2024 Collegiate eCTF Award Ceremony

#eCTF2024

2024 Collegiate eCTF Award Ceremony

WELCOME!

WE WILL GET STARTED SHORTLY

#eCTF2024

#eCTF2024

WELCOME

2024 ECTF AWARD CEREMONY

Ben Janis

Senior Embedded Security Engineer

eCTF Technical Lead

MITRE

MITRE ENGENUITY

Today's Agenda

#eCTF2024

- 9:45 Registration / Breakfast
- 10:15 Welcome from eCTF and MITRE
- 10:20 A Word from NEMC
- 10:35 Competition Briefing
- 10:45 Team Presentations
- **11:15 BREAK**
- 11:25 A Word from NSTXL
- 11:35 Team Presentations
- **12:20 LUNCH / NETWORK**
- 1:20 Team Presentations
- 1:50 A Word from Fortinet
- 2:05 Award Presentation
- 2:20 Closing Remarks / Student Dismissal

#eCTF2024

WELCOME

2024 ECTF AWARD CEREMONY

Moise Solomon

Director

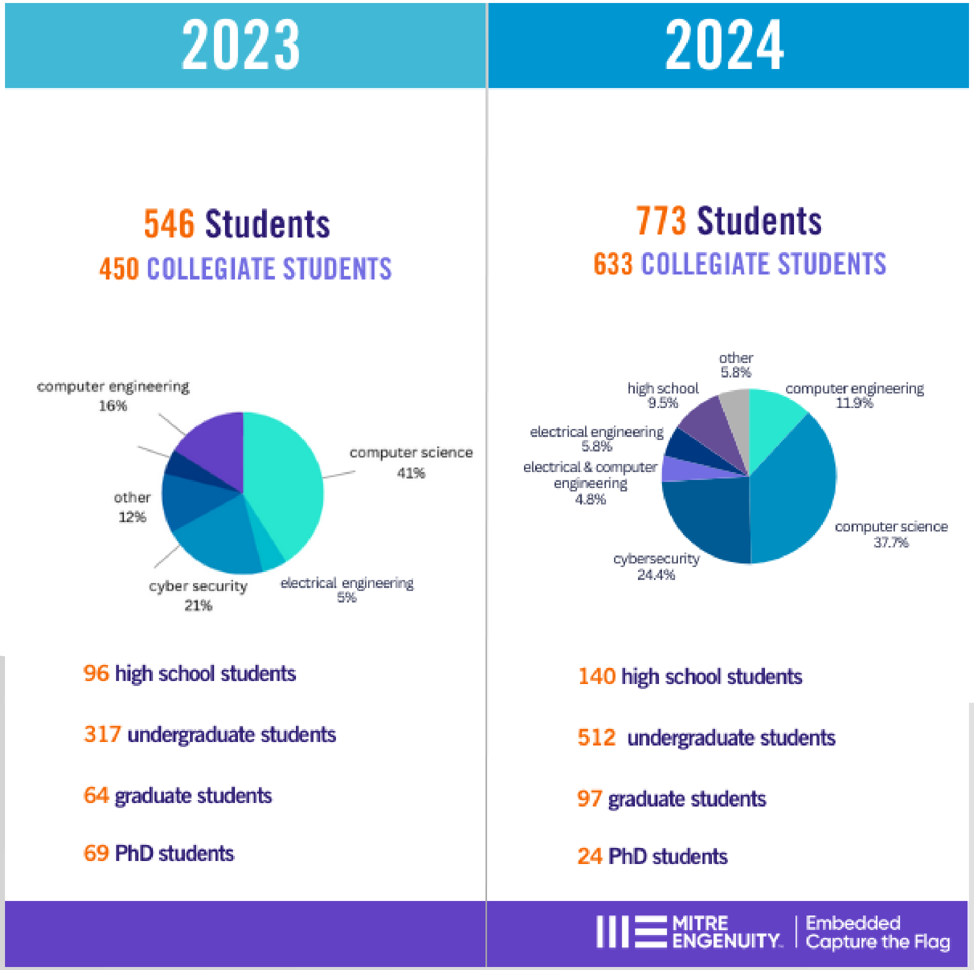
Electronic Systems Innovation Center

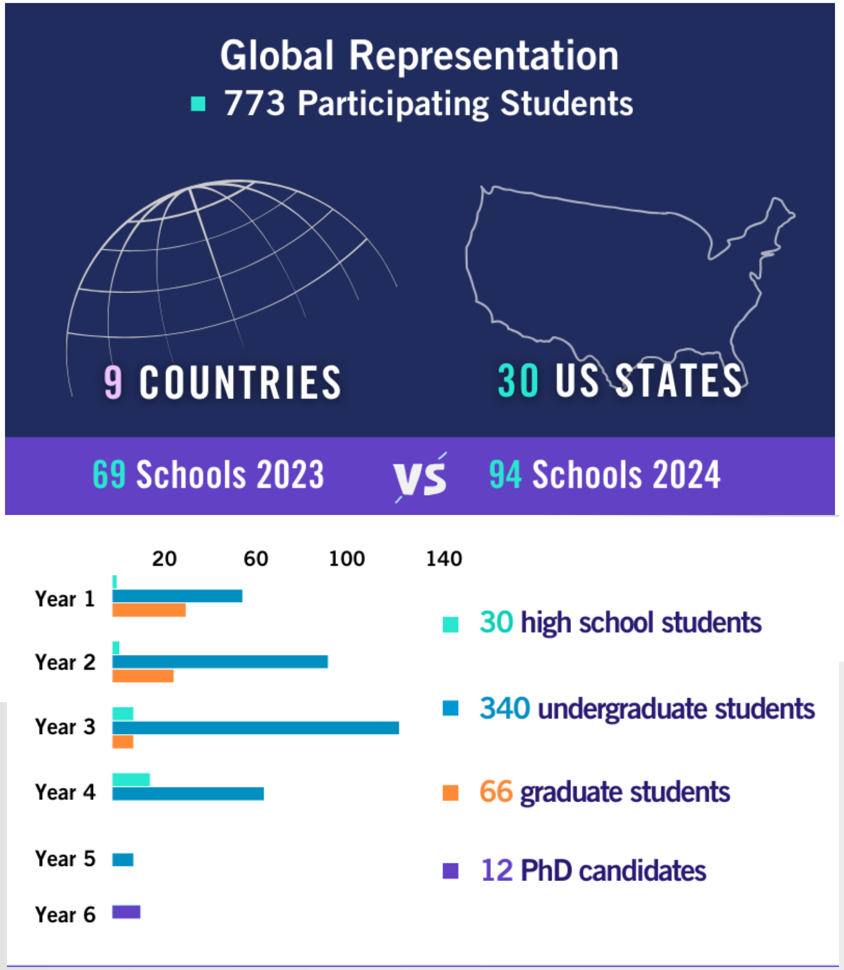
MITRE

MITRE ENGENUITY

Working to Close the U.S. Embedded & Cybersecurity Workforce Gap

#eCTF2024





MITRE MITRE ENGENUITY

#eCTF2024

PRESS RELEASE

Healey-Driscoll Administration Announces $9.2 Million to Boost Microelectronics During U.S. Department of Defense Visit

Includes $7.7 Million for New Technology at MIT and $1.5 Million to Boost Workforce Development, Education, & Student Engagement across Northeast Region

- **MITRE, Bedford, Mass.** - An award of $750,000 to expand the Embedded Capture-the-Flag (eCTF) competition, which aims to attract students and develop their skills in secure microelectronics. The program leverages gamification to bridge the educational gap in embedded systems security and microelectronics, to prepare students to work in this critical field. The eCTF program is designed as a hands-on, project-based learning experience that caters to participants of various skill levels. The program will be aimed at high school, community college, undergraduate, and graduate students, with a focus on underrepresented groups within the industry.

THANK YOU

NEMC HUB

MITRE MITRE ENGENUITY

Thank You, Sponsors!

DIAMOND SPONSOR

FORTINET

GOLD SPONSOR

CROWDSTRIKE

SUPPORTER SPONSOR

IDAHO SCIENTIFIC

SILVER SPONSOR

BCI

HARDWARE DONOR

ANALOG DEVICES

#eCTF2024

Welcome

Ben Linville-Engler

Deputy Director

Chief Investment Strategist

NEMC HUB

MITRE ENGENUITY

MASSACHUSETTS TECHNOLOGY COLLABORATIVE

OUR MISSION: We strengthen the competitiveness of the tech and innovation economy by driving strategic investments, partnerships, and insights that harness the talent of Massachusetts.

THE INNOVATION INSTITUTE

MASSACHUSETTS CENTER for ADVANCED MANUFACTURING

MBI MASSACHUSETTS BROADBAND INSTITUTE

MeHI MASSACHUSETTS eHEALTH INSTITUTE

MassCyberCenter

NEMC Northeast Microelectronics Coalition HUB

Administering ~$530M in state and federal funds through the next 2 fiscal years.

Established in 1982 by Legislative Statute

NEMC
Northeast Microelectronics Coalition
HUB

- Number of Hub Members: 170 organizations
- Number of Hub Participants: 575+ individuals
- Advisory Group: Applied Materials, ADI, BAE Systems, Columbia, MIT, MIT LL, **MITRE**, NextFlex, and Raytheon
- Year 1 NEMC Hub Funding: $19.67M
- Total Estimated Hub Value:
  - $40M+ private/3rd party contribution
  - $40M MA state match
  - $65M MA capital grants
  - $1B+ regional assets

5G/6G | AI Hardware | Commercial Leap Ahead | Electromag. Warfare | Secure Edge and IoT | Quantum

Thank You!

Let's #getCHIPSdone!

NEMC

Northeast Microelectronics Coalition

HUB

#eCTF2024

COMPETITION BRIEFING & HIGHLIGHTS

Fritz Stine

Senior Cyber Operations Engineer

Thank You, Participants!

#eCTF2024

University of Connecticut	Florida Atlantic University	Michigan Technological University	Center I (Albemarle County Public Schools)	Michigan State University	Embry-Riddle Aeronautical University, Prescott	West Virginia University	**Colombe academy of Technology**	Veermata Jijabai Technological Institute	Purdue University
United States Coast Guard Academy	University of California, Irvine	The University of Tulsa	Summit Technology Academy	Duke University	Shawnee Mission Center for Academic Achievement	California State University, Los Angeles	Baldwin Wallace University	Munster High School	Amrita Vishwa Vidyapeetham University
United States Air Force Academy	University at Buffalo	Dakota State University	Worcester Polytechnic Institute	University of California, Santa Barbara	University of Colorado, Colorado Springs	Carnegie Mellon University	New Century Technology High School	Ivy Tech Community College Valparaiso	Chennai Institute Of Technology
Indian Institute of Technology Madras	University of Texas at Arlington	University of Arizona	Virginia Tech	Ipnet Institute of Technology	Singapore Management University 1	University of Port Harcourt	Saddleback College	University of Florida	Columbia University
Government Polytechnic, Pendurthi	University of Illinois Urbana-Champaign	Tufts University	University of New Haven	Pace University	Singapore Management University 2	Massachusetts Institute of Technology	Lenoir Community College	North West Arkansas Community College	Ecole 2600
University of California Los Angeles	ISD 196	Oklahoma Christian University	Kansas State University	San Francisco State University	Air Force Institute of Technology	Tennessee Tech University	Utica University	Indian Institute of Technology Dharwad	Northeastern University
Texas A&M University	Indiana Institute of Technology	University of North Dakota	Ecole Royale de l'Air	Xavier University	United States Cyber Games	University of Maryland, Baltimore County	Norfolk State University	Lakota East High School	The Citadel
Georgia Institute of Technology	University of Washington	Parkway Spark!	Strayer University	Florida International University	St. Lawrence University	Indian Institute of Technology Indore	Thomas Jefferson High School for Science and Technology	Lakota West High School	The University of Alabama
Springfield-Clark County Career Technology Center	University of California, Santa Cruz	LJ College of Computer Applications	Northern Virginia Community College	Kilgore College	Andrada Polytechnic High School	CyberAegis	Sathyabama University of Science and Technology	East Tennessee State University	
North Carolina State University	University of Nebraska Omaha	Delaware Area Career Center	The Ohio State University	City College of San Francisco	Symbiosis Institute of Technology	Morgan State University			

Key:	New Participant	2023 Champion	High School

Unique Competition Design

#eCTF2024

Focus on **Embedded**

Physical hardware opens scope to physical and proximal attacks

Attack **and** Defend

Students wear both hats by acting as both red team and blue team

Extended Time

Semester-long competition opens door to advanced attacks and countermeasures

MITRE
SOLVING PROBLEMS FOR A SAFER WORLD

MITRE ENGENUITY
A Foundation for Public Good

Competition Overview

#eCTF2024

Design Phase

Teams design and implement systems that meets security and functionality requirements

Handoff

Organizers test each design for functionality

Attack Phase

Teams analyze and attack each other's designs for points

Jan 18 — eCTF Kickoff

Mar 1 — Attack Phase Begins

Apr 19 — Attack Phase Ends

Apr 26 — Award Ceremony

MITRE MITRE ENGENUITY

What Teams are Given

#eCTF2024

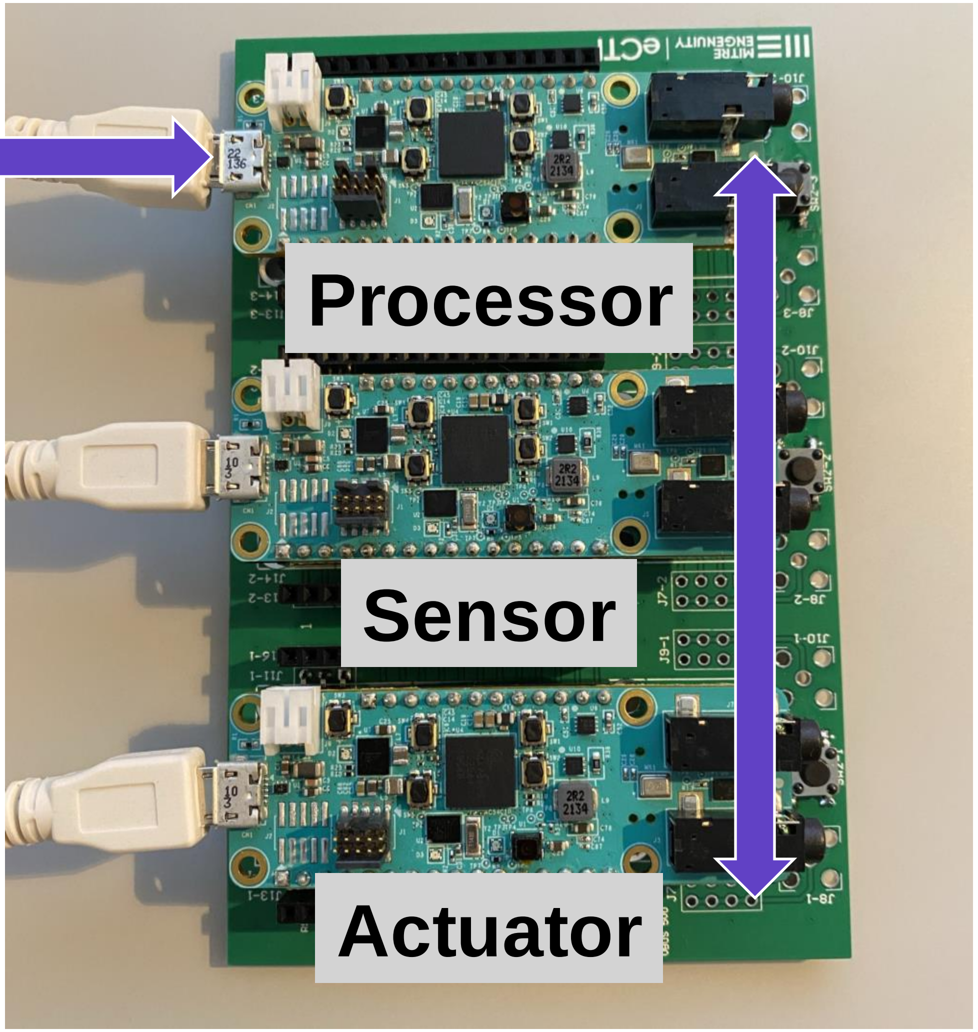
- Functional Requirements
- Hardware
- Automated Testing
- Security Requirements
- Example Code (Reference Design)
- Organizer Support

MITRE | MITRE ENGENUITY

Medical Infrastructure Supply Chain Security

#eCTF2024

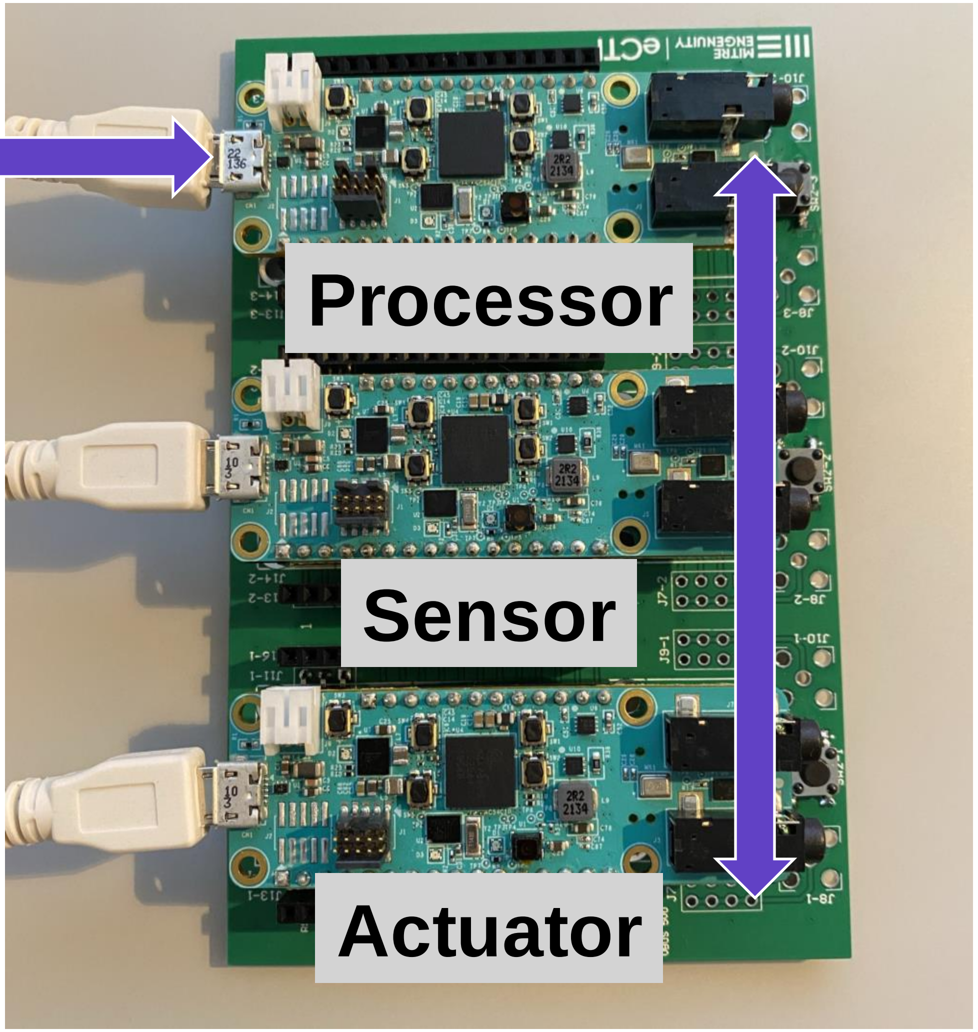
Host Computer



Medical Infrastructure Supply Chain Security

#eCTF2024

Host Computer



Component Authenticity

Communication Integrity

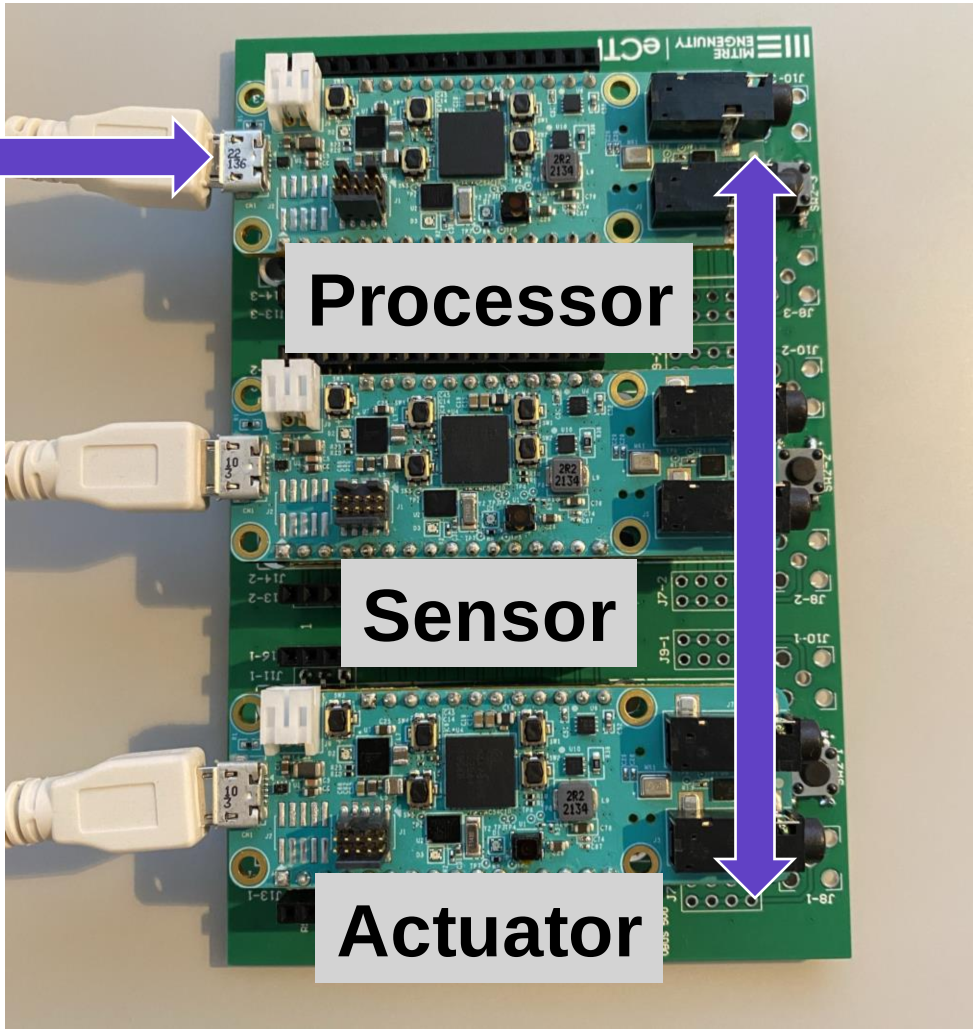
Data Security

Component Replacement

Medical Infrastructure Supply Chain Security

#eCTF2024

Host Computer

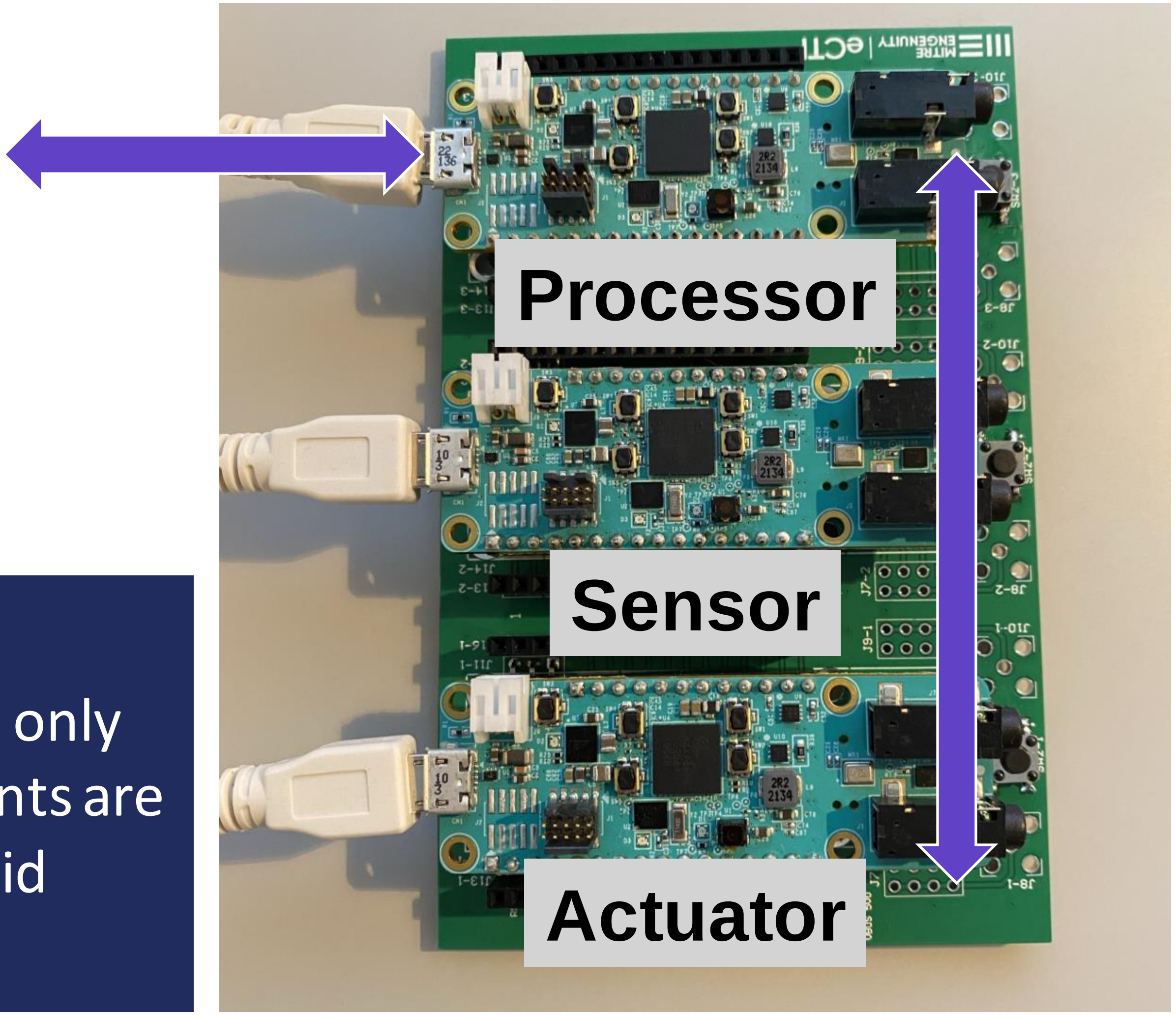


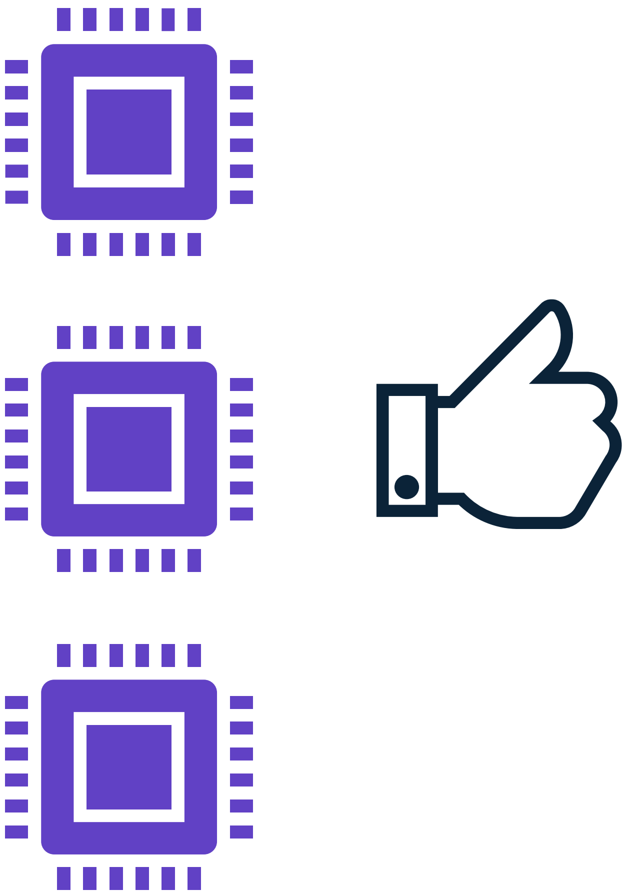
Pre-Boot	List Components
	Attest Components
	Replace Component
Boot	Boot Medical Device
Post-Boot	Secure Comms

