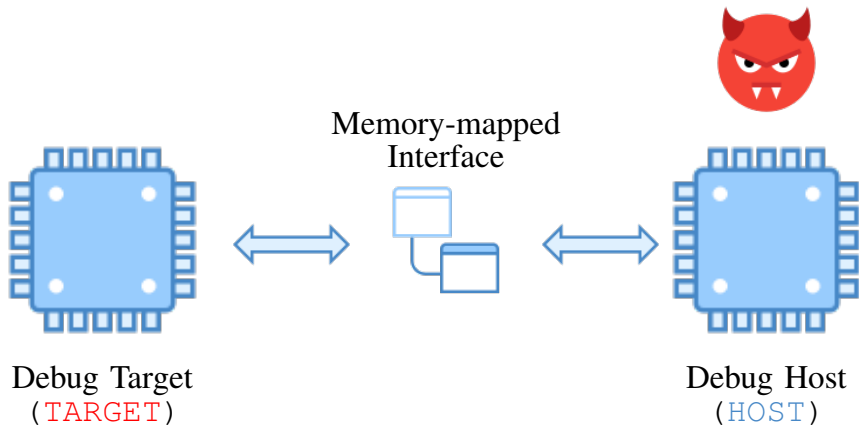
- ▶ We don't need physical access to debug a processor.
- ▶ The debug authentication also allows us to debug the processor.

- ▶ Introduction
- ▶ Obstacles for Traditional Debugging Model
- ▶ [Nailgun Attack](#)
- ▶ Mitigations
- ▶ Conclusion

# Nailgun Attack

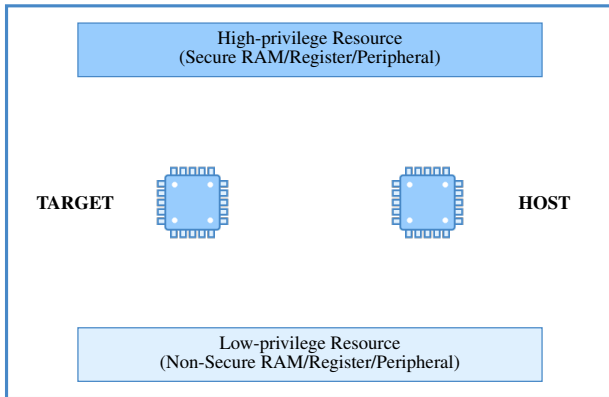


# Nailgun Attack





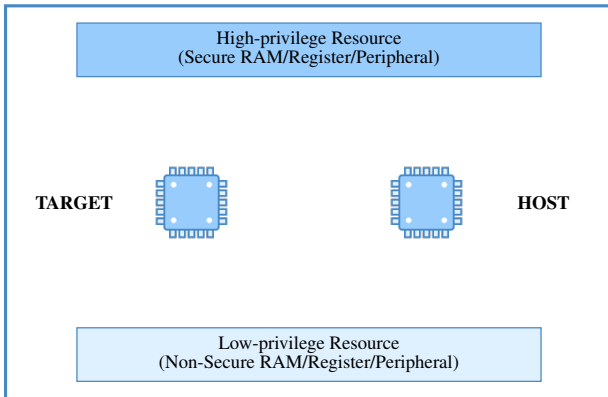
## A Multi-processor SoC System



An example SoC system:

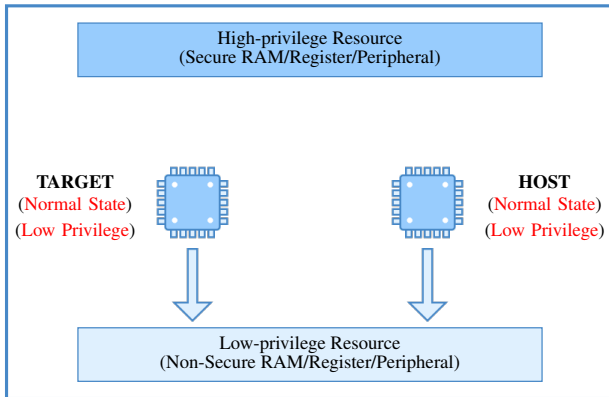
- ▶ Two processors as **HOST** and **TARGET**, respectively.
- ▶ Low-privilege and High-privilege resource.

## A Multi-processor SoC System



- ▶ Low-privilege refers to non-secure **kernel-level** privilege
- ▶ High-privilege refers to any other higher privilege

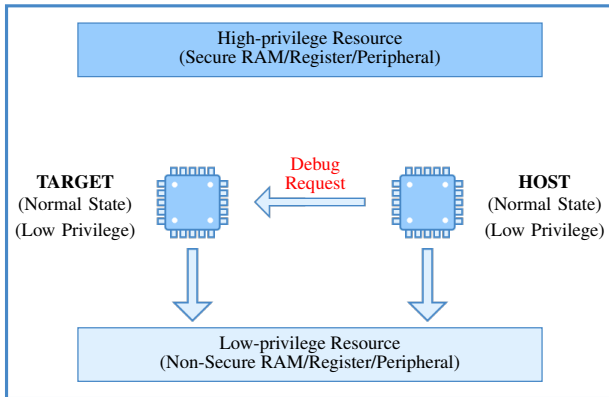
## A Multi-processor SoC System



Both processors are only access low-privilege resource.