Security Requirements

#eCTF2024

Host Computer



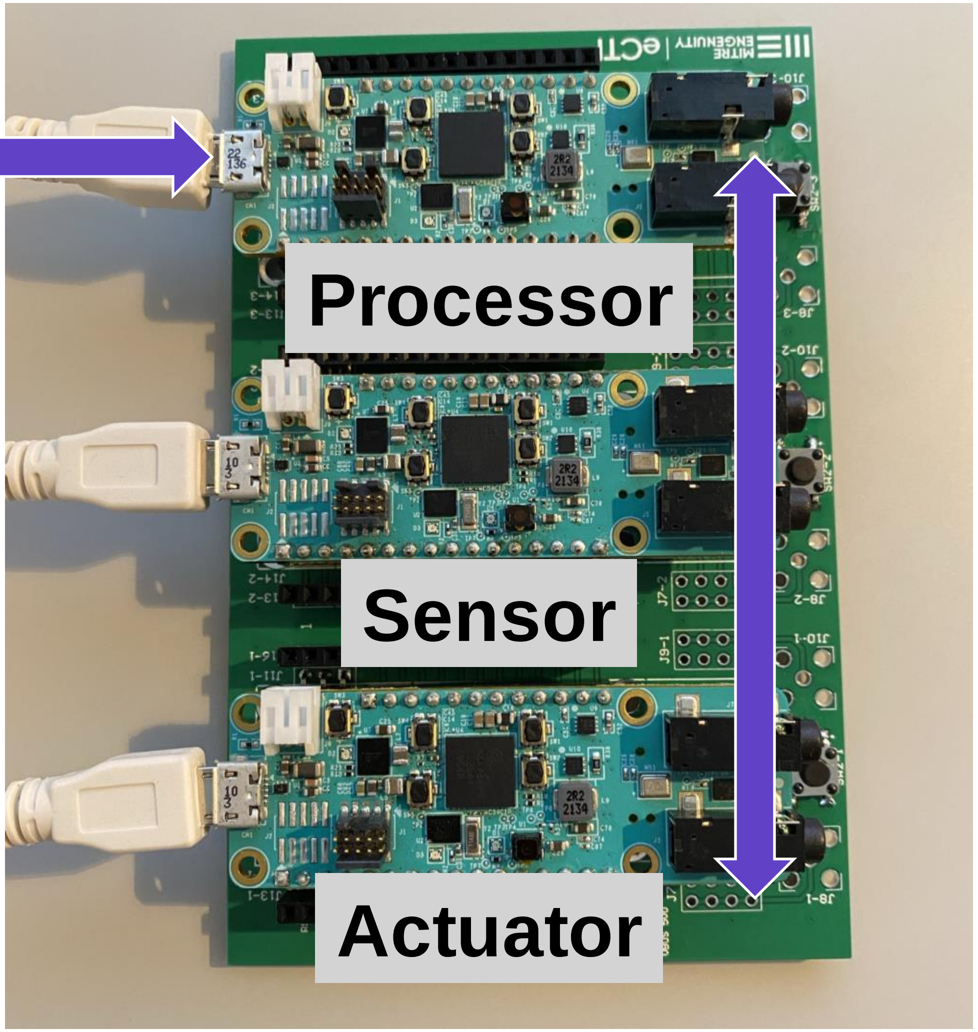


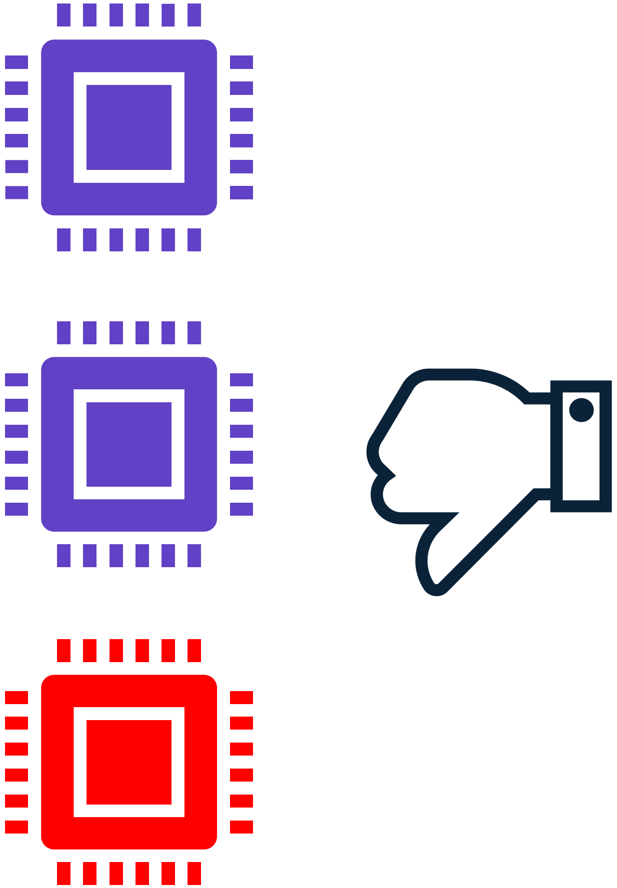
The device should only boot if all components are present and valid

Security Requirements

#eCTF2024

Host Computer





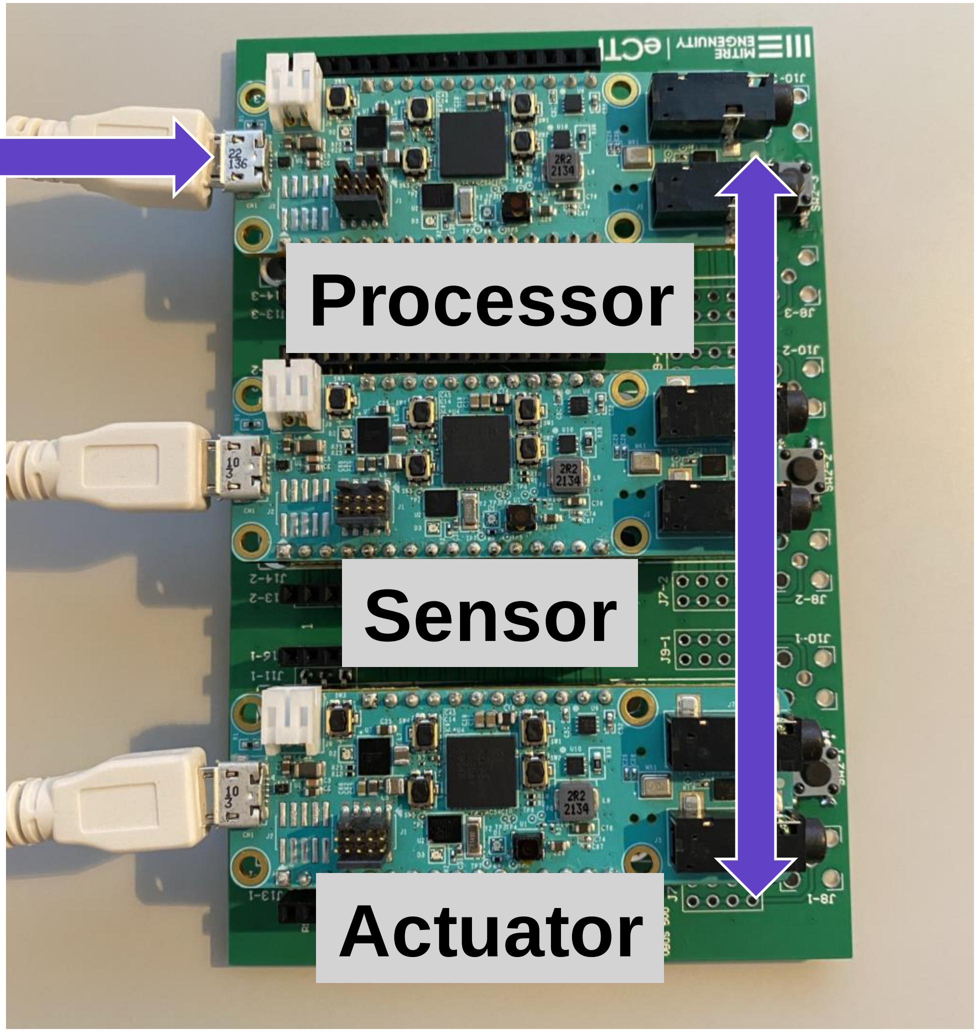
The device should only boot if all components are present and valid

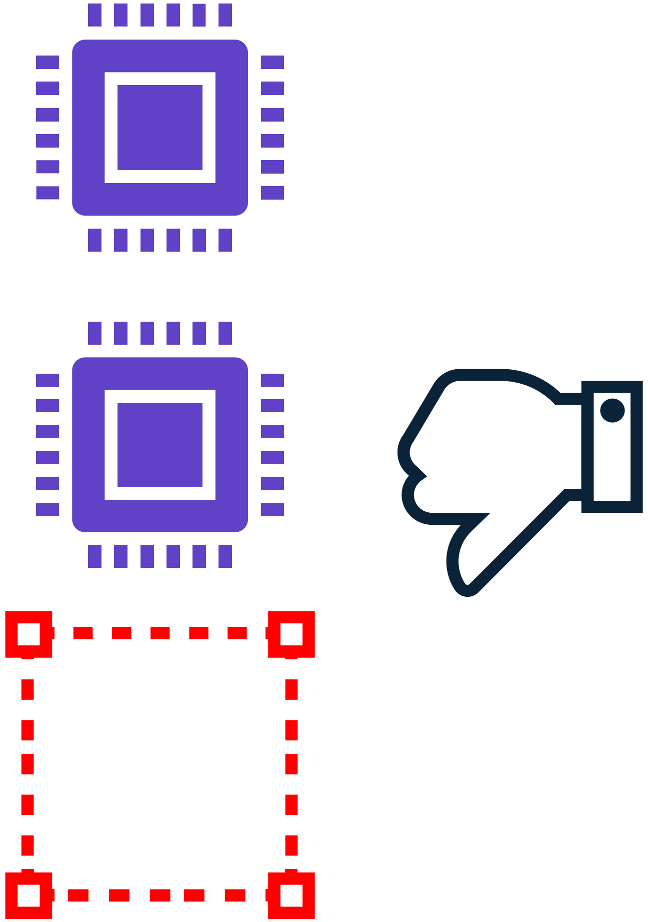
MITRE | MITRE ENGENUITY

Security Requirements

#eCTF2024

Host Computer



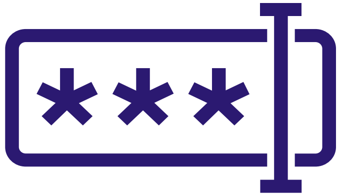


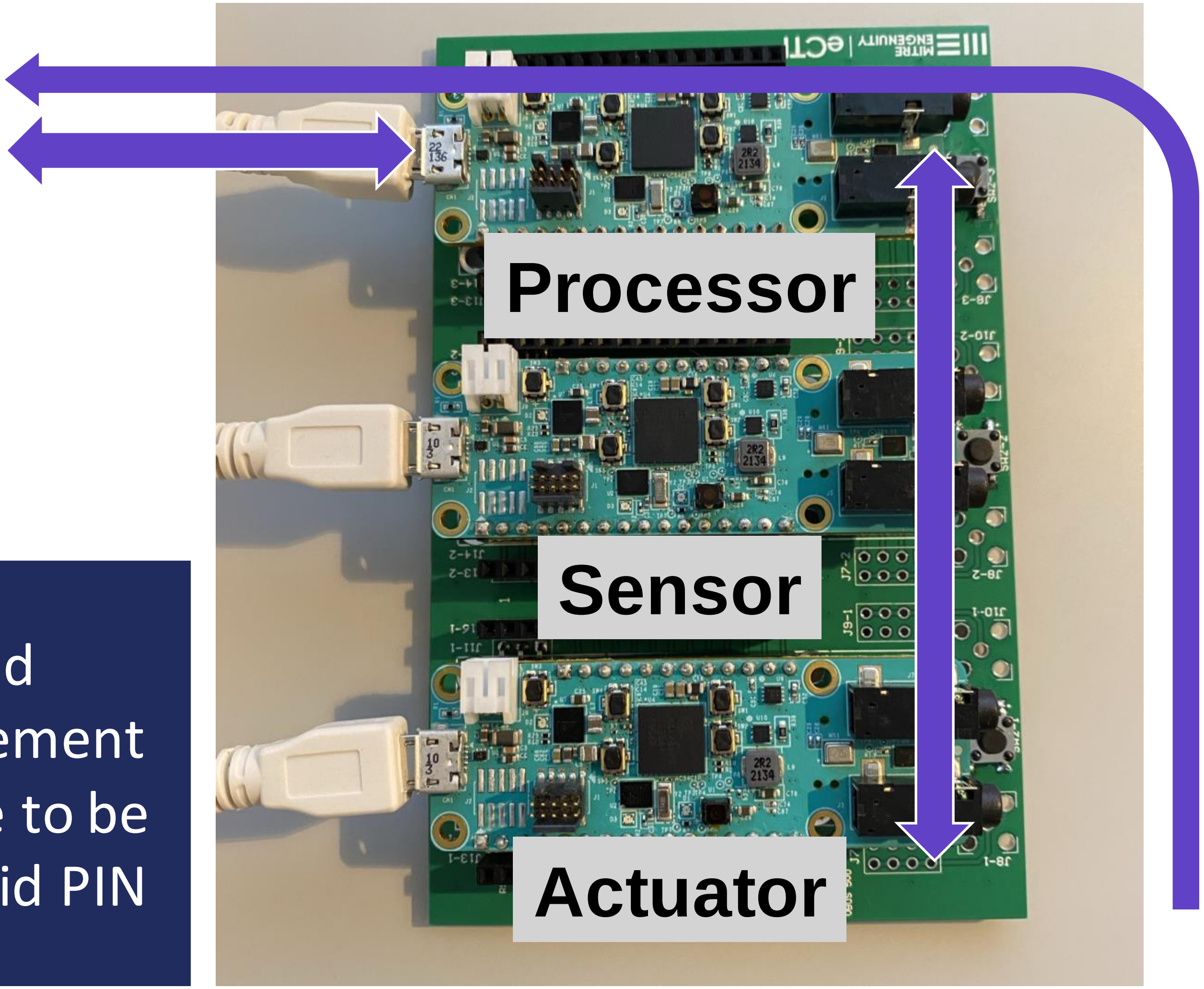
The device should only boot if all components are present and valid

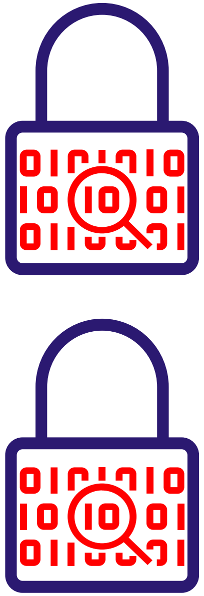
Security Requirements

#eCTF2024

Host Computer







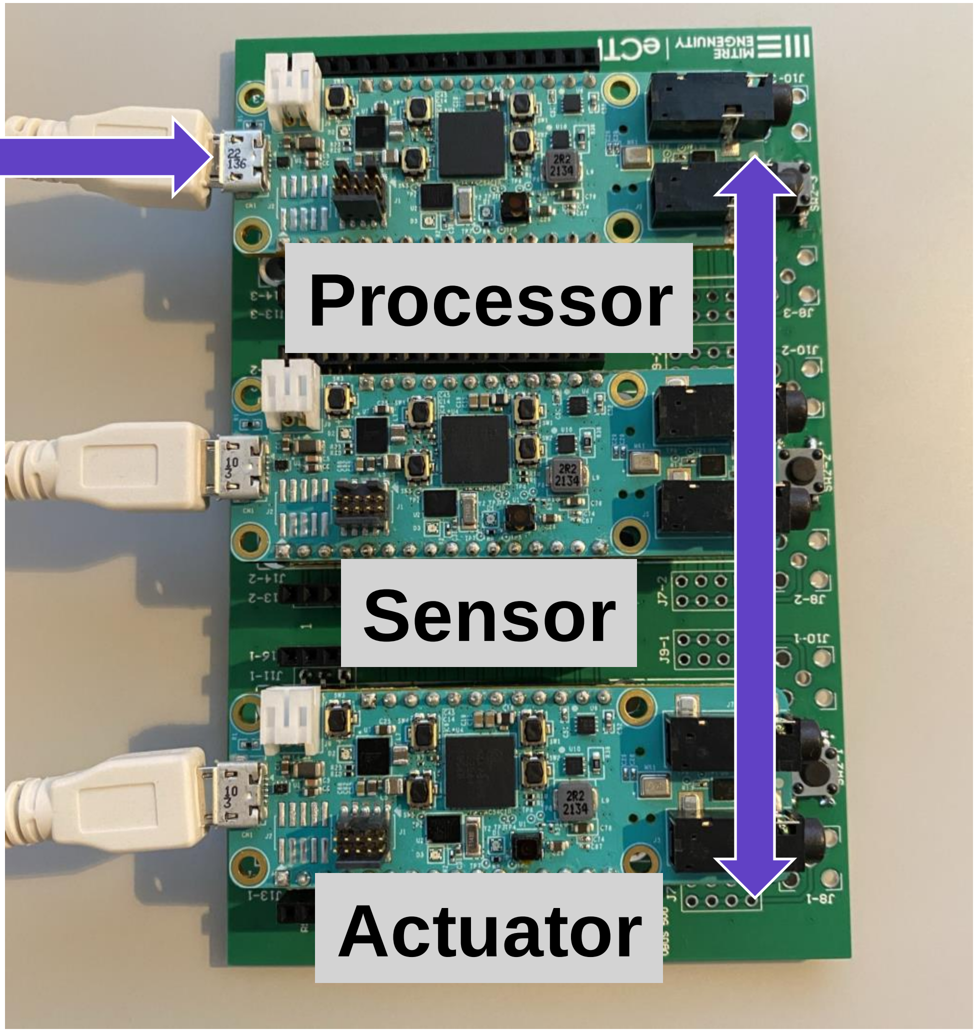
Secure data and component replacement should only be able to be accessed with a valid PIN

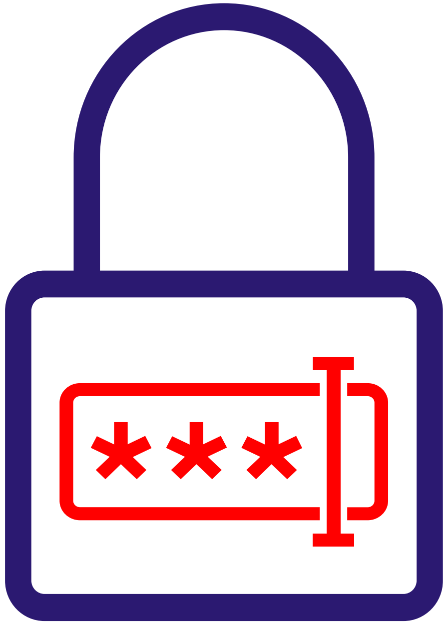
MITRE | MITRE ENGENUITY

Security Requirements

#eCTF2024

Host Computer



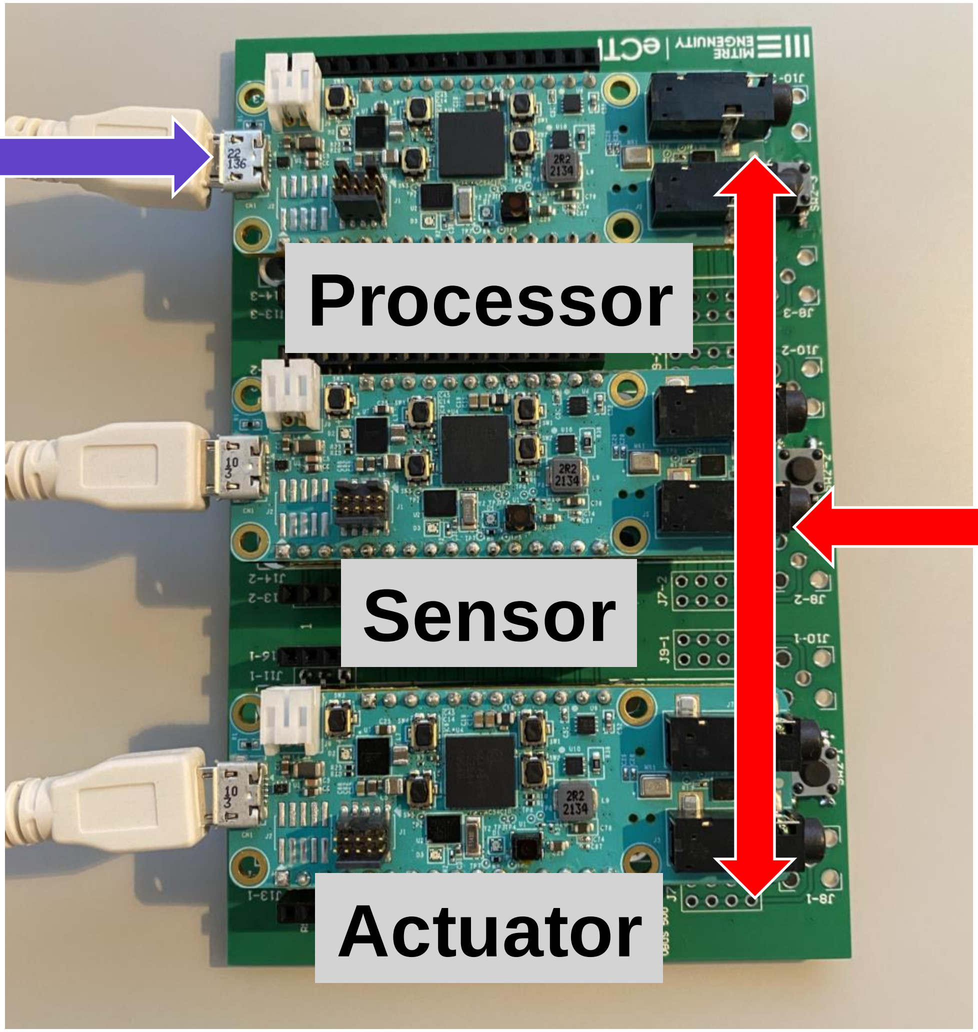


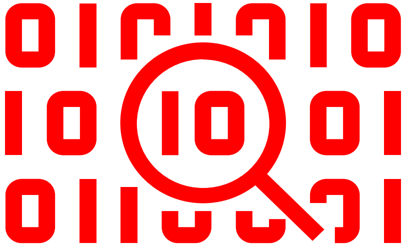
The PINs should remain confidential

Security Requirements

#eCTF2024

Host Computer





Secure communications post-boot should not be able to be forged or duplicated by an attacker

Attacker Resources

#eCTF2024

Full Access

Used by Technician

	Application Processor	Component A	Component B	Replacement Token	Attestation PIN	Physical Access
Device 0 Operational Device	Full Access	Full Access	Full Access			Full Access
Device 1 Damaged Device	Full Access	Full Access				Full Access
Device 2 Supply Chain Poisoning	Used by Technician	Used by Technician		Used by Technician	Used by Technician	
Device 3 Black Box			Full Access			Full Access

Please be respectful during presentations and Q&A

Event is being recorded

Team Presentations

Today’s Presentation Agenda

#eCTF2024

- 10:45 University of Illinois Urbana-Champaign
- 11:00 Delaware Area Career Center
- **11:15 BREAK**
- 11:25 A Word from NSTXL
- 11:35 Purdue University
- 11:50 Michigan State University
- 12:05 University of California, Irvine
- **12:20 LUNCH / NETWORK**
- 1:20 University at Buffalo
- 1:35 Carnegie Mellon University
- 1:50 A Word from Fortinet
- 2:05 Award Presentation
- 2:20 Closing Remarks / Student Dismissal

University of Illinois Urbana-Champaign

#eCTF2024

Welcome

SIGPwny

Currently 2nd place with 26,318 points

MITRE ENGENUITY

SIGPwny

University of Illinois Urbana-Champaign (UIUC)

Advisor

Professor Kirill Levchenko, PhD

Team Leads

Minh Duong, Jake Mayer, Emma Hartman, Hassam Uddin

Team Members

Juniper Peng, Timothy Fong, Krish Asher, Adarsh Krishnan, Liam Ramsey, Yash Gupta, Suchit Bapatla, Akhil Bharanidhar, Zhaofeng Cao, Ishaan Chamoli, Tianhao Chen, Kyle Chung, Vasunandan Dar, Jiming Ding, Sanay Doshi, Shivaditya Gohil, Seth Gore, Zexi Huang, George Huebner, Haruto Iguchi, Parithimaal Karmehan, Jasmehar Kochhar, Arjun Kulkarni, Julia Li, Jingdi Liu, Richard Liu, Theodore Ng, Stefan Ninic, Henry Qiu, Neil Rayu, Ram Reddy, Sam Ruggerio, Naavya Shetty, Arpan Swaroop, Raghav Tirumale, Yaoyu Wu

Design Phase

Design Methodology

- No code until protocol was fully created
  - This gave us time to properly design our implementation to ensure that there were no fundamental vulnerabilities
  - After the protocol is created, writing code is simply following the protocol – it also allows team members to easily get into writing code
- Sub-teams for each area that we wanted to focus in:
  - Pre-boot (List, Replace, Attest)
  - Secure Communications (Boot, HIDE protocol)
  - Build (Post-Boot, secrets/generation, Rust library)
  - Attack (research HW attacks, build exploits for insecure example)

eCTF 2024

Complete

Status | Timeline | + New view

Filter by keyword or by field

Discard | Save

	Title	Team	Status	End date	Labels	Milestone
	Pre-Boot/Attest Subteam 6					
20	Implement List Components #31	Pre-Boot/Attest Subteam	Done	Mar 3, 2024	FR - List Components	Begin Testing
21	Implement Attestation #32	Pre-Boot/Attest Subteam	Done	Mar 3, 2024	FR - Attestation	Begin Testing
22	Implement Replacement #33	Pre-Boot/Attest Subteam	Done	Mar 3, 2024	FR - Replace Components	Begin Testing
23	Initial protocol for List Components #4	Pre-Boot/Attest Subteam	Done	Feb 10, 2024	documentation, FR - List C	Begin Implementation
24	Initial protocol for Attestation #5	Pre-Boot/Attest Subteam	Done	Feb 10, 2024	documentation, FR - Attes	Begin Implementation
25	Initial protocol for Replacement #6	Pre-Boot/Attest Subteam	Done	Feb 10, 2024	documentation, FR - Repla	Begin Implementation
+	Add item					
	Comms Subteam 4					
26	Implement Boot Verification protocol using HIDE #28	Comms Subteam	Done	Mar 3, 2024	FR - Boot Verification	Begin Testing
27	Implement HIDE protocol #27	Comms Subteam	Done	Mar 3, 2024	FR - Secure Comms	Begin Testing
28	Initial protocol for HIDE secure communications layer #2	Comms Subteam	Done	Feb 10, 2024	documentation, FR - Secu	Begin Implementation
29	Initial protocol for Boot Verification #3	Comms Subteam	Done	Feb 10, 2024	documentation, FR - Boot	Begin Implementation
+	Add item					
	Build Subteam 8					
30	Implement fault-injection resistant patterns #47	Build Subteam	Done	Mar 5, 2024	Attack	🚀 Handoff
31	Add secure send/receive C interfaces for POST_BOOT code #22	Build Subteam	Done	Mar 4, 2024	FR - Build System	Begin Testing
32	Add `mxc_delay.h` and `led.h` support to POST_BOOT code #53	Build Subteam	Done	Mar 4, 2024	FR - Build System	Begin Testing

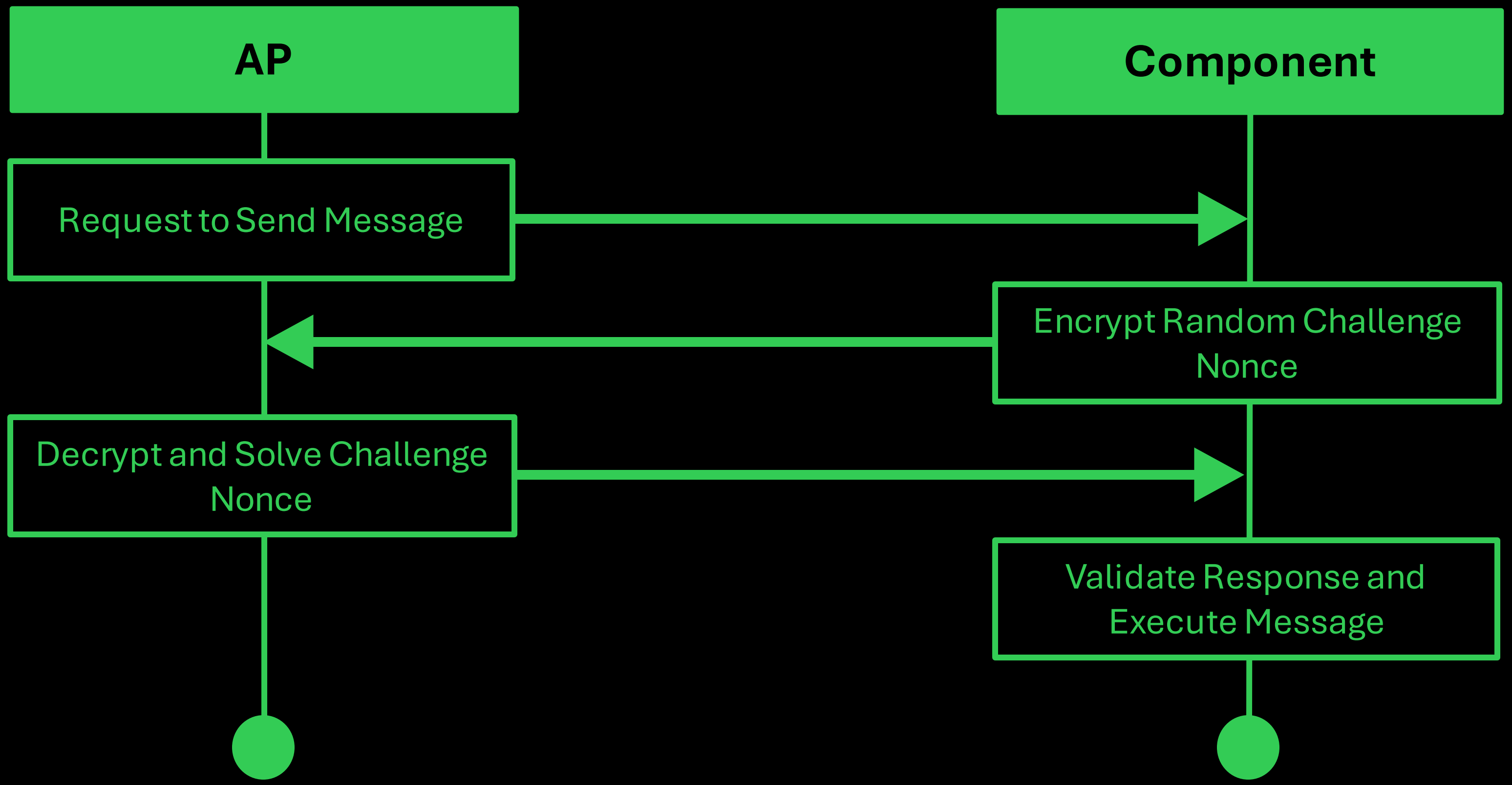
Design Overview

- Rust (memory-safe)
- HIDE protocol with Ascon-128 cryptographic scheme
  - Transforms message into three-way challenge response handshake
  - Prevents forging/replay attacks
- Delays
  - Constant delays prevent brute-force attacks
  - Random delays deter hardware attacks (fault injection)

HIDE Protocol

- Sending of message initiates HIDE Protocol
- Sender of message sends message request to begin communication
- Receiver sends random, encrypted challenge nonce
- Sender must decrypt and solve challenge
- Challenge response is encrypted and sent with message
- Receiver validates response before executing message
- Protocol ensures messages are encrypted, authenticated, verified

HIDE Protocol



Improvements to Design

- Use key-derivation functions
  - Prevents key reuse and possible cryptography attacks
- Improve anti-glitching
  - Adding more random delays
- Reduce impact from exploits
  - Component does not need to store flags in plaintext since the AP is the one that presents all boot messages or Attestation Data
- Implement memory protection unit (MPU)

Attack Phase

Attack #0: Simple I²C Component

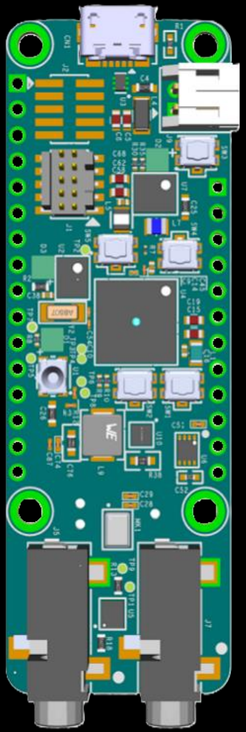
- Improper handling of I²C hardware conditions allows for a buffer overrun and arbitrary code execution
- This critical vulnerability affects the Component specifically and allows for <u>complete compromise</u> of the Component
- We developed an exploit for this vulnerability to extract Component flags and carry out attacks against the AP as well
- <u>85% of teams were vulnerable to this exploit</u> since the bug originated from the reference implementation

Attack #1

Attacking boot process with a compromised supply chain

Component A

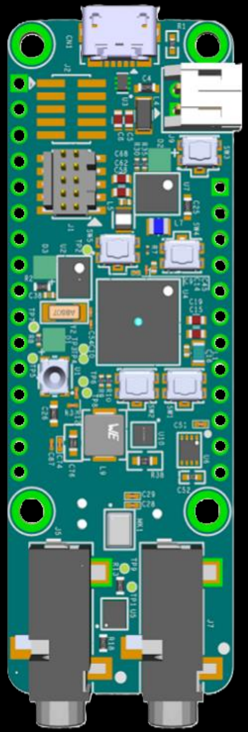
`0x11111111`



AP

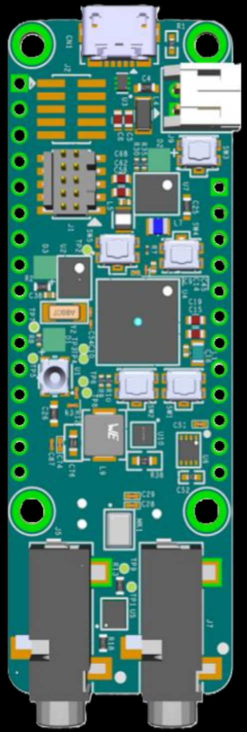
`0x11111111`

`0x22222222`



Component B

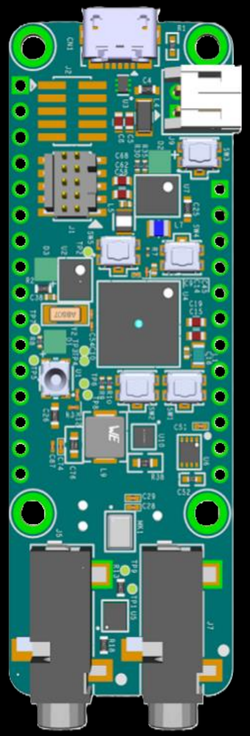
`0x22222222`



Here is a typical device configuration!

Component A

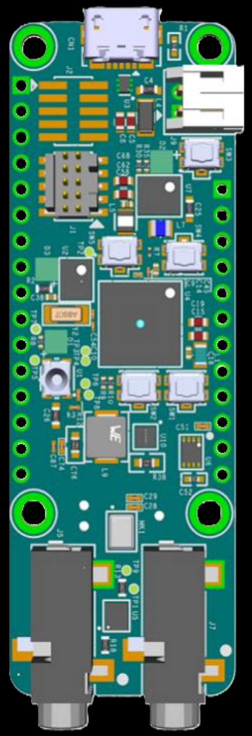
0x11111111



AP

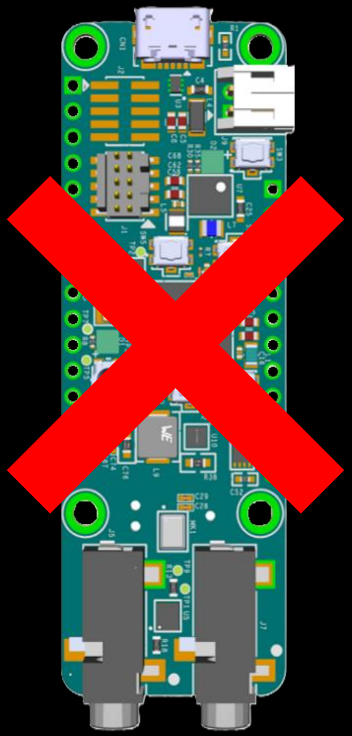
0x11111111

0x22222222



Component B

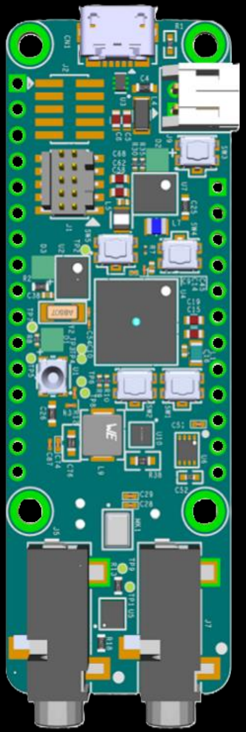
0x22222222



Component B becomes damaged!

Component A

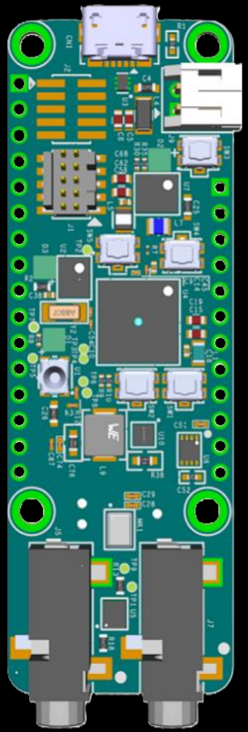
0x11111111



AP

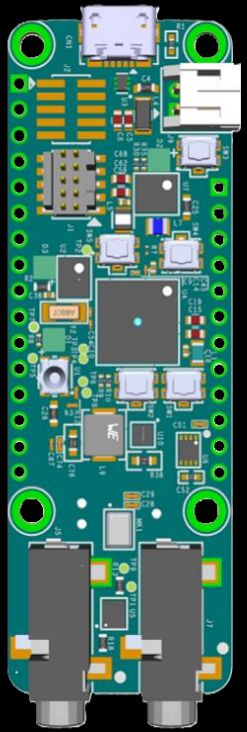
0x11111111

0x22222222



Component C

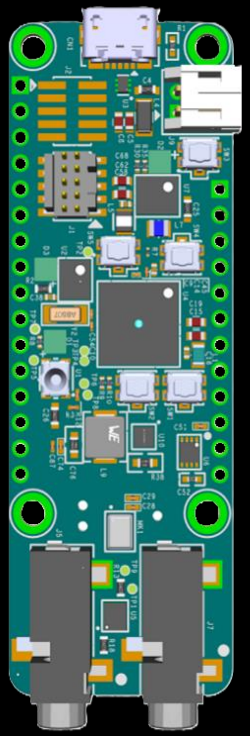
0x33333333



An authorized technician orders a new Component...

Component A

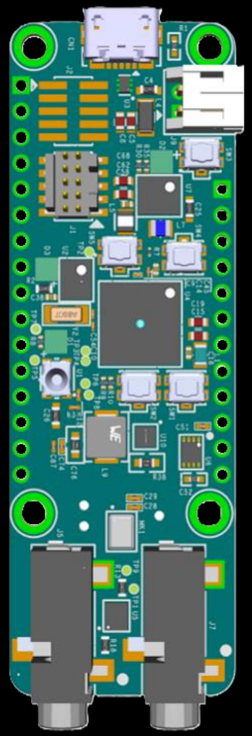
0x11111111



AP

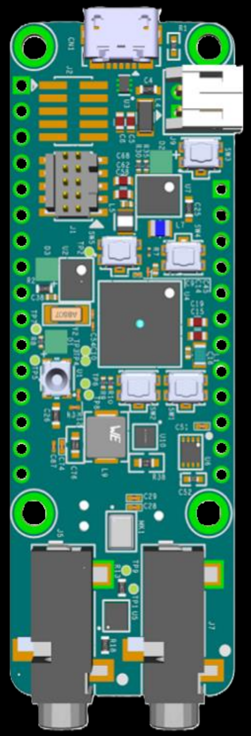
0x11111111

0x33333333

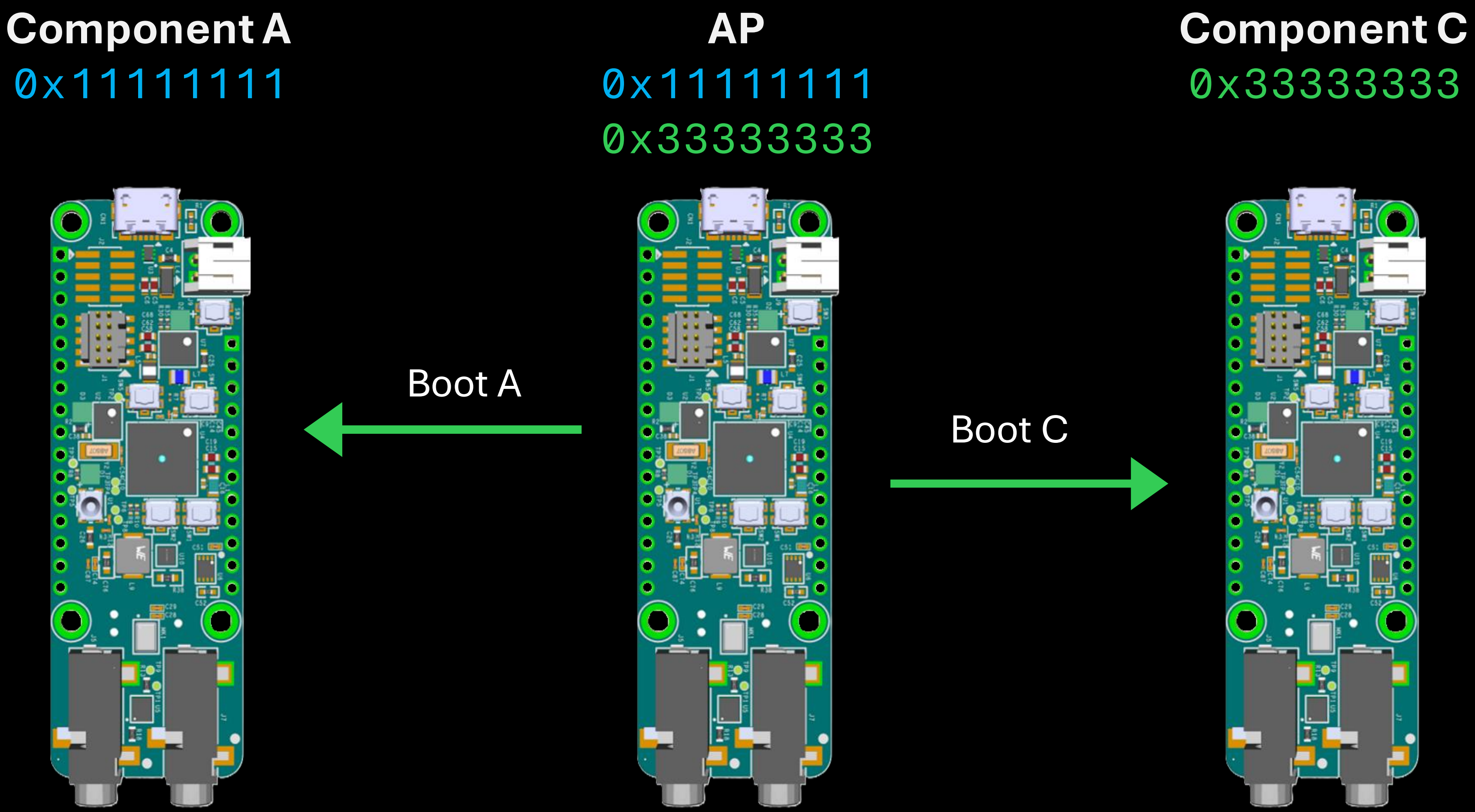


Component C

0x33333333



... and runs the replacement routine on the AP.

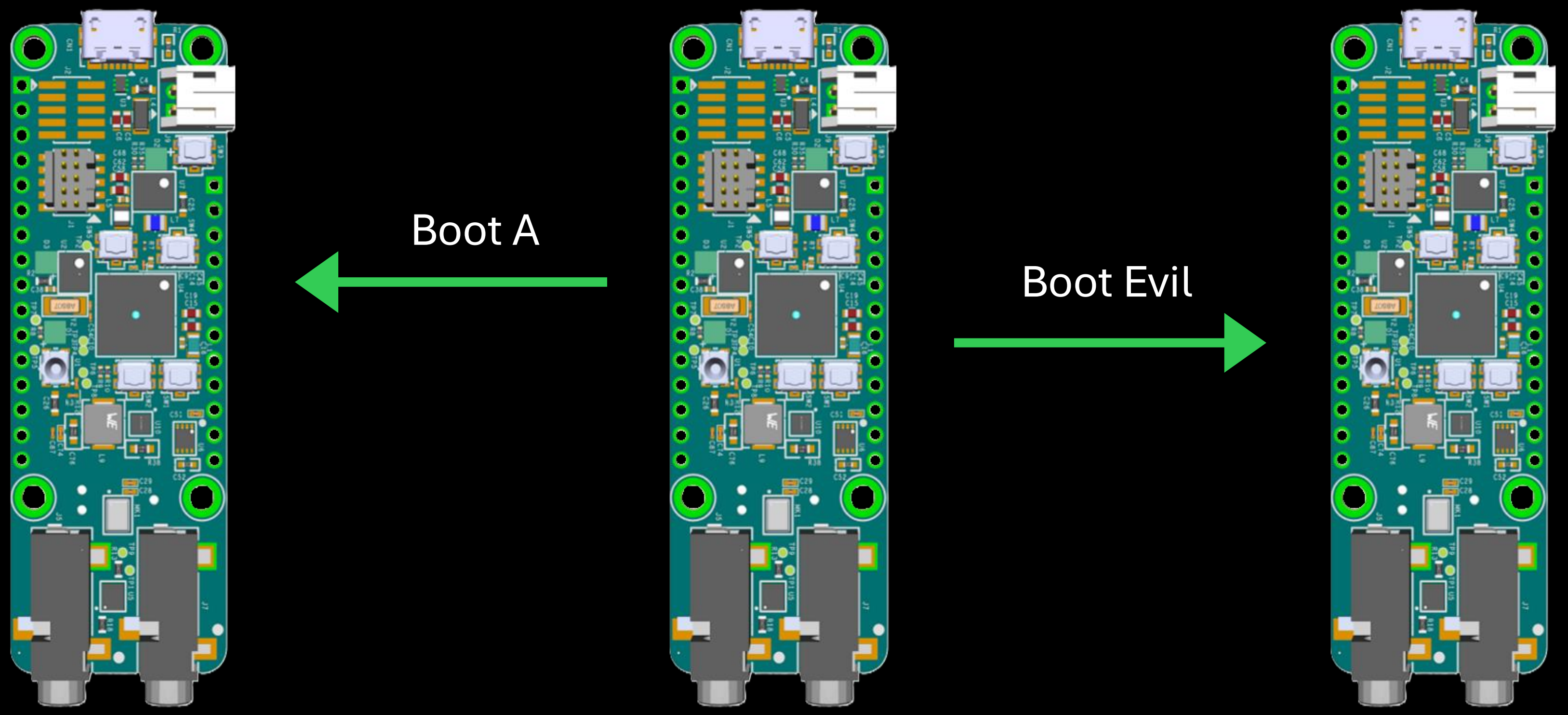


The device should be able to boot!