- ▶ Normal state
- ▶ Low-privilege mode

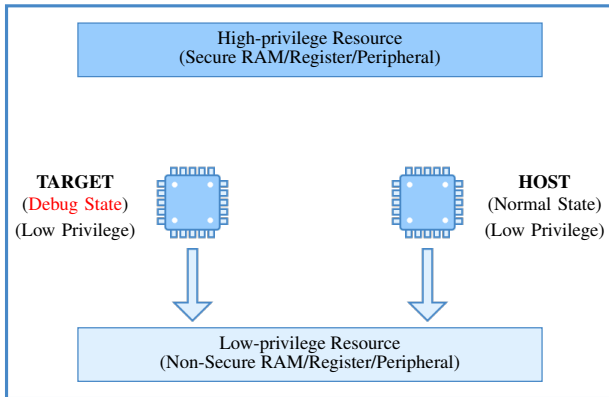
## A Multi-processor SoC System



HOST sends a **Debug Request** to TARGET,

- ▶ TARGET checks its authentication signal.
- ▶ Privilege of HOST is ignored.

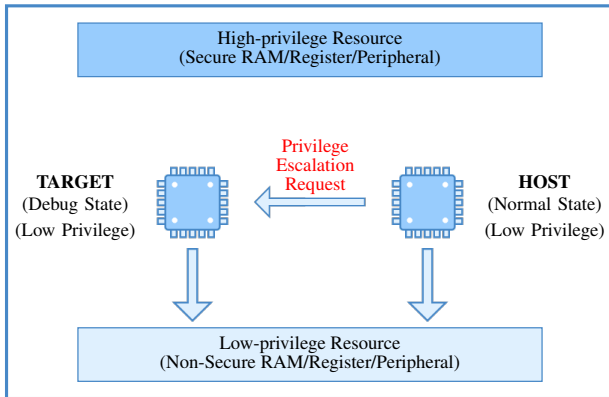
## A Multi-processor SoC System



TARGET turns to **Debug State** according to the request.

- ▶ Low-privilege mode
- ▶ No access to high-privilege resource

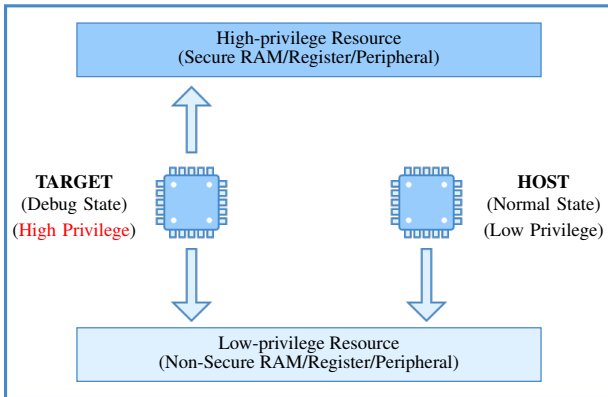
## A Multi-processor SoC System



HOST sends a **Privilege Escalation Request** to TARGET,

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

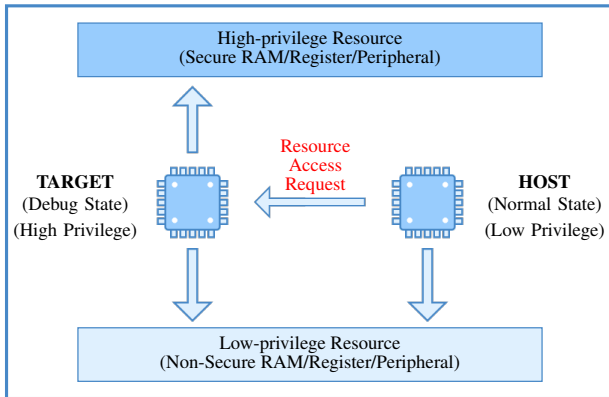
## A Multi-processor SoC System



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

- ▶ Debug state, high-privilege mode
- ▶ Gained access to high-privilege resource