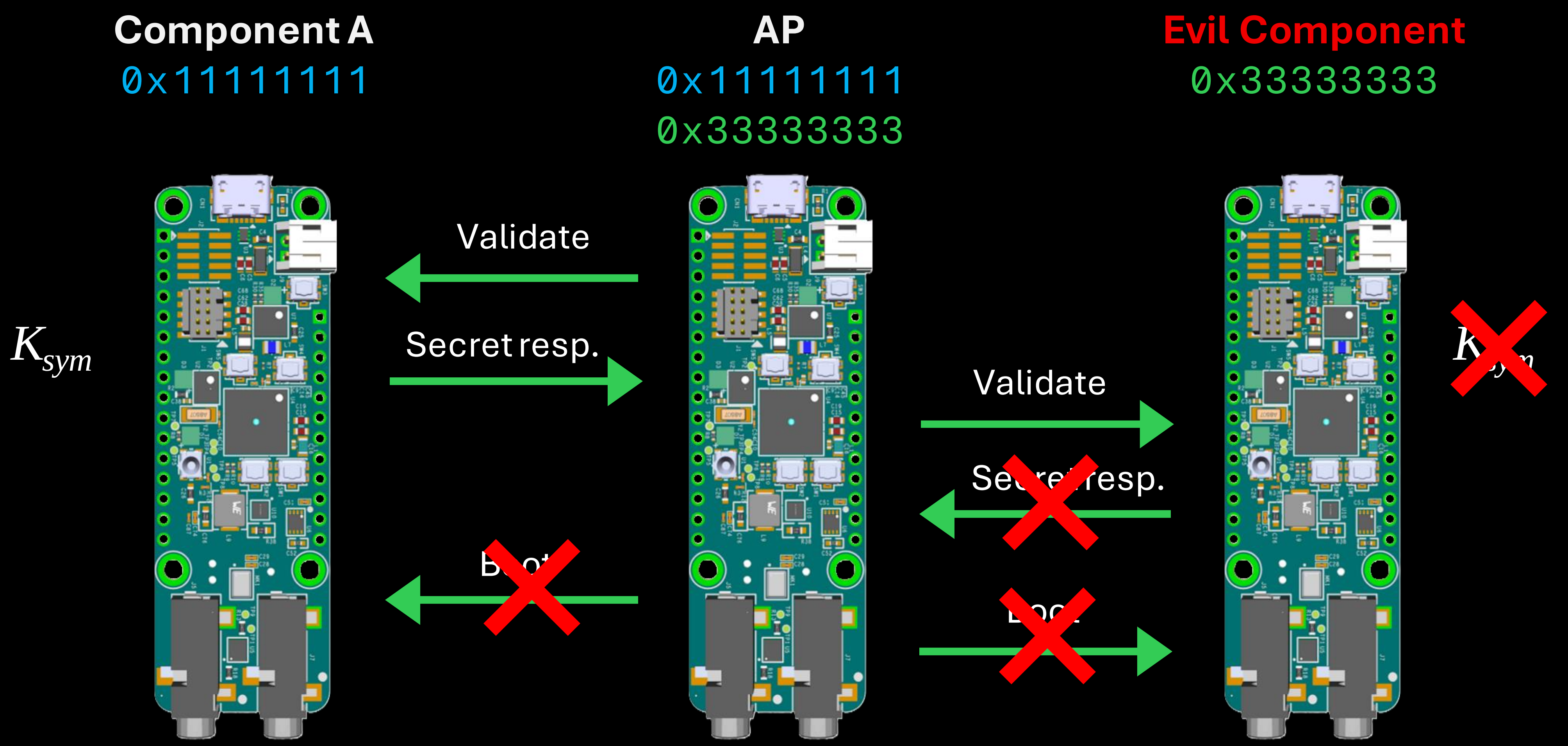
Component A
0x11111111

AP
0x11111111
0x33333333

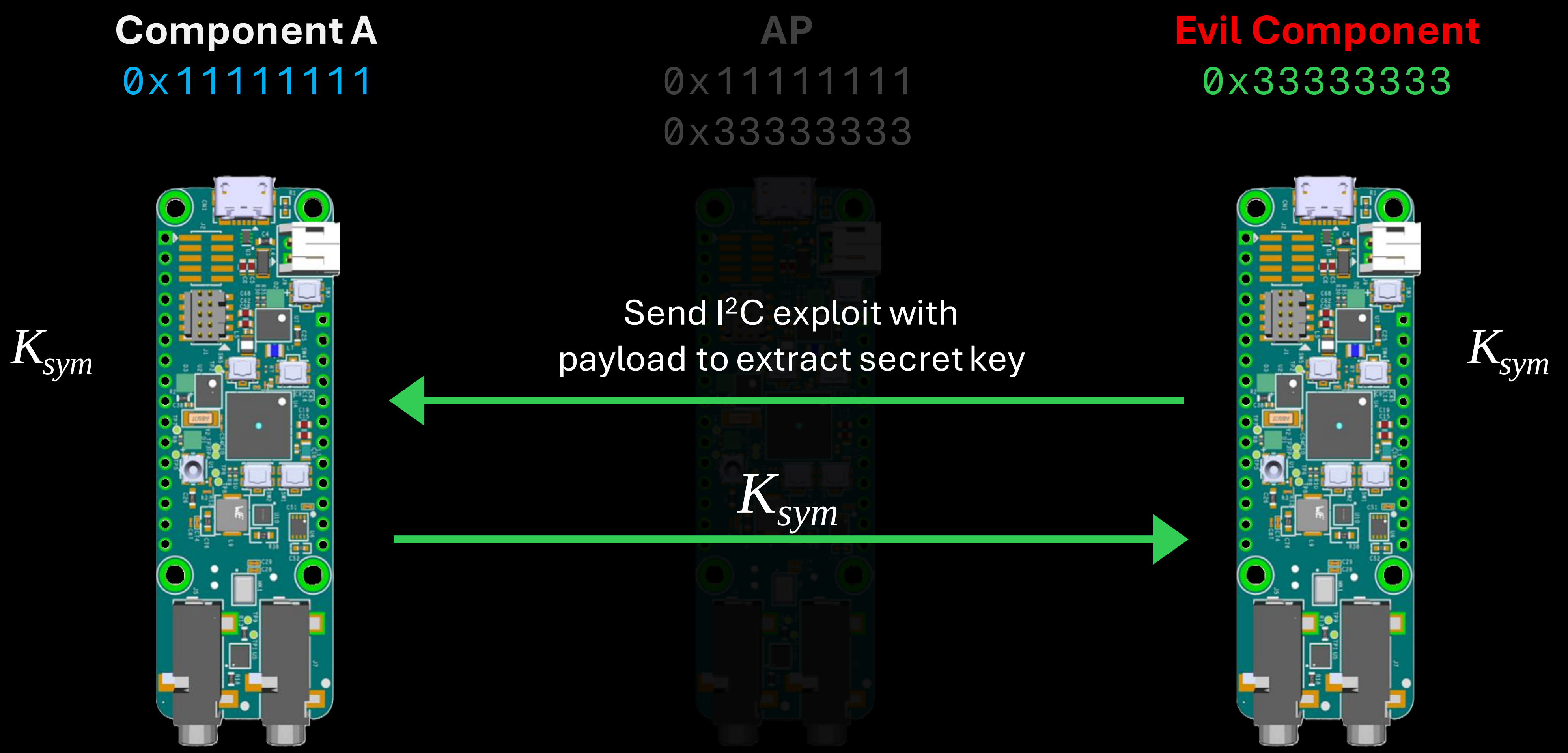
Evil Component
0x33333333



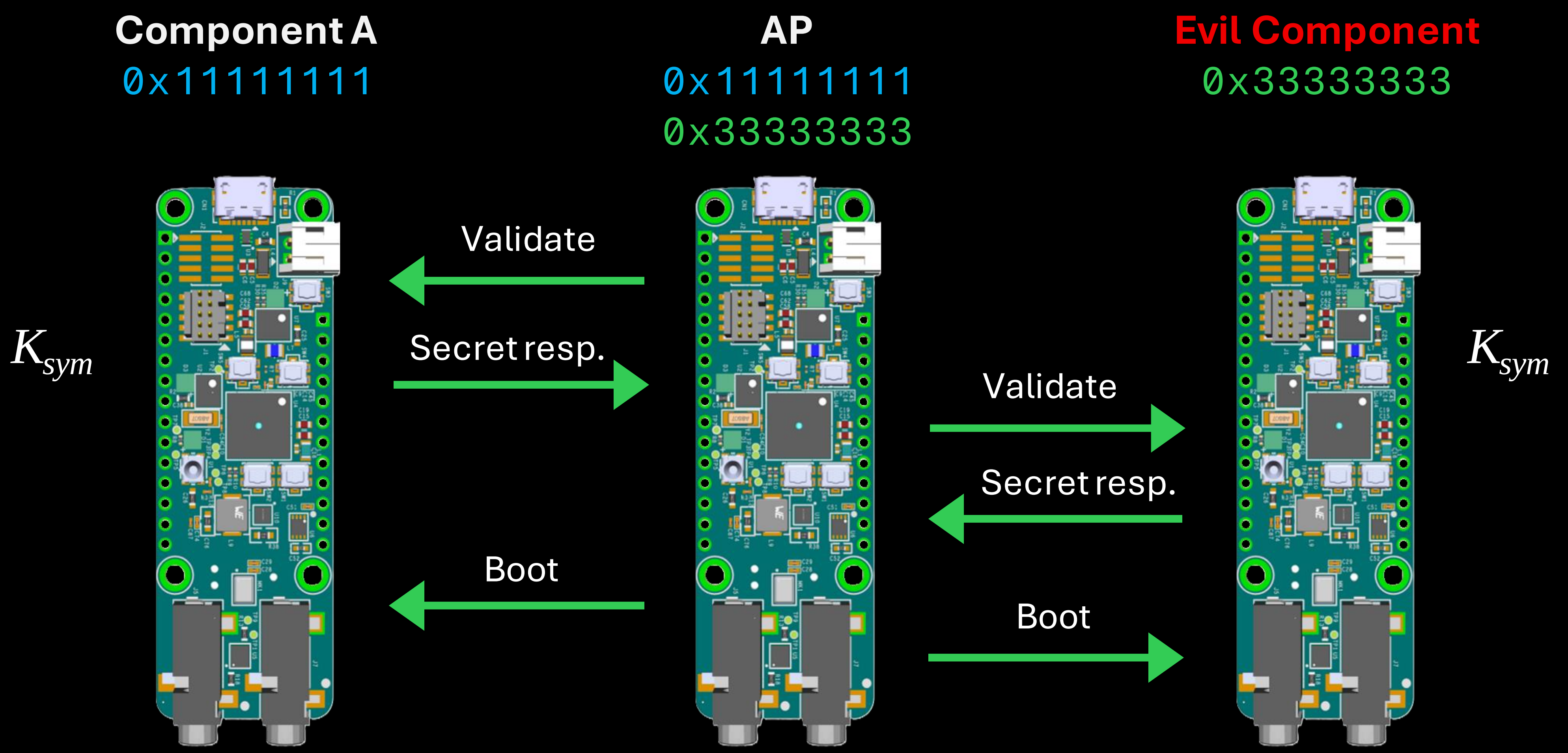
Attacker's Goal: Get the AP to boot despite an unauthentic Component being installed.



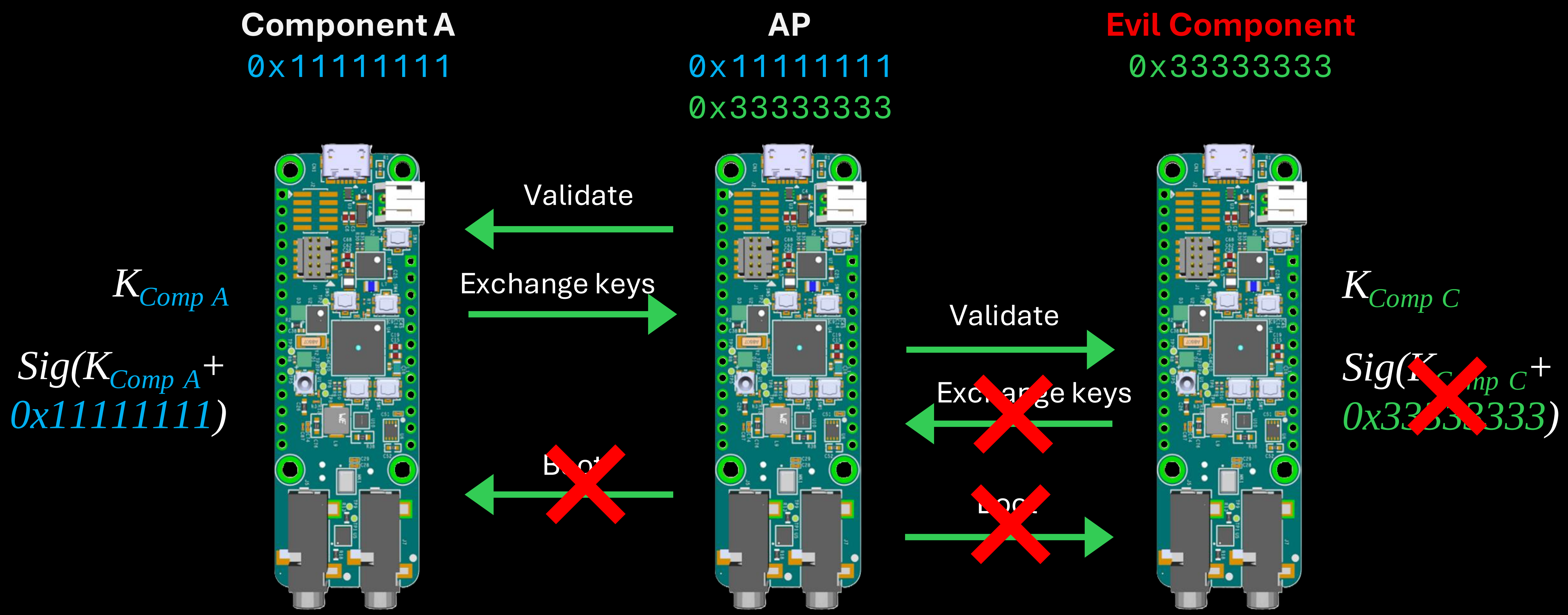
Simple Solution: Adding a validation step with a shared secret key prevents trivial attacks at booting.



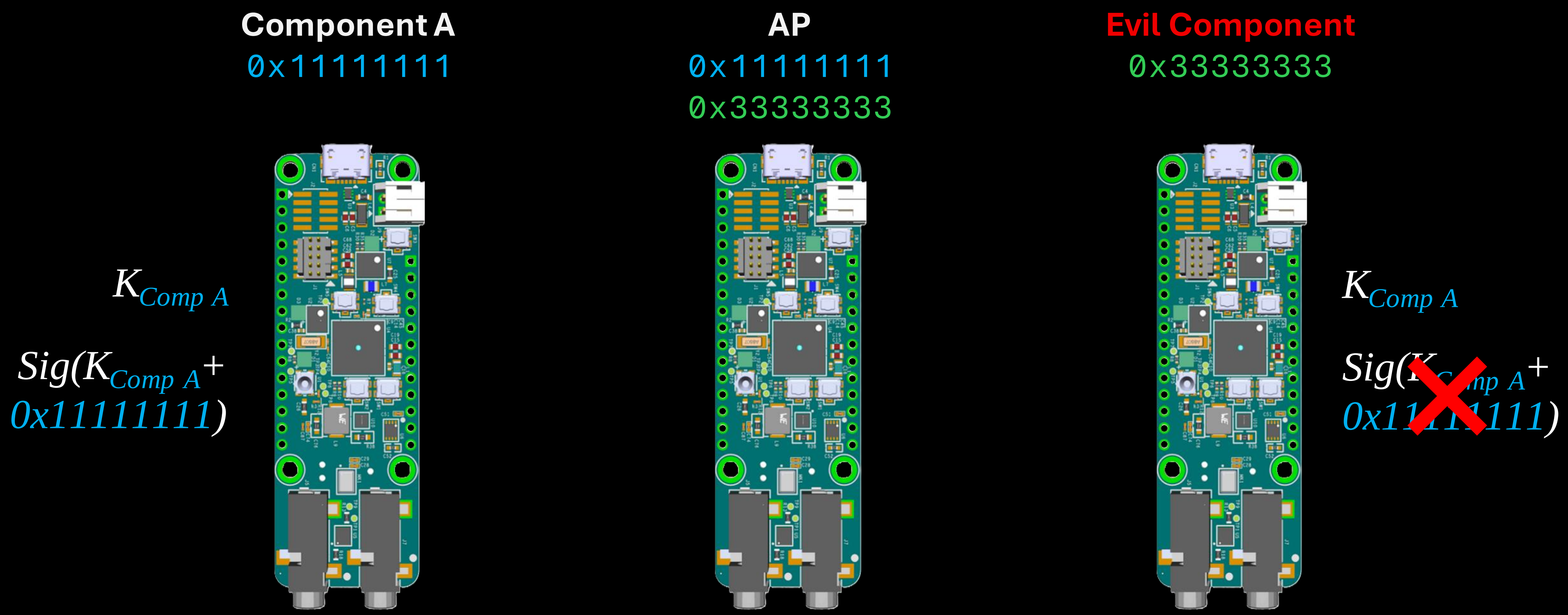
Using the I^{2}C Component exploit, we can extract secrets!



Using the I²C Component exploit, we can extract secrets!



Better Solution: Adding a validation step with <u>unique</u> secret keys and host signatures.



Better Solution: Even with the I²C exploit, the host signature is invalid because of the Component ID mismatch.

Attack #1: Analyzing Replace Code

```
if validate_token():
    CompID_New <- input()
    CompID_Old <- input()
    for i in num_components:
        if CompID_Old == component_ids[i]:
            component_ids[i] <- CompID_New
            return Success
    return Failure (“CompID_Old not found”)
return Failure (“Incorrect Token”)
```

This code does not check if `CompID_New` is <u>already</u> provisioned!

In other words: an AP can have two provisioned Components with <u>same ID</u>!

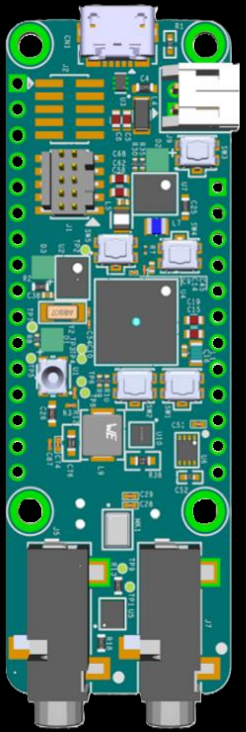
Attack #1: Exploiting Replace Code

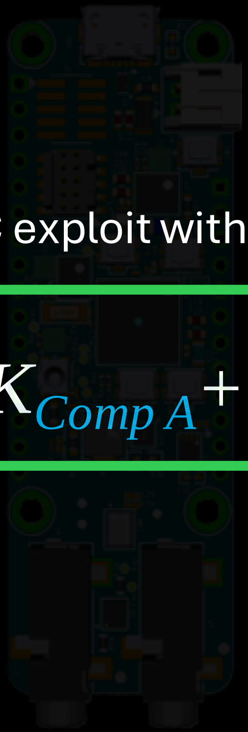
- New problem: two same Component IDs means that they share the same I^2C address, which will cause bus errors
  - Attacker's fix: use the simple I^2C exploit to disable Component A
  - This is done by changing Component A's I^2C address to 0x00
  - Our Evil Component will handle both validate and boot requests from the AP

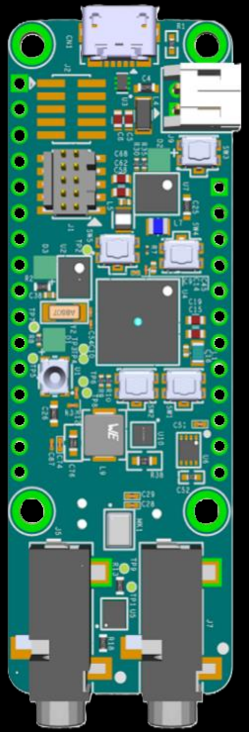
Component A
0x11111111

AP
0x11111111
0x11111111

Evil Component
0x11111111







$K_{Comp\ A}$

$Sig(K_{Comp\ A}+0x11111111)$

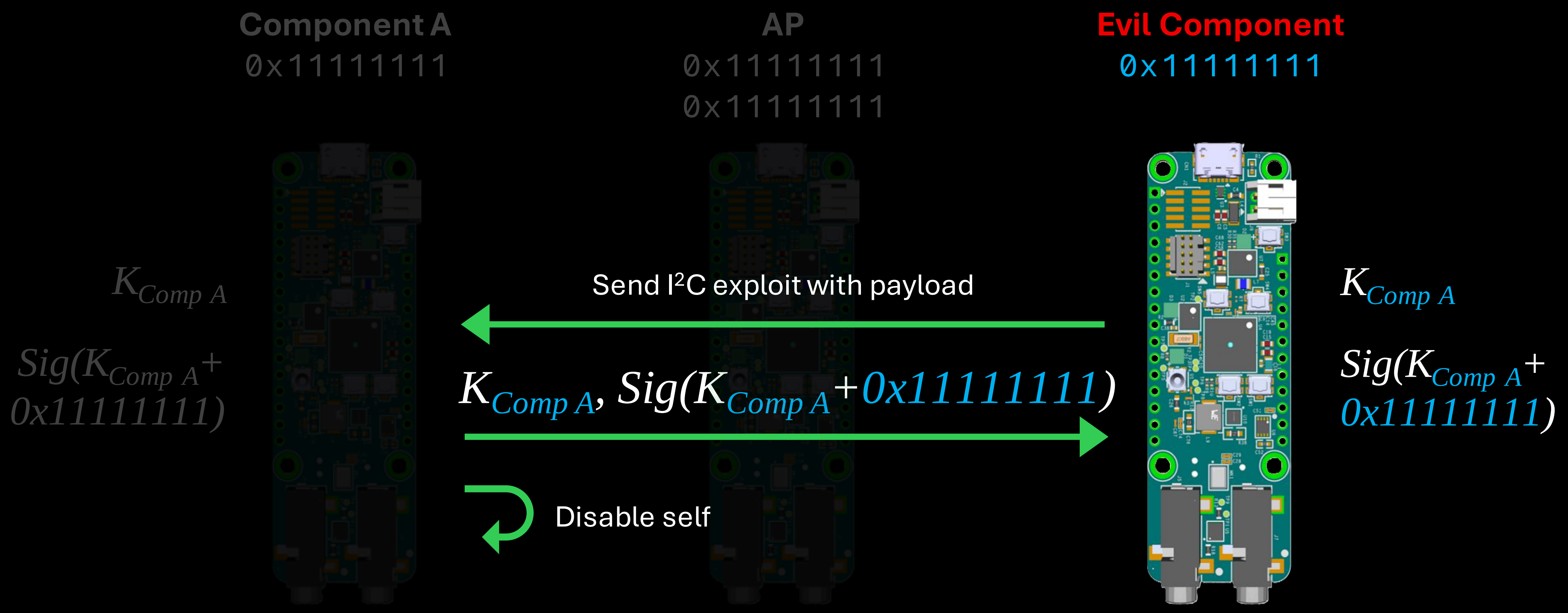
Send I²C exploit with payload

$K_{Comp\ A}, Sig(K_{Comp\ A}+0x11111111)$

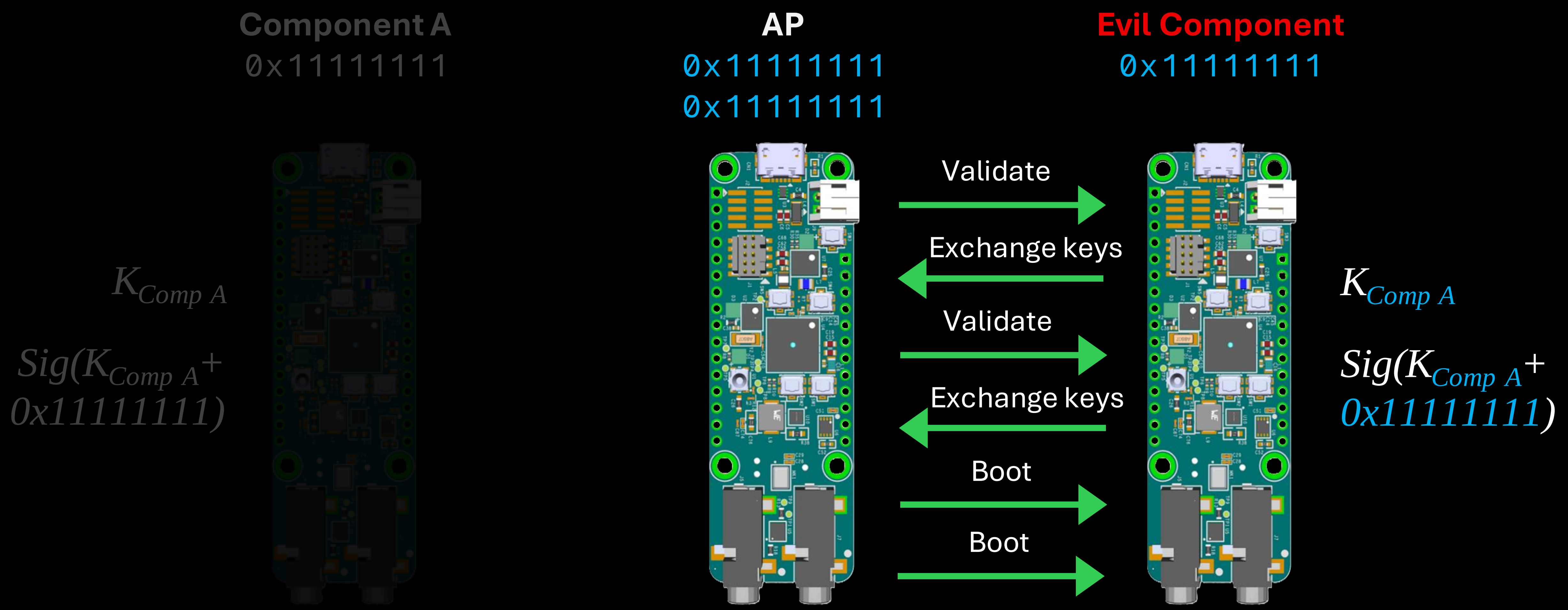
$K_{Comp\ A}$

$Sig(K_{Comp\ A}+0x11111111)$

Use the I²C Component exploit to extract the unique secret key and signature, then disable Component A!



Use the I²C Component exploit to extract the unique secret key and signature, then disable Component A!



The attacker has successfully tricked the AP into booting!