## A Multi-processor SoC System

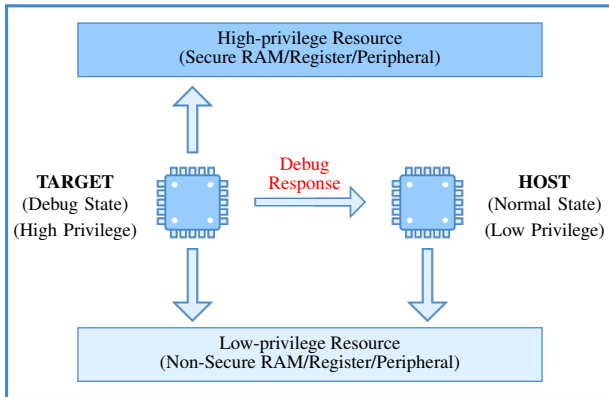


HOST sends a **Resource Access Request** to TARGET,

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



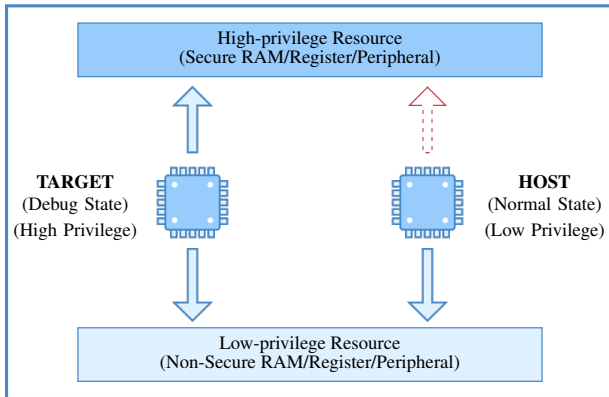
## A Multi-processor SoC System



TARGET return the result to HOST,

- ▶ i.e., content of the high-privilege resource.
- ▶ Privilege of HOST is ignored.

## A Multi-processor SoC System



HOST gains access to the high-privilege resource while running in,

- ▶ Normal state
- ▶ Low-privilege mode

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

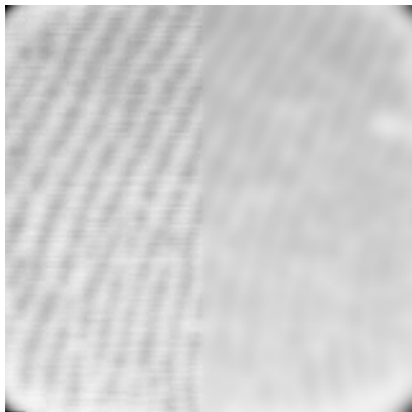
Fingerprint extraction in commercial mobile phone.

- ▶ 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/>

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

Simply **disable** the signals?



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.

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.

- ▶ Introduction
- ▶ Obstacles for Traditional Debugging Model
- ▶ Nailgun Attack
- ▶ Mitigations
- ▶ [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.

- [1] IEEE, "Standard for test access port and boundary-scan architecture," <https://standards.ieee.org/findstds/standard/1149.1-2013.html>.
- [2] D. Balzarotti, G. Banks, M. Cova, V. Felmetzger, R. Kemmerer, W. Robertson, F. Valeur, and G. Vigna, "An experience in testing the security of real-world electronic voting systems," *IEEE Transactions on Software Engineering*, 2010.
- [3] S. Clark, T. Goodspeed, P. Metzger, Z. Wasserman, K. Xu, and M. Blaze, "Why (special agent) johnny (still) can't encrypt: A security analysis of the APCO project 25 two-way radio system," in *Proceedings of the 20th USENIX Security Symposium (USENIX Security'11)*, 2011.
- [4] L. Cojocar, K. Razavi, and H. Bos, "Off-the-shelf embedded devices as platforms for security research," in *Proceedings of the 10th European Workshop on Systems Security (EuroSec'17)*, 2017.
- [5] N. Corteggiani, G. Camurati, and A. Francillon, "Inception: System-wide security testing of real-world embedded systems software," in *Proceedings of the 27th USENIX Security Symposium (USENIX Security'18)*, 2018.
- [6] L. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. A. Mohammed, and S. A. Zonouz, "Hey, my malware knows physics! Attacking PLCs with physical model aware rootkit," in *Proceedings of 24th Network and Distributed System Security Symposium (NDSS'17)*, 2017.
- [7] K. Koscher, T. Kohno, and D. Molnar, "SURROGATES: Enabling near-real-time dynamic analyses of embedded systems," in *Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT'15)*, 2015.
- [8] Y. Lee, I. Heo, D. Hwang, K. Kim, and Y. Paek, "Towards a practical solution to detect code reuse attacks on ARM mobile devices," in *Proceedings of the 4th Workshop on Hardware and Architectural Support for Security and Privacy (HASP'15)*, 2015.
- [9] S. Mazloom, M. Rezaeirad, A. Hunter, and D. McCoy, "A security analysis of an in-vehicle infotainment and app platform," in *Proceedings of the 10th USENIX Workshop on Offensive Technologies (WOOT'16)*, 2016.

# References II

- [10] Z. Ning and F. Zhang, "Ninja: Towards transparent tracing and debugging on ARM," in *Proceedings of the 26th USENIX Security Symposium (USENIX Security'17)*, 2017.
- [11] J. Zaddach, L. Bruno, A. Francillon, D. Balzarotti *et al.*, "AVATAR: A framework to support dynamic security analysis of embedded systems' firmwares," in *Proceedings of 21st Network and Distributed System Security Symposium (NDSS'14)*, 2014.

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



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

# Instruction Execution in Debug State



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

# Instruction Execution in Debug State



In debug state, write binary instruction to ITR for execution

- ▶ 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.