Attack #2

Hardware attacks against the MAX78000FTHR board

Attack #2: Hardware Attack

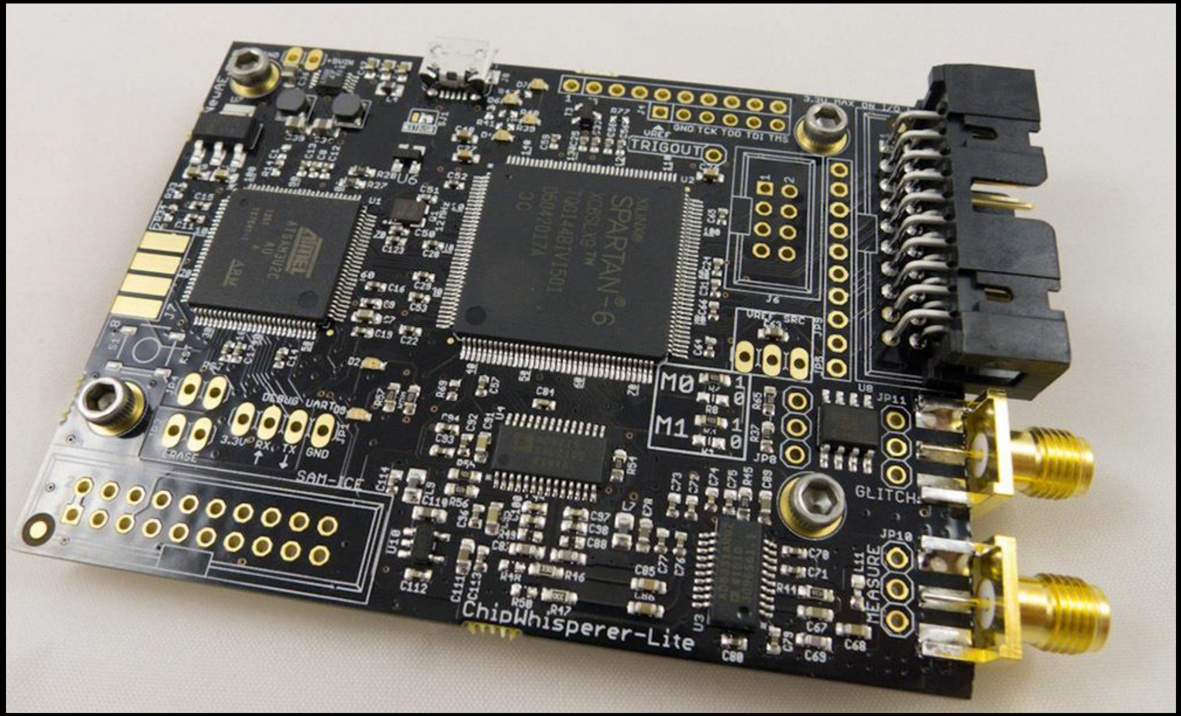
Goal: Skip an executing instruction with fault injection by a voltage glitch

Method:

- Connect ChipWhisperer to the voltage line MCU Arm core
- Pull the voltage to ground while the core is executing an instruction

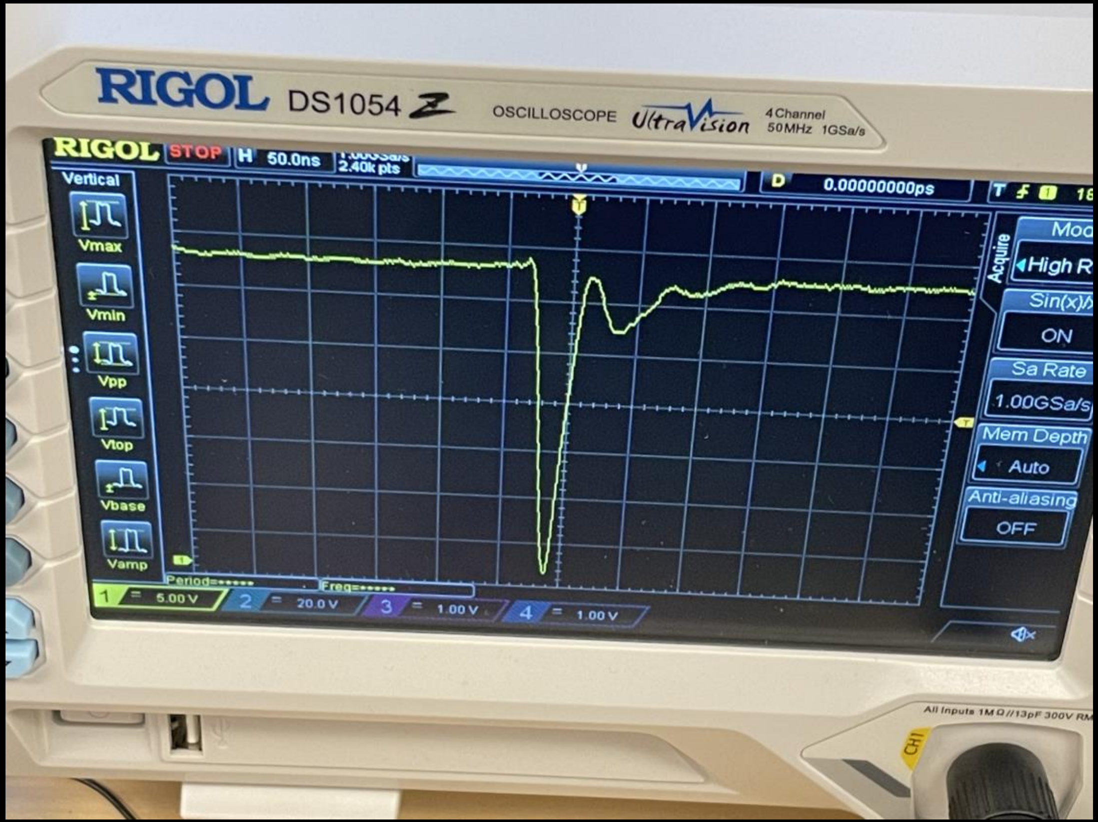
Challenges:

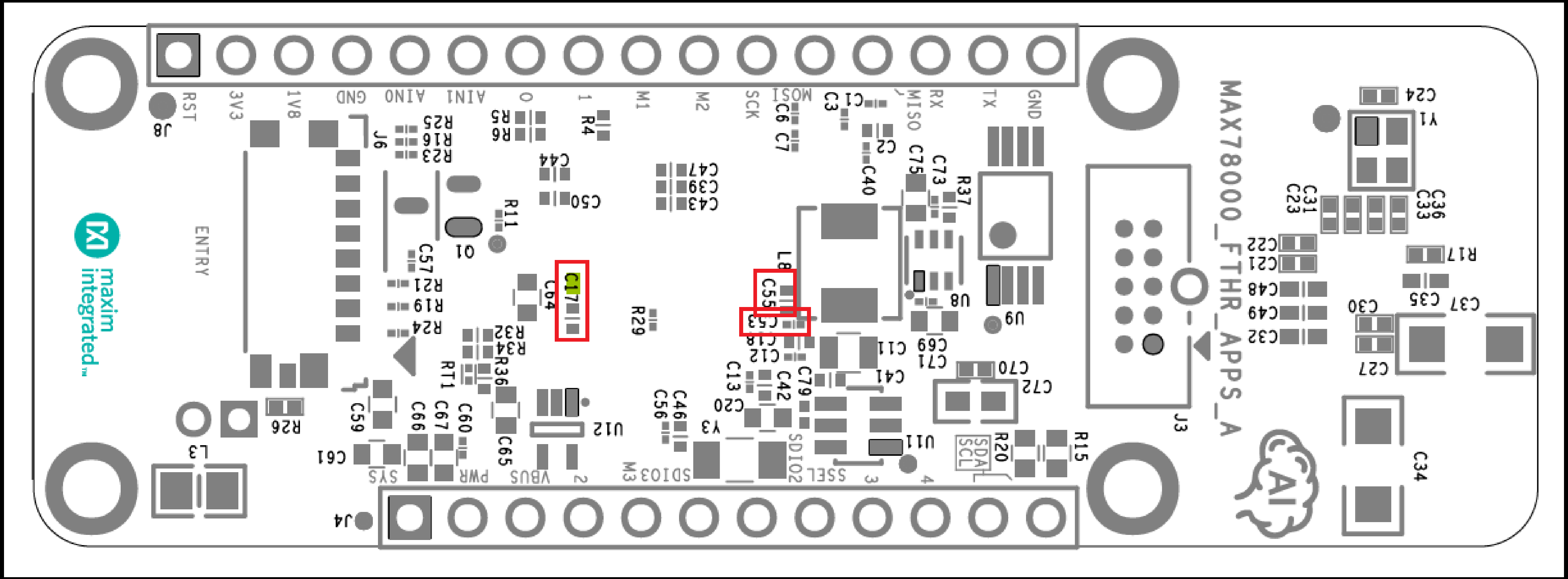
- Pulling voltage to ground for too long will cause a power reset
- Requires precise timing to pinpoint instruction to skip
- Capacitors provide limited power even though we pull to ground



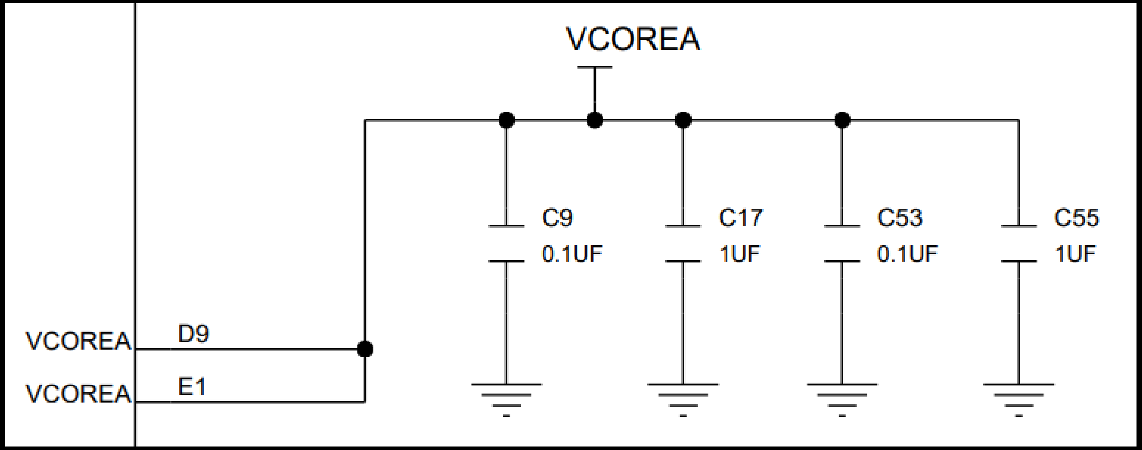
This year, we invested in a ChipWhisperer-Lite and an oscilloscope!

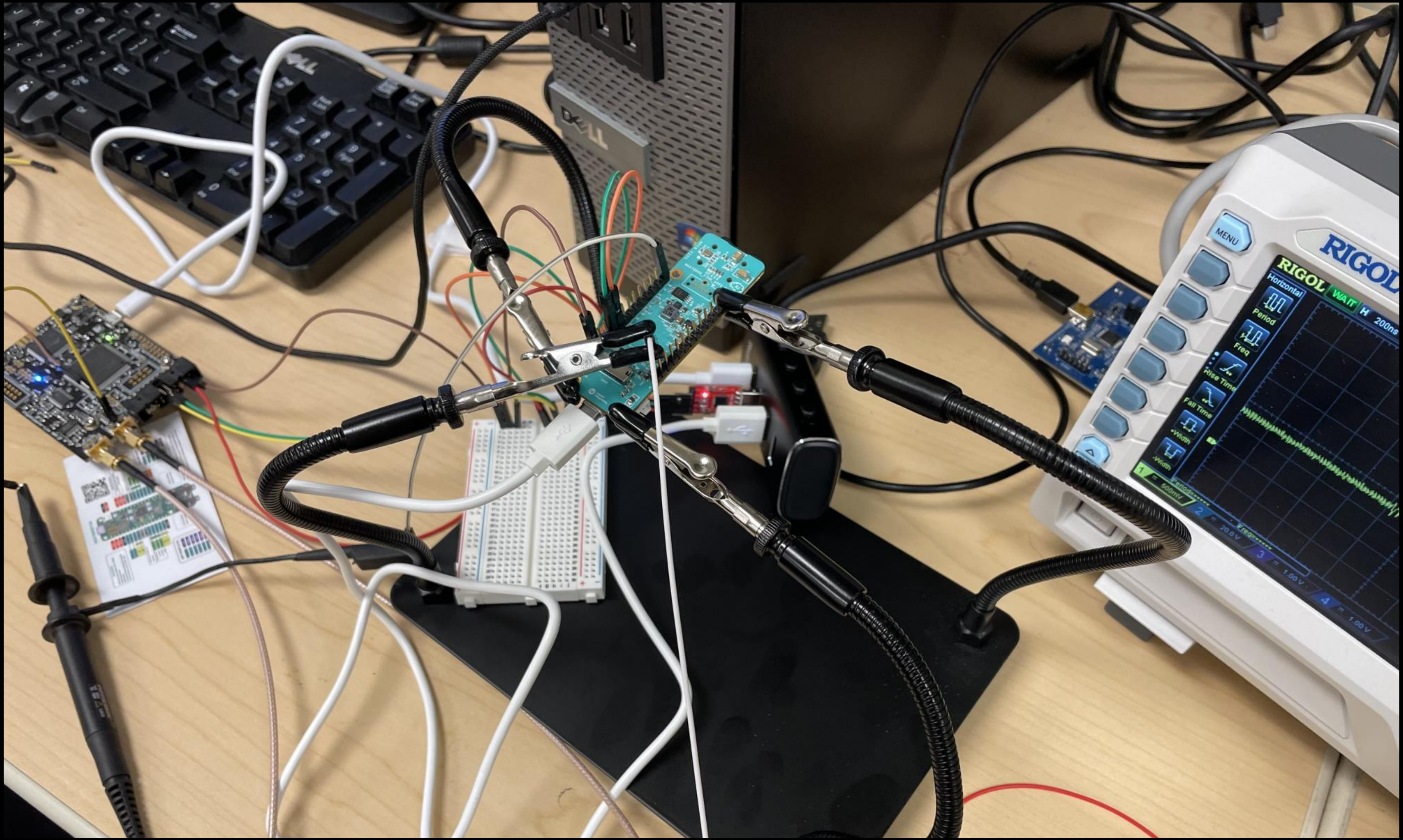
The oscilloscope demonstrates a voltage glitch attack, briefly bringing power to ground.





Reliable voltage glitching requires the removal of some capacitors.





Our test board setup for voltage glitch attacks!

Attack #2: Summary

- Implication: If you could skip any single instruction in the code, what instruction would you skip?
  - Most teams did not implement protections against this scenario
  - Voltage glitching allows bypassing security checks altogether
- Mitigations:
  - Adding truly random delays
    - If a delay is random, the attacker doesn't know when to apply the glitch
  - Multiple if statements and condition guards
    - It's difficult to skip multiple instructions in a row or time sequential skips

Other Attacks

- Attestation PIN brute force
  - Only 6 hexadecimal digits (`000000` – `ffffff`)!
  - No delays means this can be cracked quickly
- Bad schemes + secrets sent over the wire to authenticate
  - Record these secrets with a logic analyzer, build new device with secrets
- For Damaged Boot, use the same working Component to respond to validation/boot requests for a broken Component
  - Requires a MITM device to translate the I^2C addresses

Thank you! Any questions?

SIGPwny

Today's Presentation Agenda

#eCTF2024

- 10:45 University of Illinois Urbana-Champaign
- 11:00 Delaware Area Career Center
- **11:15 BREAK**
- 11:25 A Word from NSTXL
- 11:35 Purdue University
- 11:50 Michigan State University
- 12:05 University of California, Irvine
- **12:20 LUNCH / NETWORK**
- 1:20 University at Buffalo
- 1:35 Carnegie Mellon University
- 1:50 A Word from Fortinet
- 2:05 Award Presentation
- 2:20 Closing Remarks / Student Dismissal

Delaware Area Career Center

#eCTF2024

Welcome
0xDACC

DACC

Currently 5th place with 6,225 points

MITRE ENGENUITY

DELAWARE AREA CAREER CENTER

Beau Schwab **Project Manager**	Samuel Goodman **Comms. Director**	Andrew Langan **Lead Developer**	Eli Cochran **Team Advisor**
David Nunley **Developer**	Ezequiel Flores **Developer**	Grayson Seger **Developer**	Henry Reid **Developer**
Cameron Crossley **Developer**	Tyler McColeman **Developer**	Ethan Martindale **Video Editor**	Seth Tydings **Video Editor**

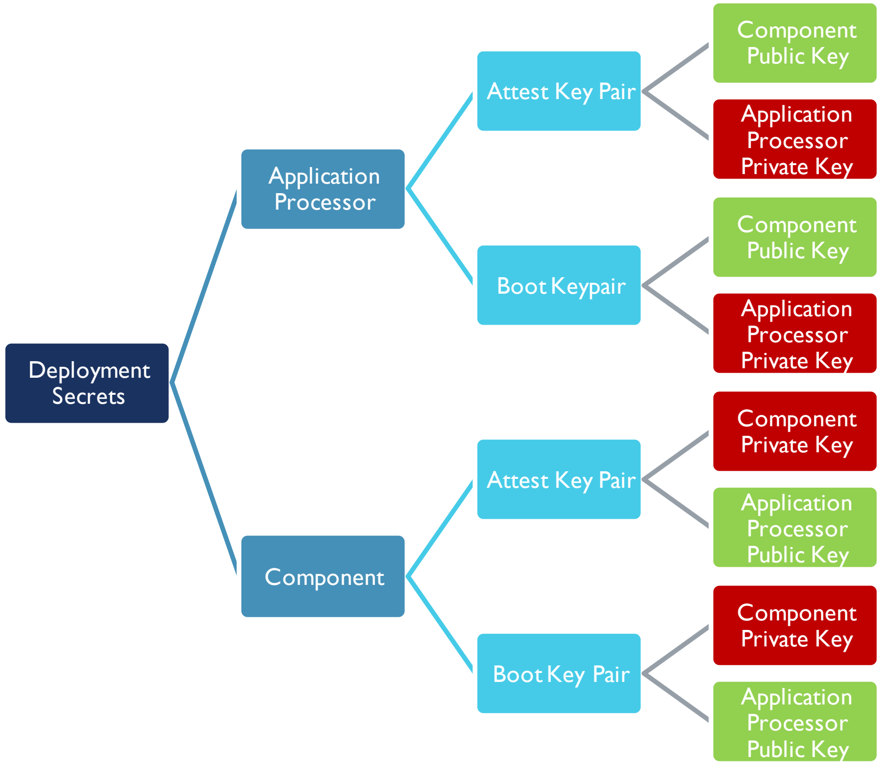
CRYPTOGRAPHIC ALGORITHMS

- AES-128-CTR
- HMAC-SHA256
- SECP256r1 ECC
- SHA256

SECURE SECRET GENERATION

Compile-time Python scripts

- deployment/make_secrets.py
  - Generates shared public/private ECC keys
  - Generates unwrapped attest AES key
  - Generates shared HMAC key
- application_processor/make_secret.py
  - Hashes PIN n times for comparison
  - Hashes PIN n-1 times as wrapper for global attest key
  - Hashes Replacement Token
- component/make_secret.py
  - Encrypts attestation data with unwrapped key



0xDACC

NEW I²C PROTOCOL

- No more registers
- Common packet format
- Packet checksums
- Callbacks instead of polling
- 1:1 Request to Response
- Length encoded along with data

mitre - packets.h

```
struct header_t {
    packet_magic_t magic;
    uint32_t checksum;
};

template<packet_type_t T> struct __packed payload_t;

// Encrypted packet payload
template<> struct __packed payload_t<packet_type_t::SECURE> {
    uint8_t magic;
    uint8_t len;
    uint32_t nonce;
    uint8_t data[64];
    uint8_t __padding[10];  // Pad to multiple of AES block size
    uint8_t hmac[32];
};

template<packet_type_t T> struct packet_t {
    header_t header;
    payload_t<T> payload;
};
```

0xDACC

ATTEST

Strengths

- Encryption of data
- Hashed PIN as wrapper
- Signature validation

mitre - application_processor.cpp

```
tc_sha256_init(&sha256_ctx);
for (uint32_t i = 0; i < ITERATIONS; ++i) {
    tc_sha256_update(&sha256_ctx, pin, 6);
}
tc_sha256_final(hash, &sha256_ctx);

if (memcmp(hash, ATTEST_HASH, 32) != 0) {
    print_error("Error :(\n");
    return;
}

tc_sha256_init(&sha256_ctx);
for (uint32_t i = 0; i < ITERATIONS - 1; ++i) {
    tc_sha256_update(&sha256_ctx, pin, 6);
}
tc_sha256_final(hash, &sha256_ctx);

unwrap_aes_key(unwrapped_key, ATTEST_KEY_WRAPPED, hash, ATTEST_WRAPPER_NONCE);
```

Weaknesses

- No delays
- Small key space
- Signature reuse

mitre - application_processor.cpp

```
const char *const fmts[3] = {"LOC", "DATE", "CUST"};

tc_aes128_set_encrypt_key(&aes_key, unwrapped_key);
memcpy(tx_packet.payload.data, "ATTEST", 0x06);

tc_sha256_init(&sha256_ctx);
tc_sha256_update(&sha256_ctx, tx_packet.payload.data, 0x07);
tc_sha256_final(hash, &sha256_ctx);

if (uECC_sign(ATTEST_A_PRIV, hash, 32, tx_packet.payload.sig,
              uECC_secp256r1()) != 1) {
    return error_t::ERROR;
}

tc_sha256_init(&sha256_ctx);
tc_sha256_update(&sha256_ctx, rx_packet.payload.data, 64);
tc_sha256_final(hash, &sha256_ctx);

tc_ctr_mode(out, 0x40, rx_packet.payload.data, 0x40, ctr, &aes_key);
print_info("%s>%.64s\n", fmts[i], out);
```

REPLACE

Strengths

- Hashed token

Weaknesses

- Could not get validation to function

mitre - application_processor.cpp

```cpp
tc_sha256_init(&sha256_ctx);
tc_sha256_update(&sha256_ctx, buf, 16);
tc_sha256_final(hash, &sha256_ctx);

if (memcmp(hash, REPLACEMENT_HASH, 32) == 0) {
    return error_t::SUCCESS;
} else {
    return error_t::ERROR;
}
```

BOOT

Strengths

- Secure key exchange protocol for secure send
- TRNG engine
- Mutual signature validation

mitre - component.cpp

```
// Packet checks and re-construction omitted for brevity.

uECC_make_key(public_key, private_key, uECC_secp256r1());
uECC_shared_secret(rx_packet.payload.material, private_key, shared_secret,
                   uECC_secp256r1());

tc_sha256_init(&sha256_ctx);
tc_sha256_update(&sha256_ctx, shared_secret, 32);
tc_sha256_final(hash, &sha256_ctx);

memcpy(ctr, "\x00X\xDA\xCC\x00X\xDA\xCC", 8);
memcpy(&ctr[8], &hash[16], 0x8); // AES-128-CTR nonce utilized secure send and receive
memcpy(aes_key, hash, 16);       // AES-128-CTR key
```

Weaknesses

- Only 1 key pair
- Fault injection during RNG

mitre - application_processor.cpp

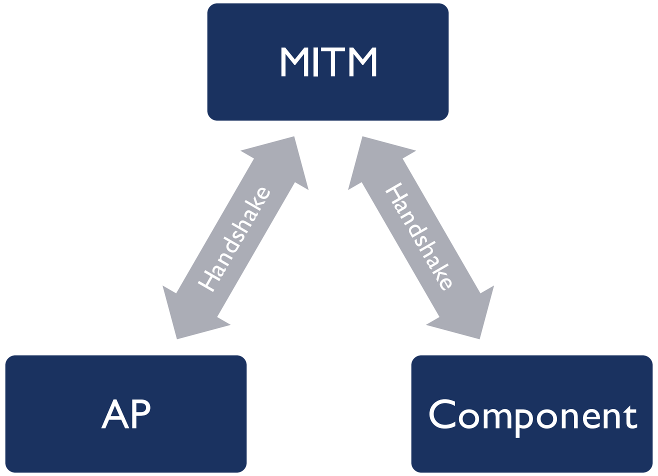
```
random_bytes(tx_packet.payload.data, 0x20);

if (uECC_sign(BOOT_A_PRIV, tx_packet.payload.data, 0x20,
              tx_packet.payload.sig, uECC_secp256r1()) != 1) {
    return error_t::ERROR;
}

tc_sha256_init(&sha256_ctx);
tc_sha256_update(&sha256_ctx, rx_packet.payload.data, 0x40);
tc_sha256_final(hash, &sha256_ctx);

if (uECC_verify(BOOT_C_PUB, tx_packet.payload.data, 0x20,
                rx_packet.payload.sig, uECC_secp256r1()) != 1) {
    // Invalid signature
    return error_t::ERROR;
}
```

0xDACC

SECURE SEND & RECEIVE

Strengths

- Ephemeral keys prevent replay across boots ***(Authenticity)***
- Nonces prevent replay during sessions ***(Authenticity)***
- HMAC to prevent modification ***(Integrity)***
- AES encryption to prevent reading ***(Confidentiality)***



Weaknesses

- No MITM protection during KEX
- Leaked HMAC key + MITM = Full compromise

mitre - application_processor.cpp

```cpp
// Send a packet to the specified component
const uint8_t index = addr_to_idx(address);

payload[0] = static_cast<uint8_t>(packet_magic_t::DECRYPTED);
payload[1] = len;
memcpy(&payload[2], &nonces[index], 0x04);
memcpy(&payload[6], buffer, len);

tc_hmac_init(&hmac_ctx);
tc_hmac_set_key(&hmac_ctx, HMAC_KEY, 32);
tc_hmac_update(&hmac_ctx, &payload[0], sizeof(payload) - 32);
tc_hmac_final(hmac, 32, &hmac_ctx);
memcpy(&payload[sizeof(payload) - 32], hmac, 32);

tc_aes128_set_encrypt_key(&aes_key, aes_keys[index]);
tc_ctr_mode(reinterpret_cast<uint8_t *>(&tx_packet.payload),
            sizeof(tx_packet.payload), payload, sizeof(payload),
            ctrs[index], &aes_key);

++nonces[index];
```

0xDACC

NOT SO RANDOM RAND()

Problem

- g_nKey is used to encrypt all communications
- time(NULL) returns -1 without implementation
- srand(time(NULL)) has a deterministic output
- rand() is not a CSPRNG

mitre -

```
srand(time(NULL));
bzero(g_nKey, BLOCK_SIZE);
for(int i = 0; i < KEY_SIZE;i++)
    g_nKey[i] = rand() % 256;
```

Solution

- Use onboard TRNG hardware
- Remove all rand() based code

mitre -

```
bzero(g_nKey, BLOCK_SIZE);
MXC_TRNG_Init();
MXC_TRNG_Random(g_nKey, KEY_SIZE);
MXC_TRNG_Shutdown();
```

BINARY LEAK

- Exploit binary leaked in public channel "uccon_supply_dump.img"
- Written in Rust
- Exploits Mitre-provided I²C Peripheral library
- Unsuccessful RE due to lack of time

FINAL COMMENTS

- Compile-time secrets
- Memory corruption
- Embedded hardware
- Read the documentation

QUESTIONS?

HTTPS://GITHUB.COM/0XDACC/2024-MITRE-ECTF-DACC.GIT

HTTPS://WWW.LINKEDIN.COM/IN/ANDREWLANGAN/

HTTPS://WWW.LINKEDIN.COM/IN/SAM-GOODMAN-CYB/

HTTPS://WWW.LINKEDIN.COM/IN/BEAUSCHWAB26/

BREAK
11:15AM-11:25AM

Restrooms, Refreshments

See you soon!

Keynote

#eCTF2024

WELCOME

Stephanie Lin

Director, Microelectronics Commons

National Security Technology Accelerator

NSTXL

NATIONAL SECURITY TECHNOLOGY ACCELERATOR

MITRE ENGENUITY

Today's Presentation Agenda

#eCTF2024

- 10:45 University of Illinois Urbana-Champaign
- 11:00 Delaware Area Career Center
- **11:15 BREAK**
- 11:25 A Word from NSTXL
- 11:35 Purdue University
- 11:50 Michigan State University
- 12:05 University of California, Irvine
- **12:20 LUNCH / NETWORK**
- 1:20 University at Buffalo
- 1:35 Carnegie Mellon University
- 1:50 A Word from Fortinet
- 2:05 Award Presentation
- 2:20 Closing Remarks / Student Dismissal

Purdue University

#eCTF2024

Welcome
b01lers

Currently 3rd place with 24,501 points

b01lers

Purdue University

Jacob White
Jack Roscoe
Gabriel Samide
Jihun Hwang (Jimmy)
Kevin Yu
Phillip Frey

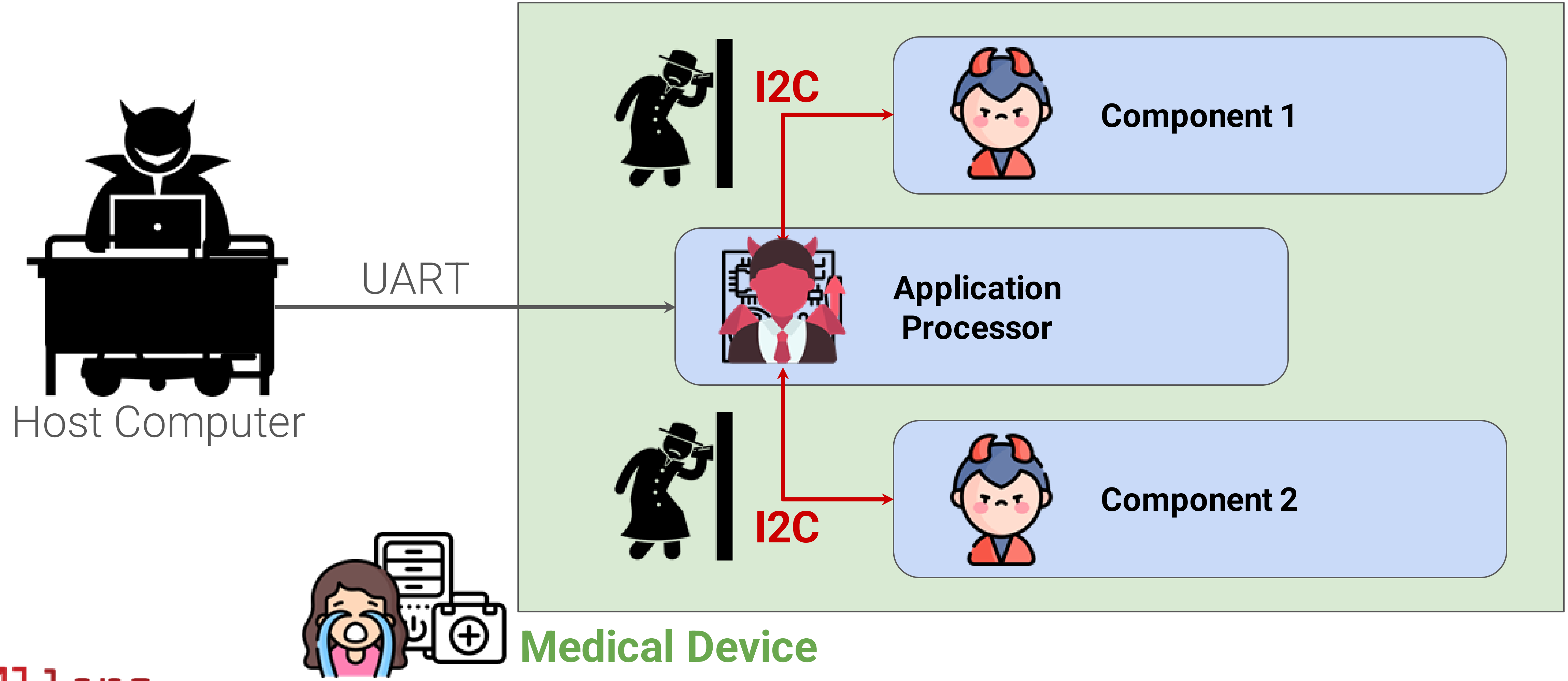
b01lers... Assemble!

Nick Andry, Philip Frey, Jorge Hernandez, Neil Van Eikema Hommes, Jihun Hwang, Siddharth Muralee, Jaxson Pahukula, Mihir Patil, Adrian Persaud, Jack Roscoe, Gabriel Samide, Lucas Tan, Vinh Pham Ngoc Thanh, Vivan Tiwari, Jacob White, Susan Wu, Kevin Yu

Advised By: Professors Christina Garman, Kazem Taram, Santiago Torres-Arias

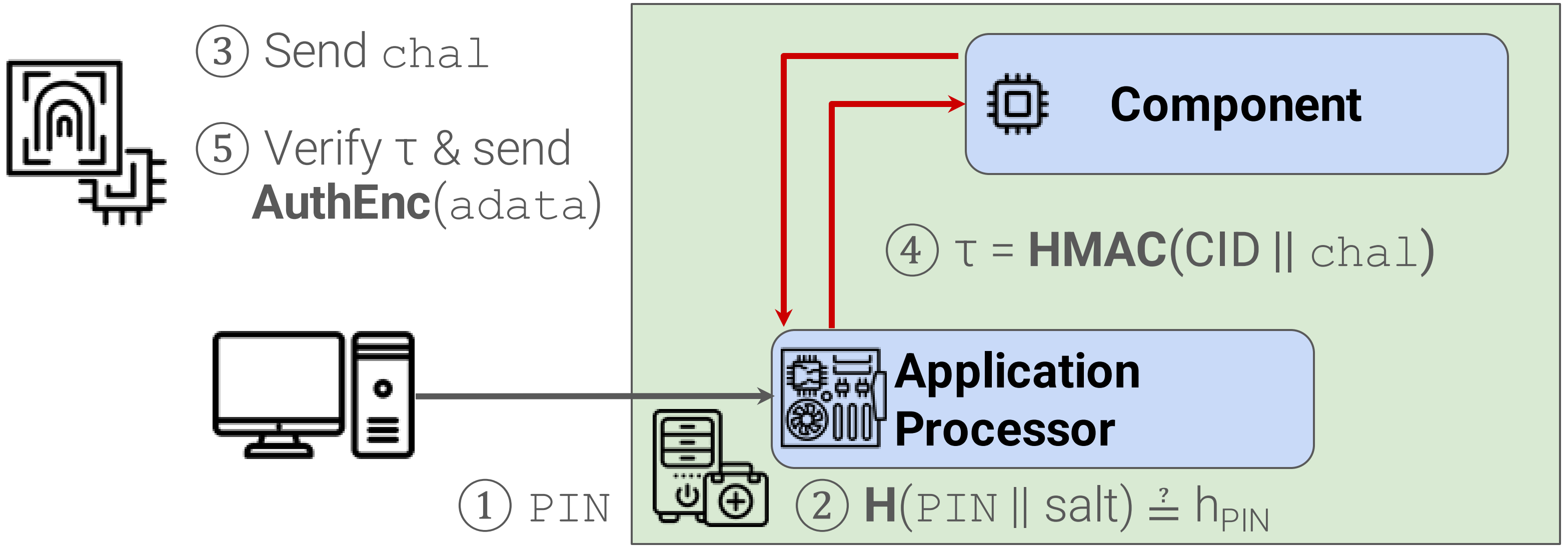
Design Phase

Design Goals & Overview



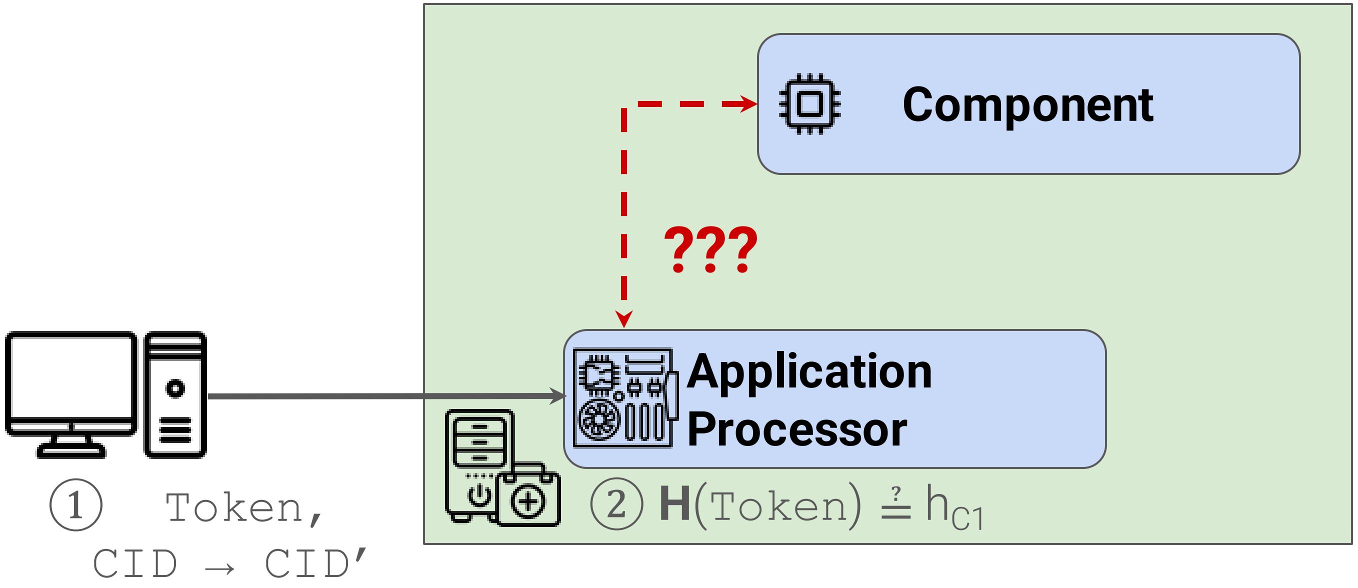
Attest Protocol

- Hash & Check PIN (**0x000000–0xFFFFFF**) like a password.
- Component challenges the AP to respond with expected ID.
- Component encrypts attest data for valid AP to decrypt.



Replace Protocol

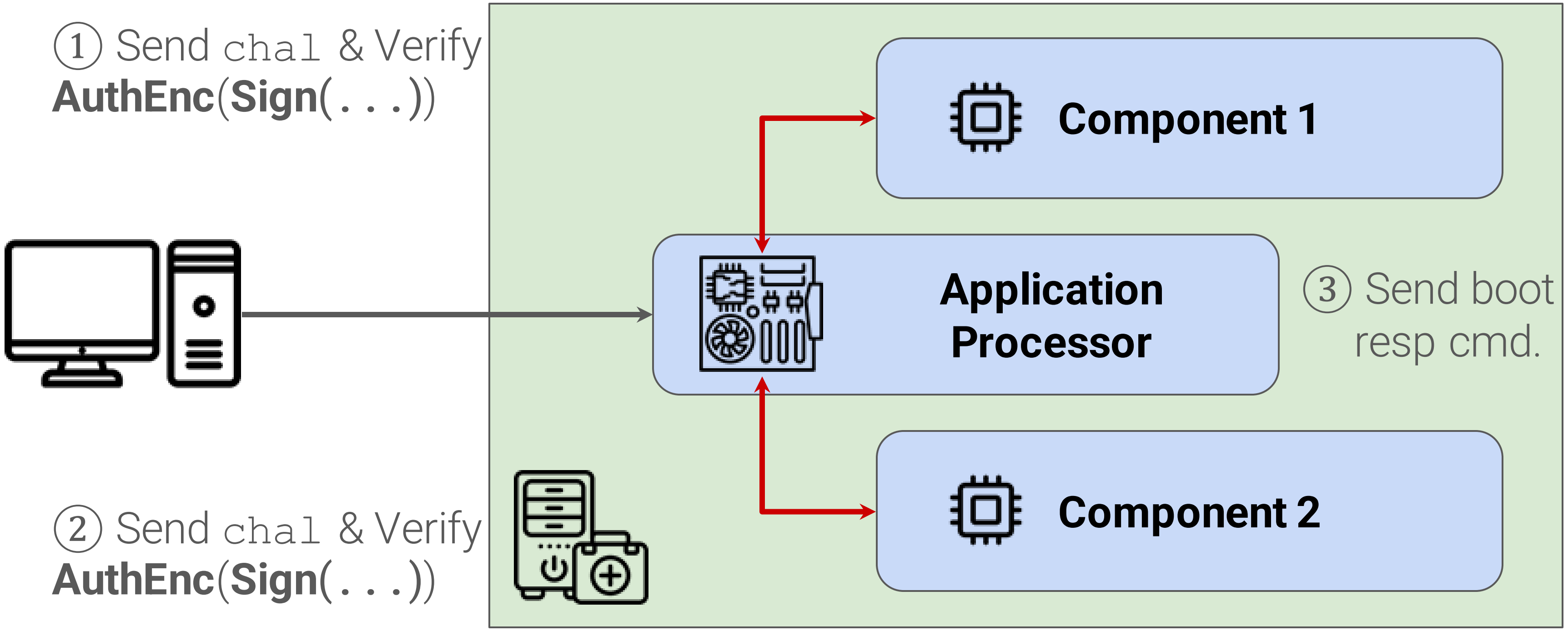
- Hash & Check **Replacement Token (16 B!)** like a password…



b01lers

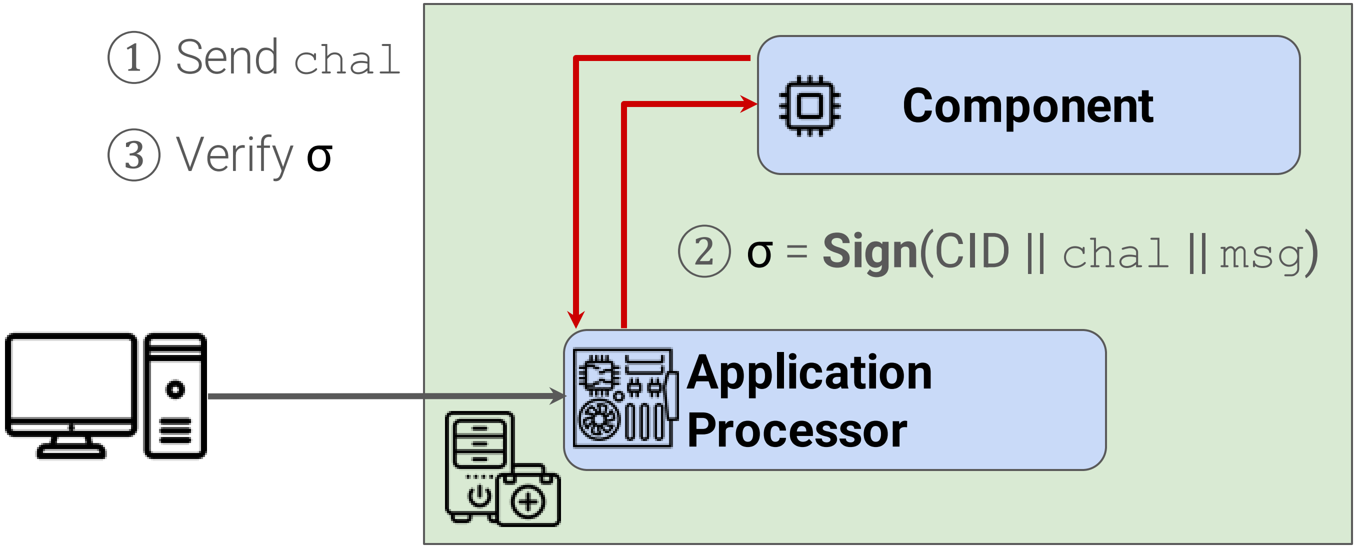
Boot Protocol

- Components and AP challenge each other to check who they say they are.
- AP and Components wait to verify each other and respond before booting.



Messaging Protocol

- Generates random challenge to include with signed message.
- Signatures verify the specific identity of Component or AP's messages.
  - Actuator shouldn't pretend to be an insulin pump.
  - Components shouldn't pretend to be AP.



Security Features

Passwords (PINs)

- Salted and hashed with a “slow” hash function (`bcrypt`).
- Defends against both offline and online dictionary attacks.

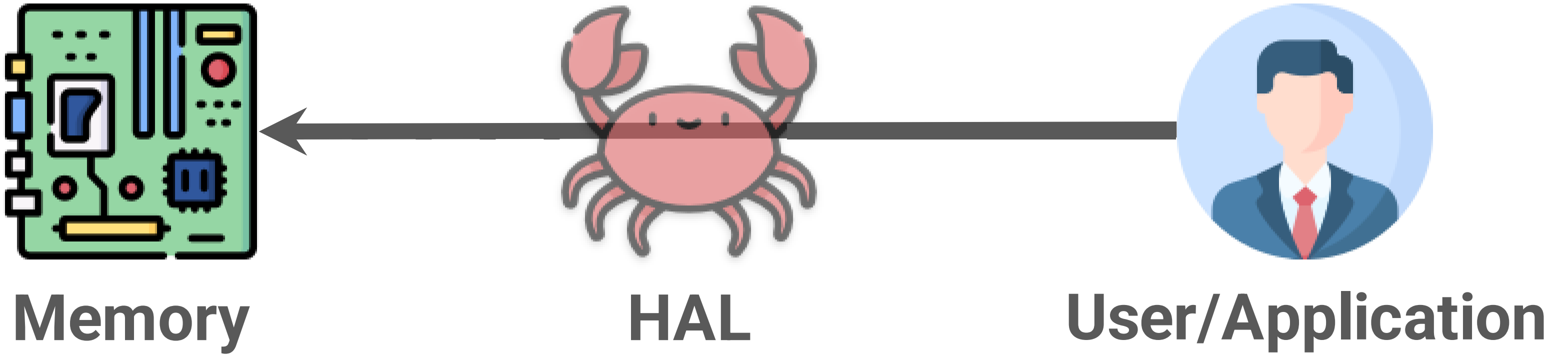
`bcrypt` (

)

Security Features (Cont.)

Hardware Abstraction Layer (HAL)

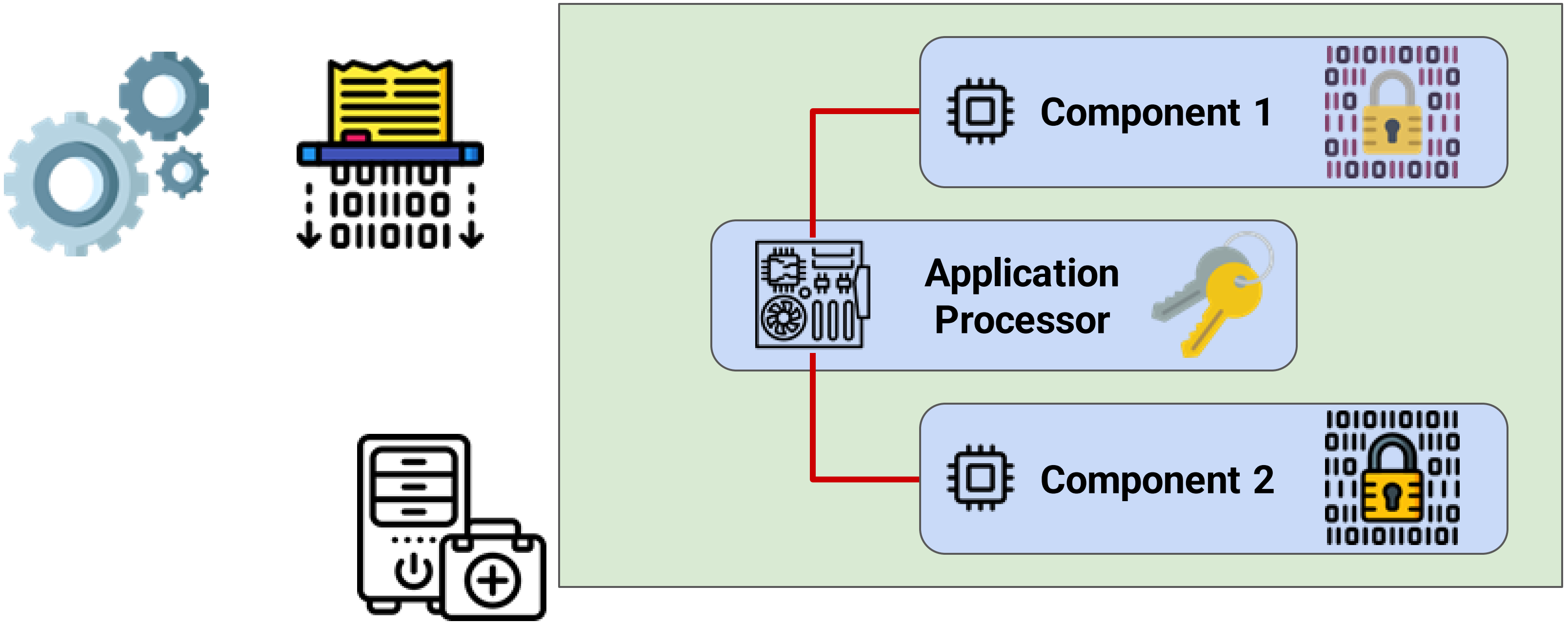
- Re-wrote firmware in Rust.
- Memory safe by default!



Security Features (Cont.)

Build-time Encrypted Data

- Attestation Key is only stored on the AP, not on Components.
- Attestation Data is not directly stored anywhere inside.

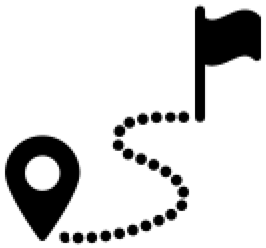


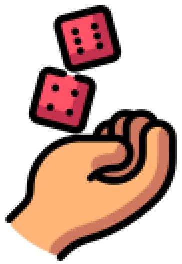
b01lers

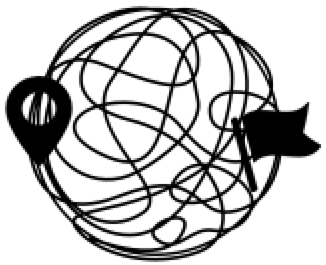
Security Features (Cont.)

Compile-time address randomization (ASLR)

- Randomized memory section offsets.
- Frustrates buffer overflow or similar attacks.



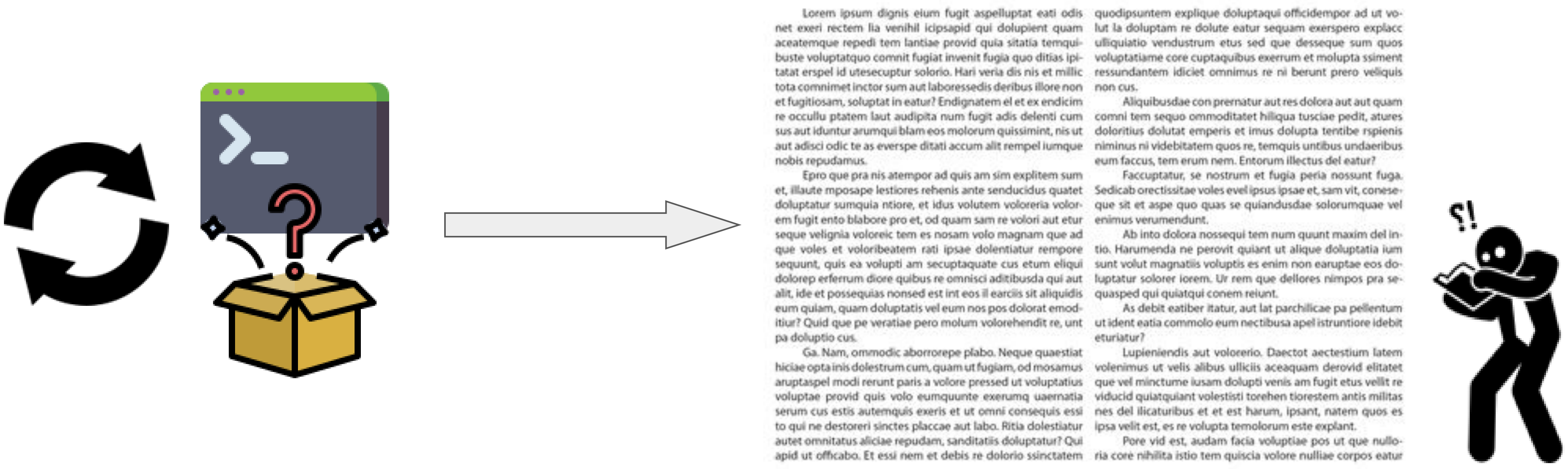




Security Features (Cont.)

Random Delays and Repeated Checks

- Protection against fault injection, timing and glitching attacks, etc.



Design Summary

Potential Improvements

- Require secret sharing from all components & AP to boot.
- Two way C-R and sequence numbers for post-boot.
- Activate the board's Memory Protection Unit (MPU).
- Use stronger memory-hard hashes (e.g. Argon2id) [1] for Attest PIN.

[1] Jeremiah Blocki, Benjamin Harsha, and Samson Zhou. "On the economics of offline password cracking." 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018.

Attack Phase

I2C Interrupt Handler Buffer Overflow

The interrupt handler did not properly handle repeated restart.

- Only checks if the stop flag is set for executing end of transaction code.
  - … which is only set when I2C stop code transmitted!
- A repeated restart message can start a transaction without sending an I2C stop code.

```
// Transaction over interrupt
if (Flags & MXC_F_I2C_INTFL0_STOP) {
```

I2C Exploit (Cont.) Transaction Threshold

```
// RX fifo Threshold Met on Write
        RX_THD) {
        ster before writing data so select register
        {
        C_INTERFACE, (volatile unsigned char*) &ACTIVE_REG, 1);

    }
    // Read remaining data
    if (ACTIVE_REG <= MAX_REG) {
        int available = MXC_I2C_Ge                         CE);
        if (available < (I2C_REGS_                         ) {
                          I2C_ReadRXFIFO(I2C_INTERFACE,
                          E_REG][WRITE_INDEX],
                          FOAvailable(I2C_INTERFACE));

        else {
            WRITE_INDEX += MXC_I2C_ReadRXFIFO(I2C_INTERFACE,
                &I2C_REGS[ACTIVE_REG][WRITE_INDEX],
                I2C_REGS_LEN[ACTIVE_REG]-WRITE_INDEX);
        }
```

1. Transaction start sets `WRITE_START` register to be TRUE

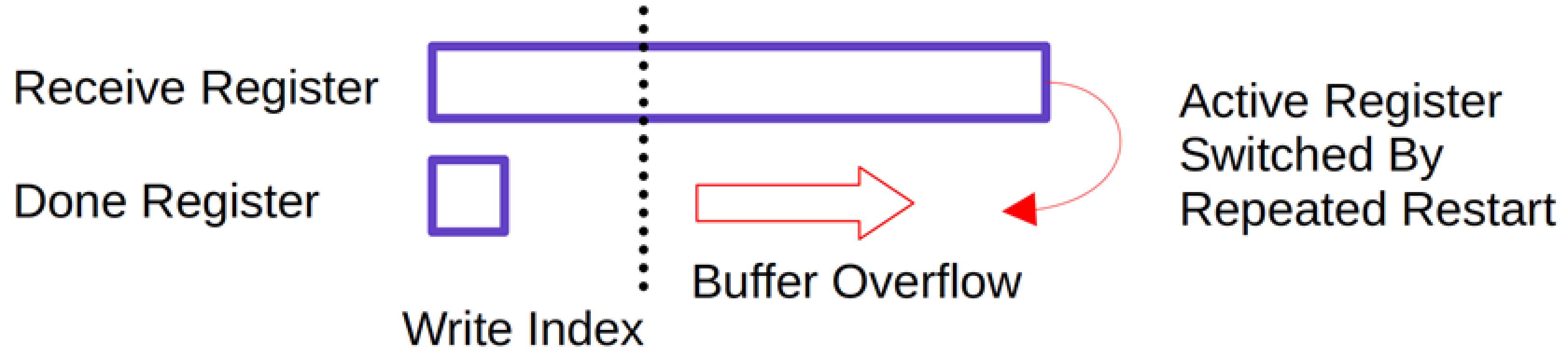
2. Active register is set when `WRITE_START` is TRUE.
This allows active register to be switched mid transaction

3. If `WRITE_INDEX` > active register size large buffer overflow is possible`

I2C Exploit Details

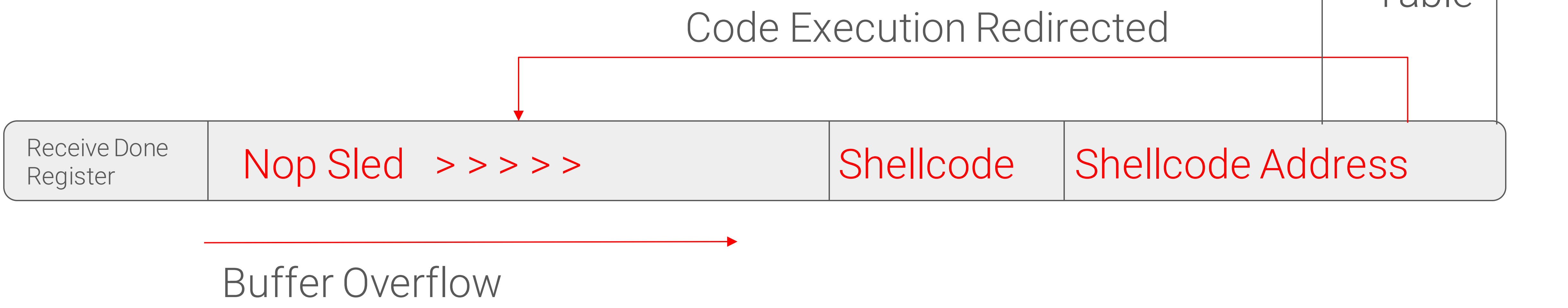
Buffer overflow -> Memory Corruption

- Write bytes, switch from a large register to a small register with repeated restart.
- **`WRITE_INDEX` will be out of bounds!**
- A large buffer overflow occurs, and bytes can be written in from I2C in many repeated restarts
  - Pointer in interrupt vector table overwritten to jump to shellcode



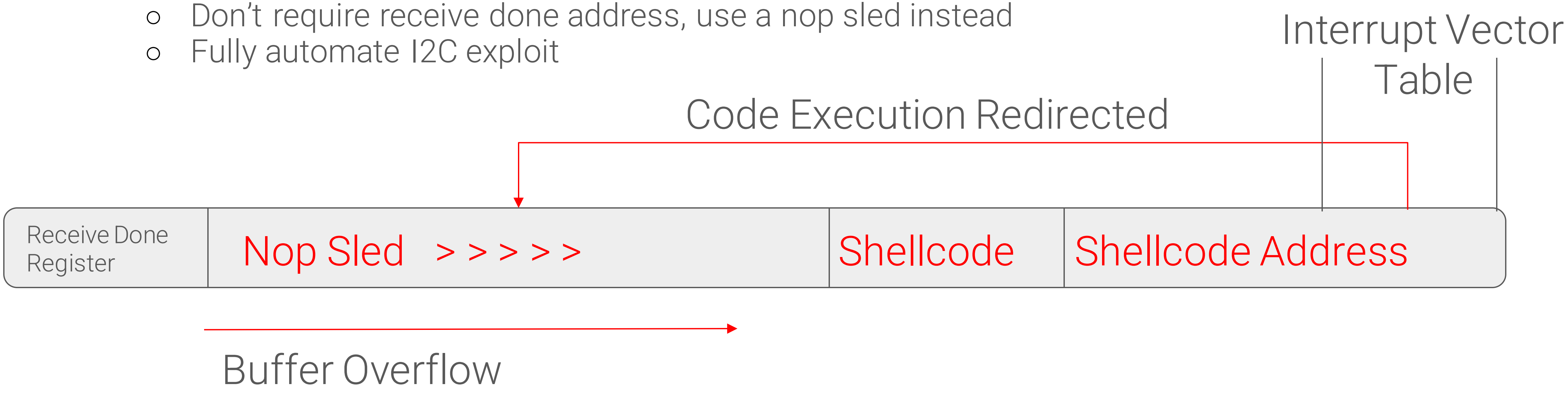
How We Used I2C Exploit

- Attacks with Physical Access
  - Python scripts communicate with malicious AP and shellcode on component to retrieve flash dump
  - Flags and keys can be retrieved from flash dump
- Supply Chain Flags
  - Shellcode running on the component will dump its flash over i2c
  - Malicious AP will receive this dump and print it out base64 encoded



Potential Improvements to I2C Exploit

- Other teams were aware of the exploit, so it was a race to get first bloods
  - We could manually do the easier 4 in physical access flags in about 3-4 minutes, but other teams automated systems could do it in about one minute
- Improvements
  - Don’t require receive done address, use a nop sled instead
  - Fully automate I2C exploit



b01lers

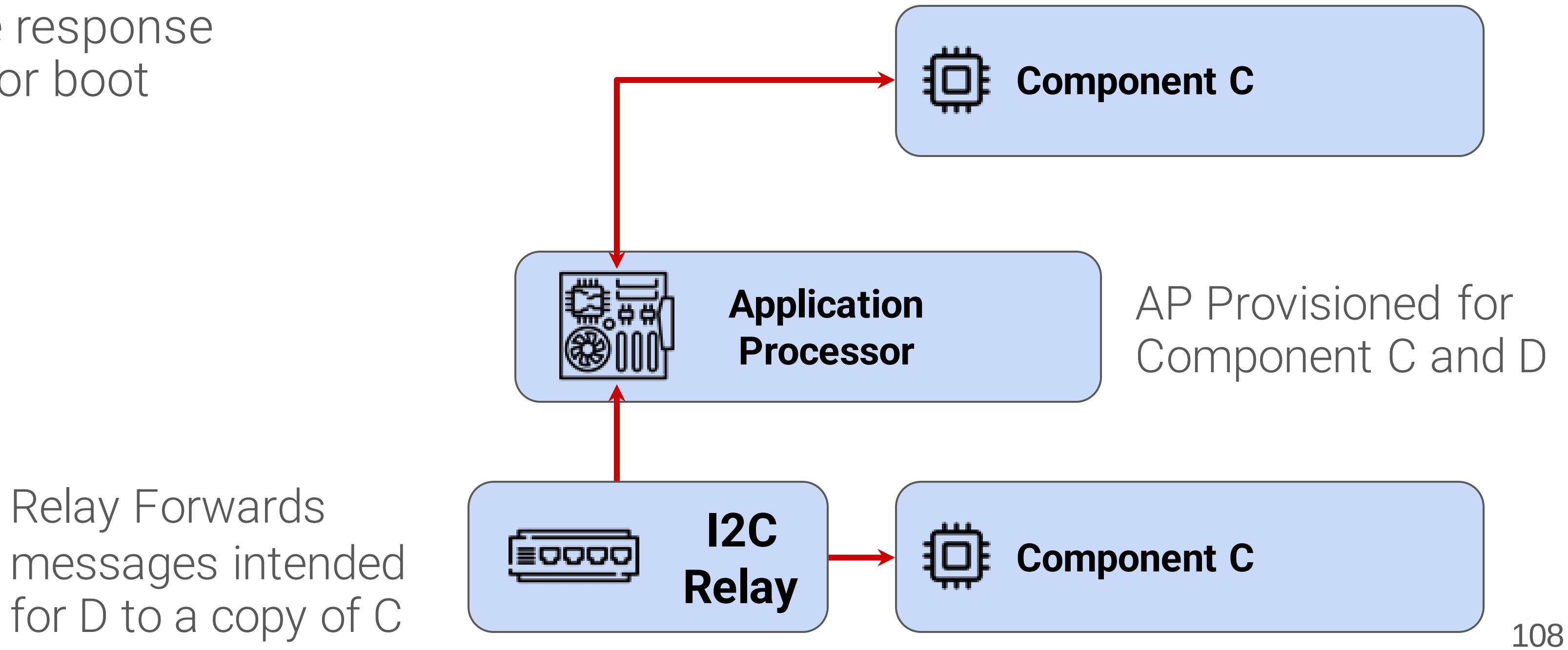
Attack Impacts and Countermeasures

- Potential Impact
  - Arbitrary code execution resulting in numerous potential attacks
    - Exfiltrating attestation data, boot messages, and secret keys stored in the flash, modifying vital hardware registers, and manipulating intended functionality

- Suggested Countermeasures
  - Reset the read and write indexes are reset even after a repeated restart
  - Ensure out of bounds writes are not possible
  - Redesign the interrupt handler

Exploiting Protocol Flaws

- Static token to authenticate AP and components
  - Some teams had a secret token used to authenticate the AP sent in plaintext
  - The token could be received by a malicious component, then replayed by a malicious AP
- Boot authentication does not include component ID
  - e.g. in challenge response authentication for boot



b0llers

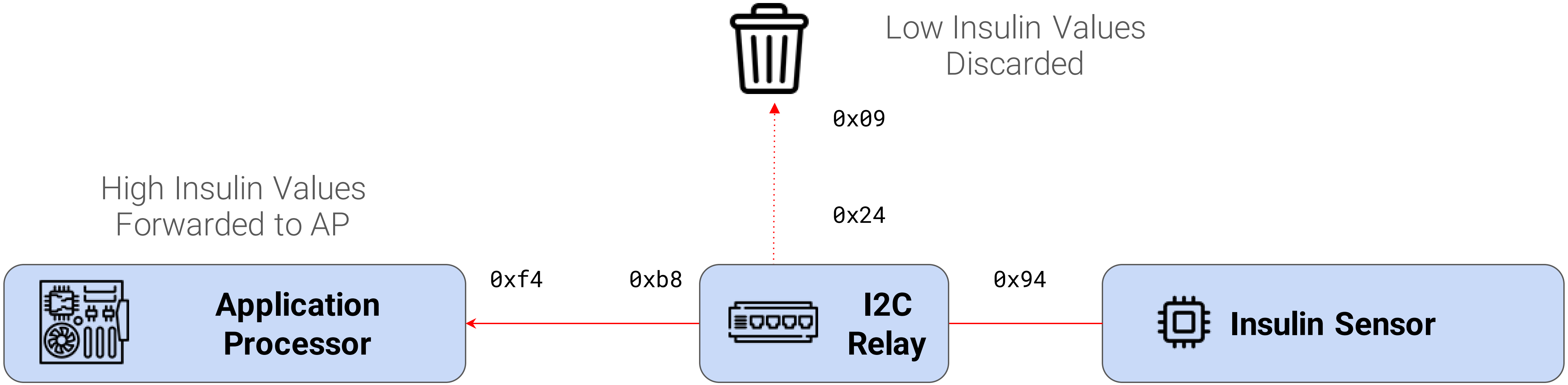
Protocol Flaws Countermeasures

- Potential Impacts
  - Attacker can maliciously impersonate devices and perform operations only authorized devices should be able to perform
  - Includes operations such as booting MISC system and querying attestation data without the need for authentication

- Countermeasures
  - Utilize challenge response instead of just a fixed token
  - Challenge response should include the component ids in some form

Other Attacks

- We investigated possibility of dropping packets in operational pump swap
  - Drop the low insulin reading packets, and only forward the high packets
    - Several teams (us included) did not fully ensure post boot packets arrived in order
  - This ended up not working due to the way post boot messaging was implemented



b01lers

What We Learned

- Focus on getting infra set up
  - Having to pass around boards manually, and not having a way for everyone to work on development / exploits at the same time hindered us a lot
- Read rules more carefully
  - We had to redesign our replace / boot protocol near the end of design phase because we didn't realise that AP cannot talk to component during replace
  - We didn't realise until the very end that all flags were stored in attest data and boot message, so we can do extra things like encrypting all of them at build time and use some sort of secret sharing scheme to recover decryption keys

Seems Familiar...

Final Comments

- With more time and resources, what other things would you have done?
    - Design Phase: **Prevent fault injection attacks,** digitally sign features, randomize binary layout, compile with Checked C, thoroughly audit crypto libraries + code
    - Attack Phase: Side-channel attacks, automate common attacks
- What was the most valuable thing you learned during the competition?
    - Read the rules properly (Strategy is very important)
    - Prep infra/tools for attack phase earlier

Source: Purdue eCTF 2023 slides

Final Comments

- Improvements to be made:
  - Quickly establish threat model and outline of protocol implementation
  - Spin up (fully…) functioning development and attack infra
  - Immediately start Rusty development - What even is an MSDK?

We immensely enjoyed the competition; thank you to the MITRE organizers and eCTF sponsors for your hard work in making this event possible.

See You Next Year!

Today's Presentation Agenda

#eCTF2024

- 10:45 University of Illinois Urbana-Champaign
- 11:00 Delaware Area Career Center
- **11:15 BREAK**
- 11:25 A Word from NSTXL
- 11:35 Purdue University
- 11:50 Michigan State University
- 12:05 University of California, Irvine
- **12:20 LUNCH / NETWORK**
- 1:20 University at Buffalo
- 1:35 Carnegie Mellon University
- 1:50 A Word from Fortinet
- 2:05 Award Presentation
- 2:20 Closing Remarks / Student Dismissal

Michigan State University

#eCTF2024

Welcome
Sp4rtans

Currently 9th place with 4,483 points

MITRE eCTF 2024
Team Spartans

Michigan State University

Udbhav Saxena, Felipe Marques Allevato, Charles Selipsky, Riley Cook, Aashish Harishchandre, Aditya Chaudhari, Fatima Saad, Samay Achar, Krishna Patel, Ramisa Anjum, Radhe Patel

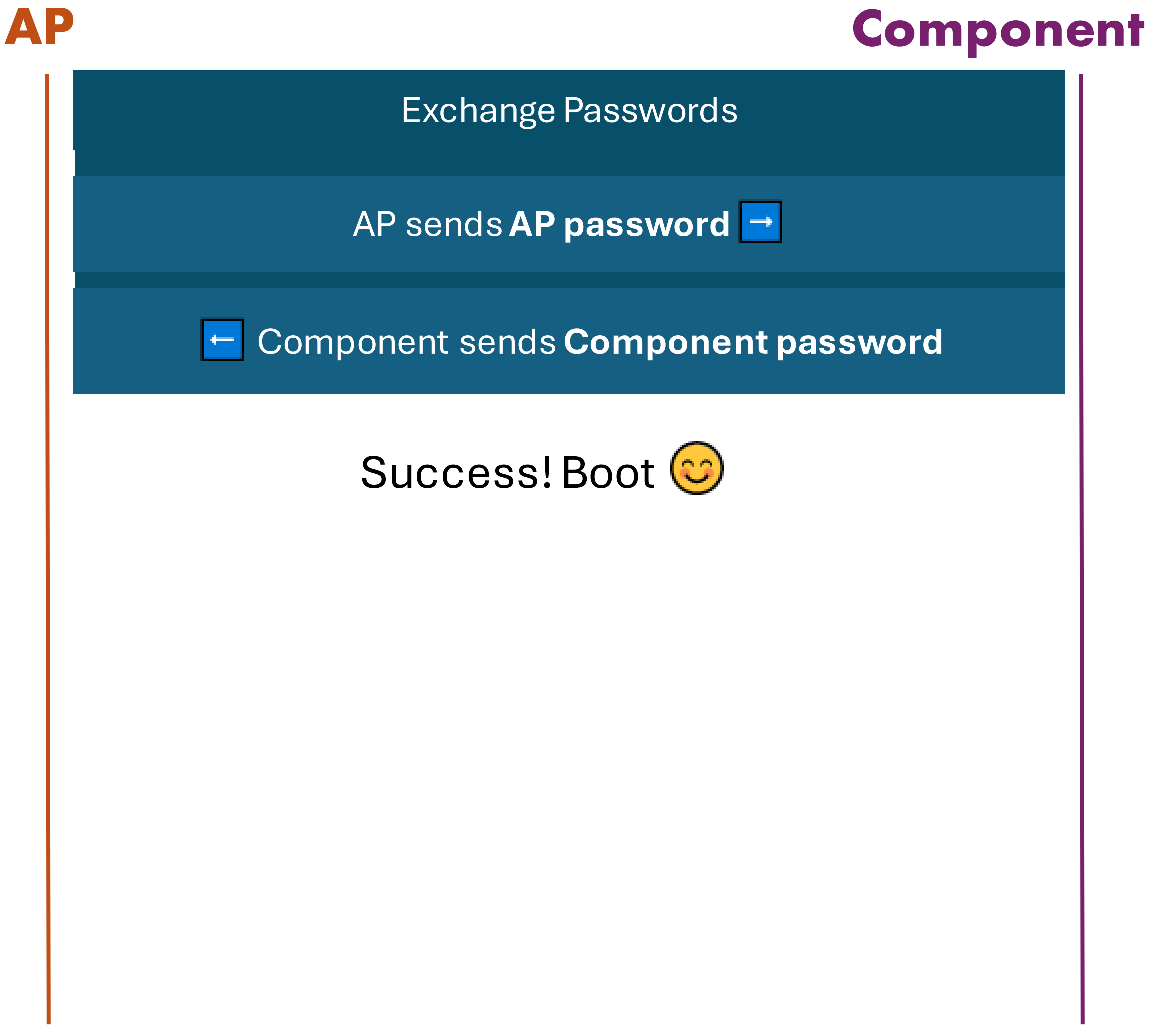
Building Up Our Design, One Step at a Time

Goal: Create a secure medical system composed of an **Application Processor (AP)** and **Components** that boots only when all devices are genuine and from the manufacturer.

After booting, create a secure channel for communication between the AP and components, ensuring the **integrity** and **authenticity** of messages.

Idea 1

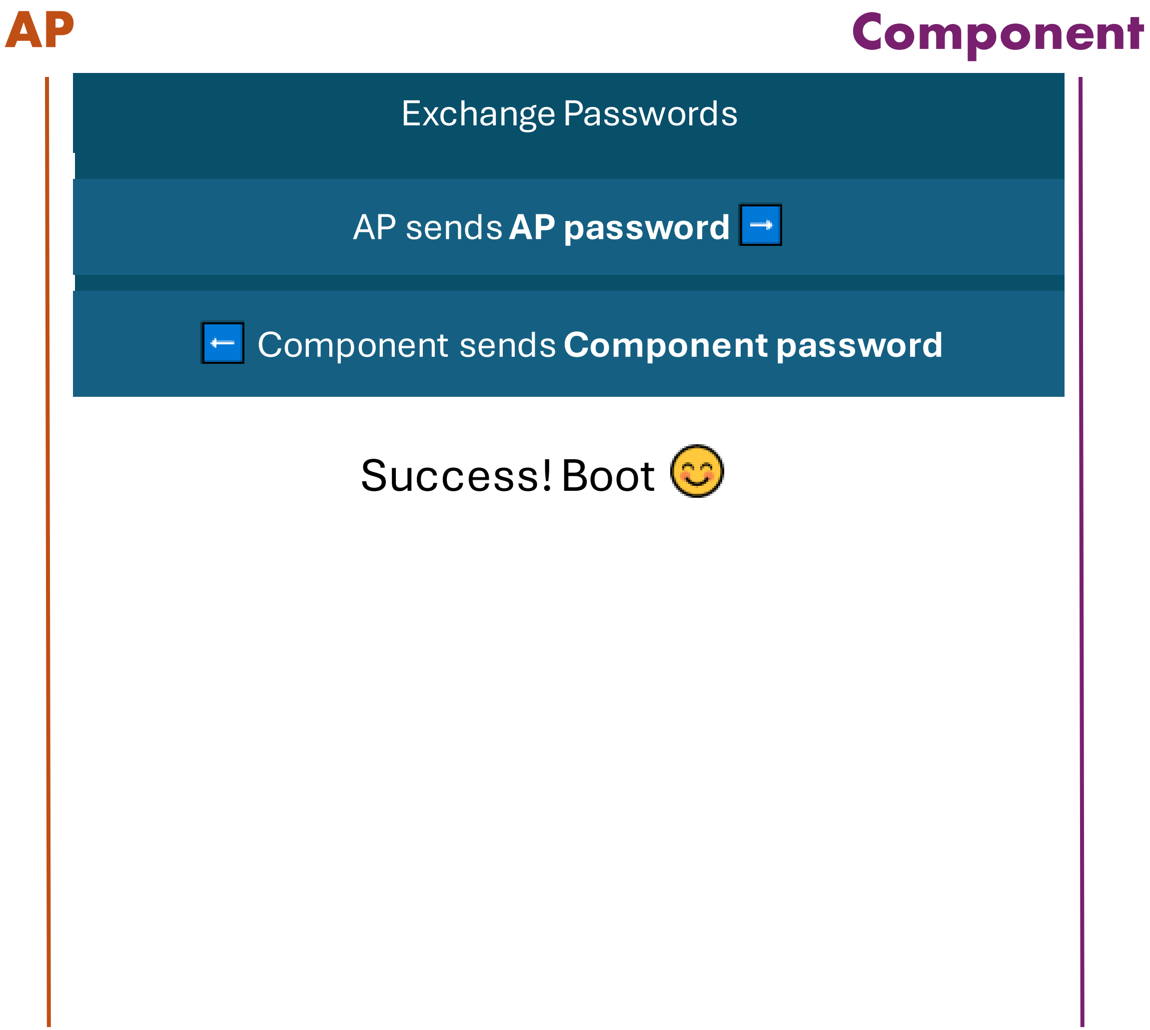
Let's use a **password** on either side, one for the AP and one for the Component. On a boot command, the AP and component simply share their password to verify each other and then boot.



Idea 1

Problem: A malicious third party can initiate a boot command with a fake component, make the AP share the password, then reuse it as a fake AP to make a component boot.

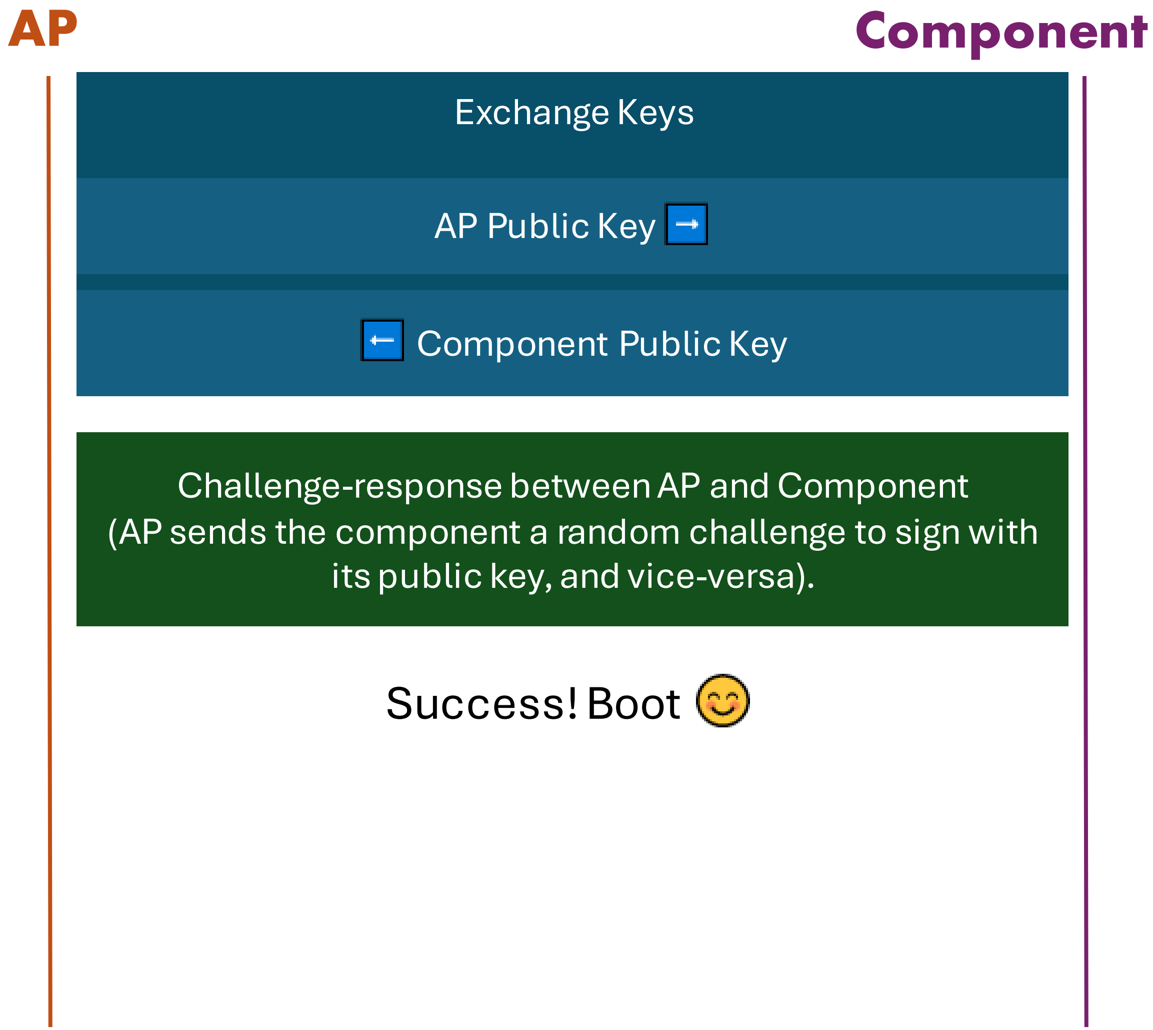
This was an attack we performed against a team that used this design.



Idea 2

Use **public key cryptography** along with **challenge-response**. Every AP and Component gets a keypair made up of a **Public Key** and a **Private Key**.

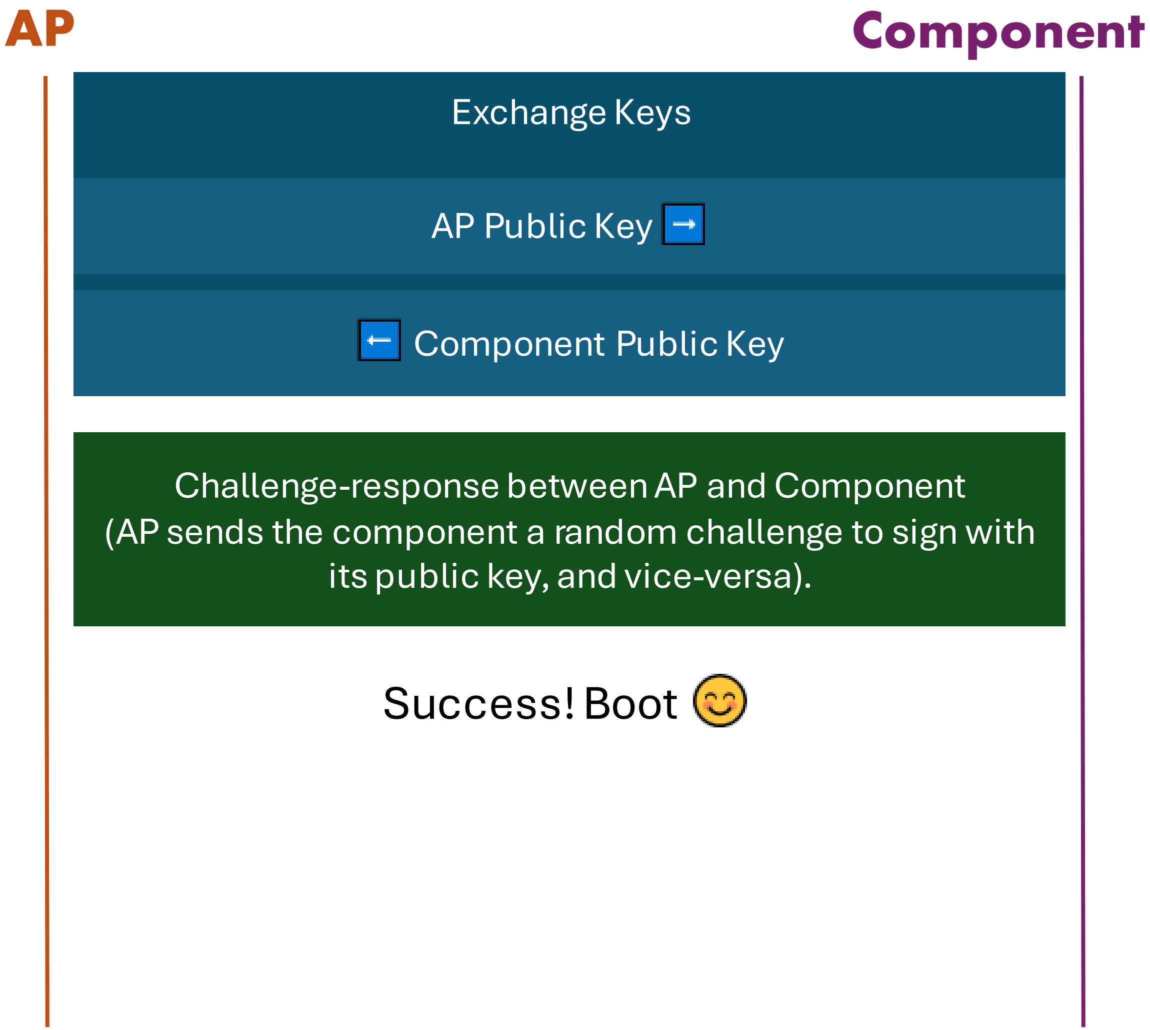
They exchange their Public Keys, and then sign a randomly generated challenge from the other party with their private key. Since the challenge is randomly generated each time, this makes sure that the other end owns the key since responses from older exchanges cannot be reused.



Idea 2

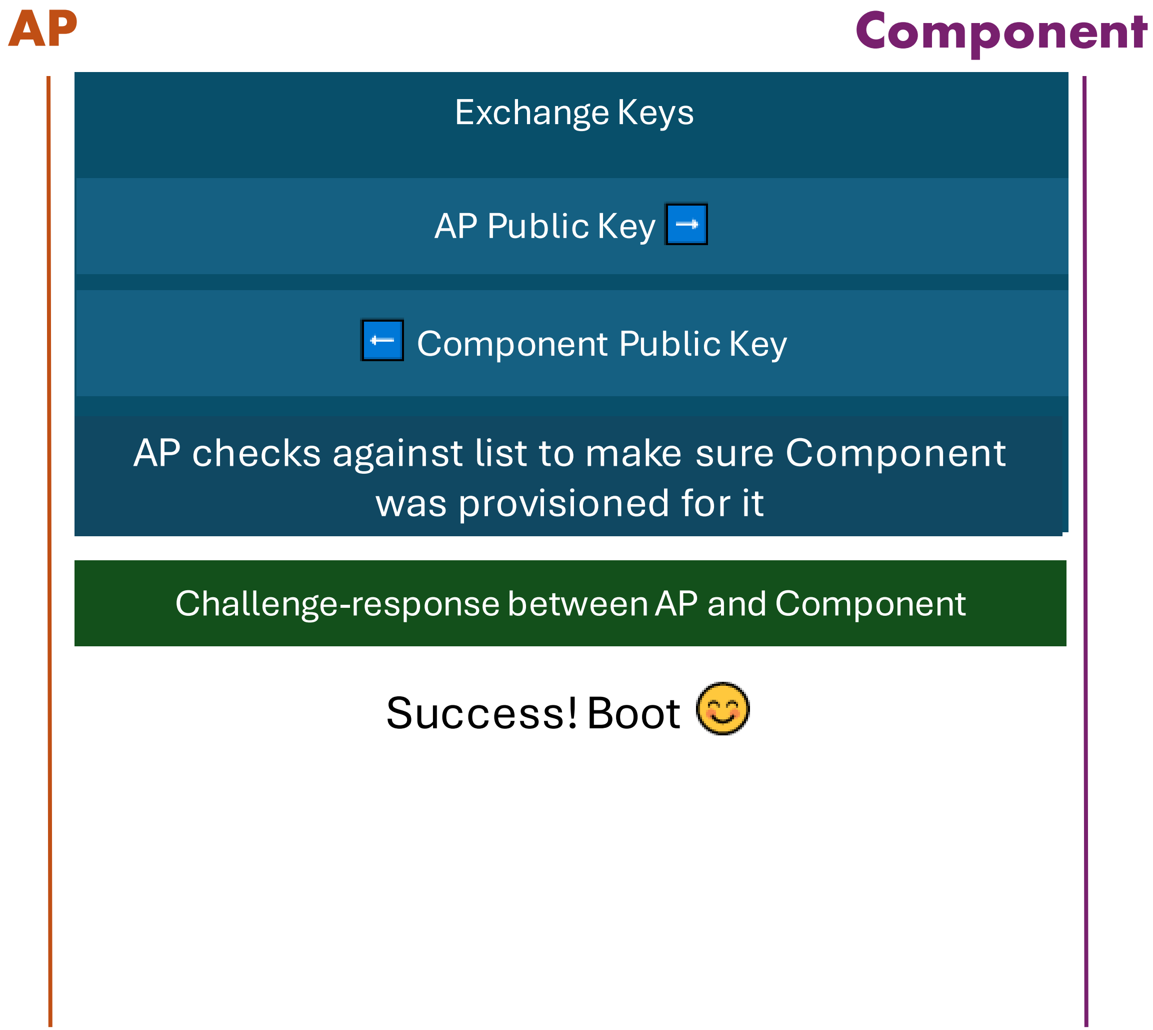
Problem: This design doesn't make sure that the APs and Components are actually from the manufacturer! Any device participating in the handshake, as long as it presents a random keypair, will cause a successful boot.

This was an attack we successfully executed against a team, and the only step required was to flash a board with their source code as-is and boot the MISC.



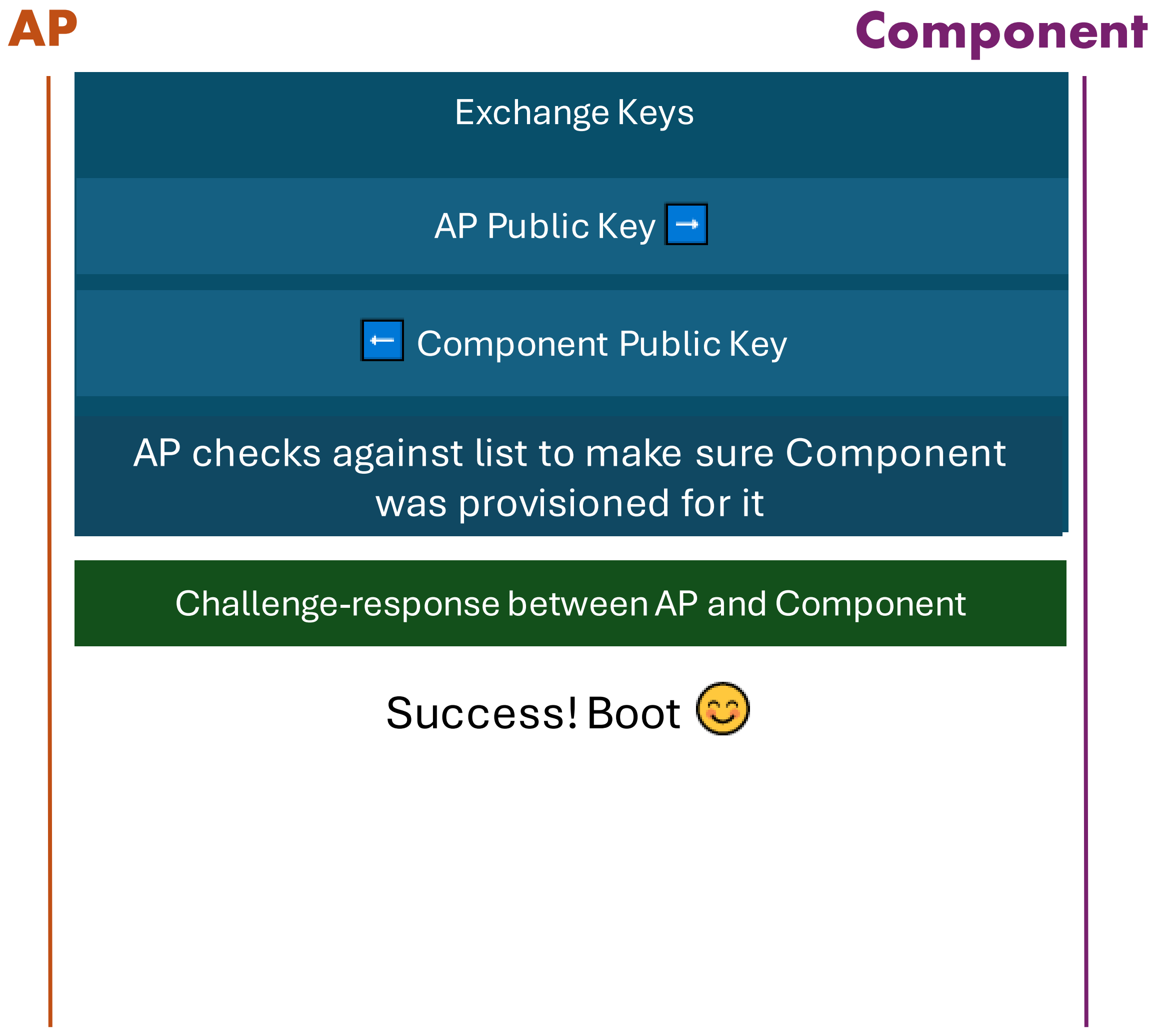
Idea 3

Have the AP store the public keys of all provisioned components when built, so that on a boot attempt it can check the provided key against the list and make sure that it is legitimate.



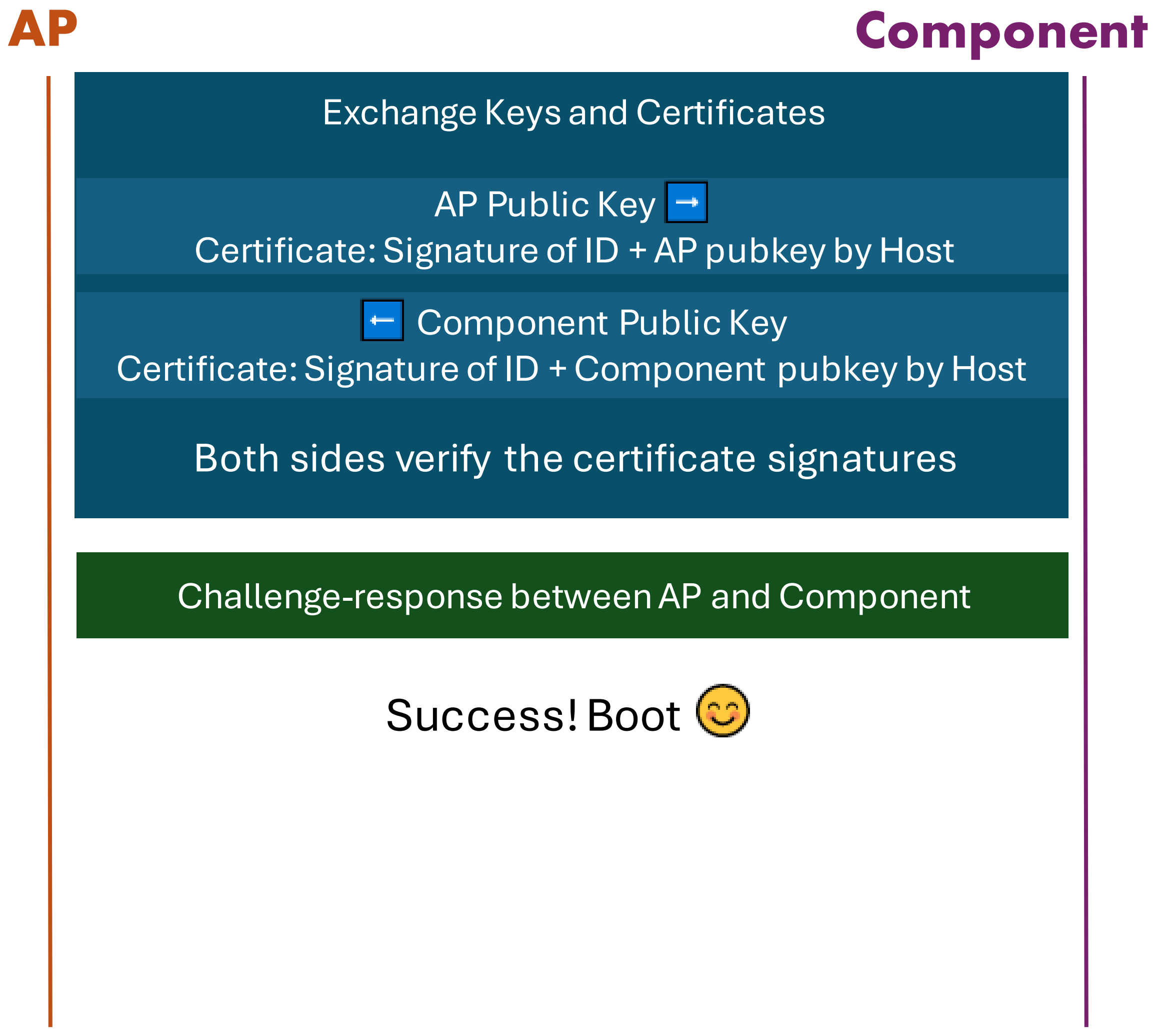
Idea 3

Problem: This design is not possible with the infrastructure we have in the competition! A MISC can be built in any order, and an application processor can have an arbitrary number of components built for it even after the AP has already been built. Thus, it is not possible to know all the possible component keys in advance when building the AP.



Idea 4

Use **certificates**. We can give the manufacturer (or the **host system**) its own keypair stored in the host secrets. Any AP or Component that is built will get its own keypair, but it will also get an additional packet of data which would be a combination of the device's identifier + the device's public key cryptographically signed by the host's secret key, which we call a **certificate**. Then, we also give all devices the host's public key so that they can verify these signatures before booting.



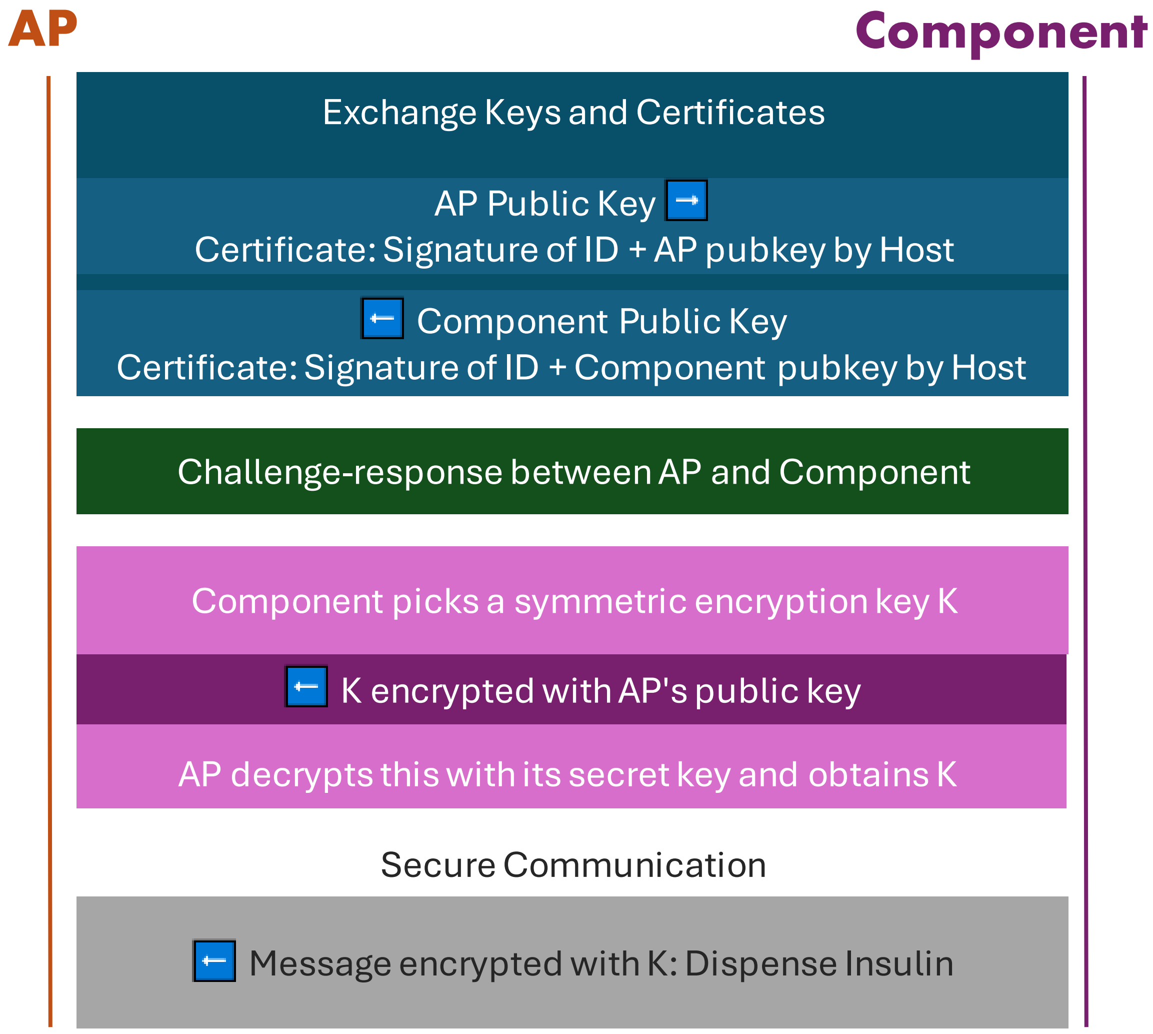
New Problem: Secure Communications

After booting, an AP needs to establish a secure channel of communication with all components where both sides can ensure **integrity** and **authenticity** of messages. Public key cryptography becomes inefficient for continuously sharing large messages, so it is preferred to do this using symmetric encryption. We can use **authenticated encryption** (ChaCha20-Poly1305 in our case) to be able to detect if message packets are tampered by a third party, in addition to providing confidentiality.

This now presents a problem of sharing a symmetric key between the devices.

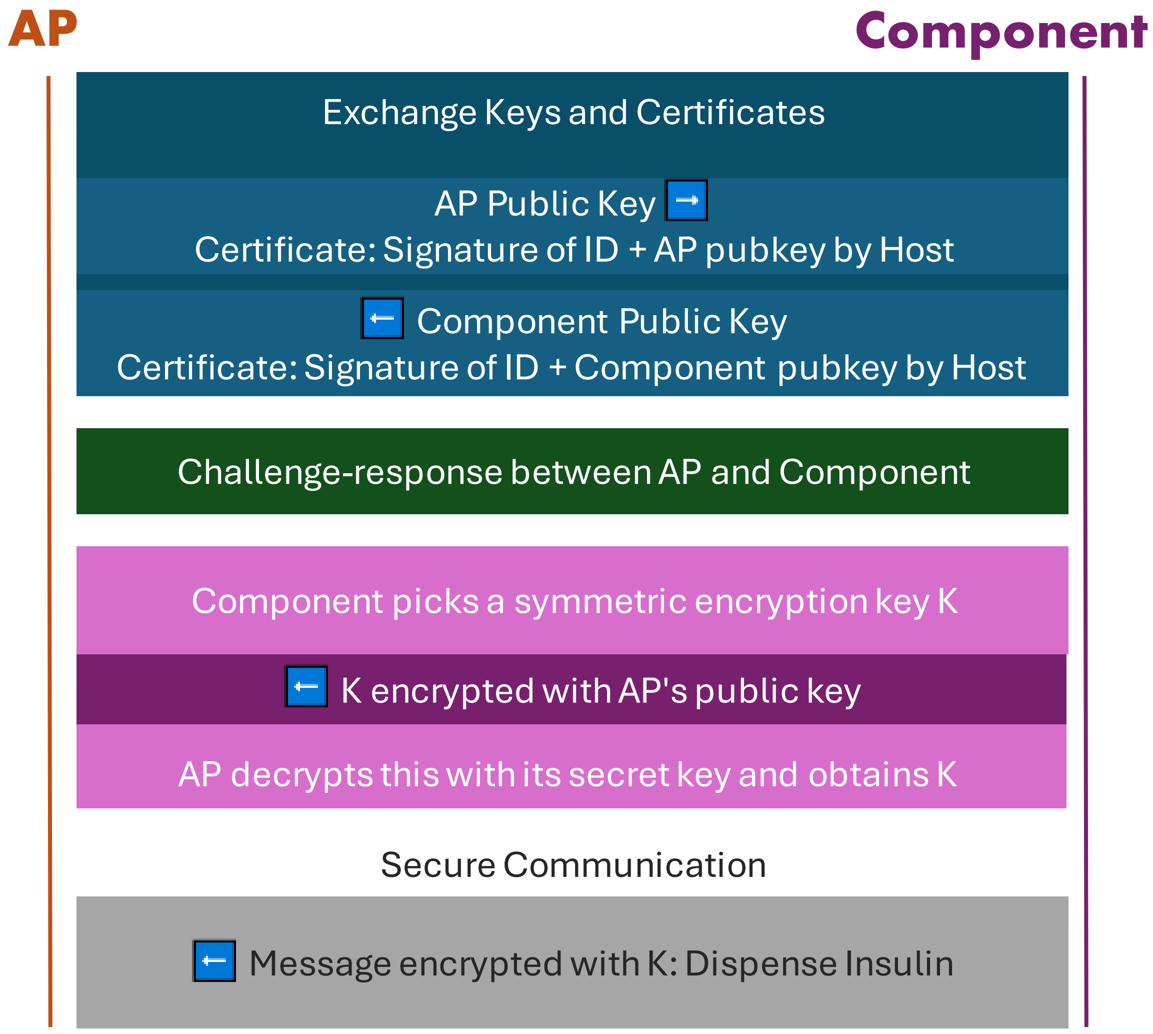
Idea 5

Since we've already verified the public keys between devices, one side (e.g., component) can simply choose a random encryption key, encrypt it using the other party's (AP's) public key, and send it across. This ensures that only the AP could decrypt this with its secret key, and then continue to use it for symmetric encryption.



Idea 5

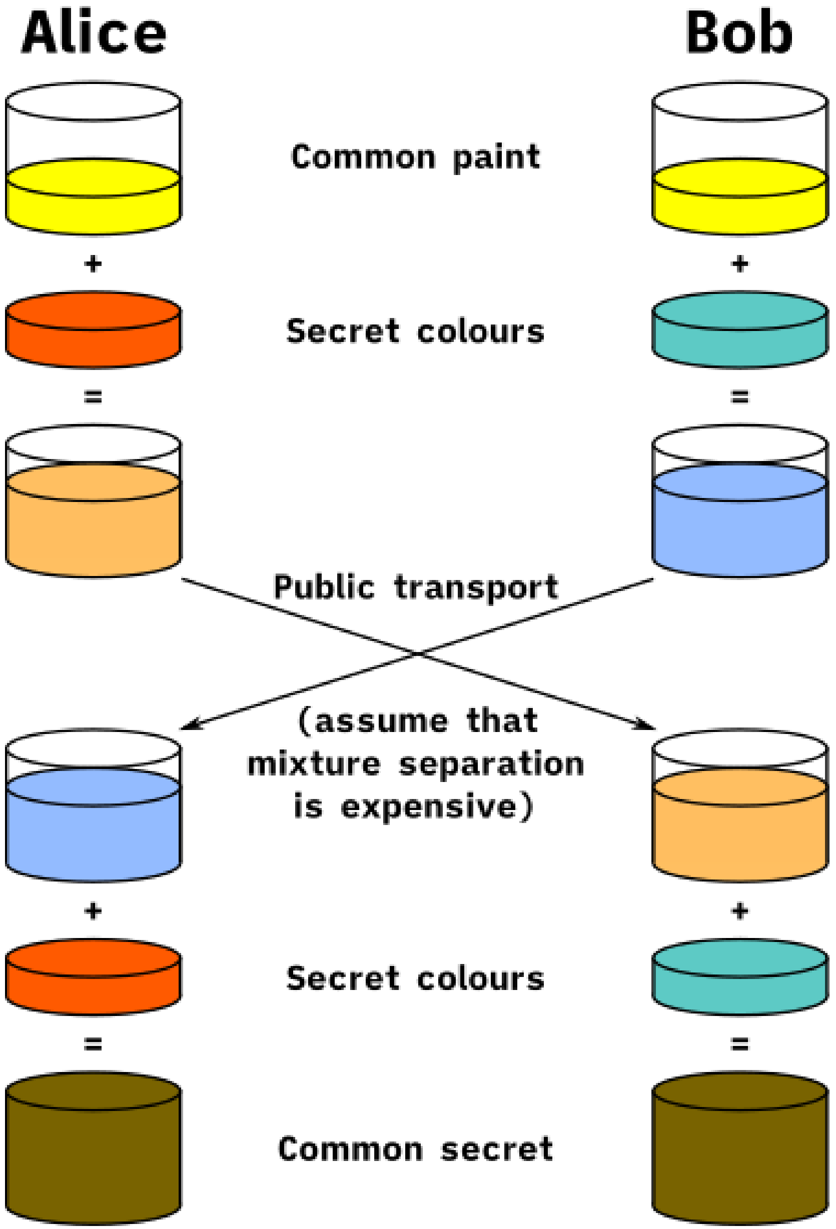
Problem: Hinges the security of all communications on the component's private key remaining a secret. An attacker could collect and store all communications between the AP and component. If there is a compromise of the AP's keys in the future, the attacker can go back and decrypt the packet sent by the component.



Idea 6

Use a **Diffie-Hellman Key Exchange** with ephemeral keys. In our design, both the AP and Component generate pairs of random **Curve25519** keys during the handshake. They then exchange the public points and combine them such that they both arrive at the same result, and no outside party observing this exchange can figure out the same key. This shared result is then used as the secret key for symmetric encryption, and the keys are then discarded after the session end (which is why they are called ephemeral).

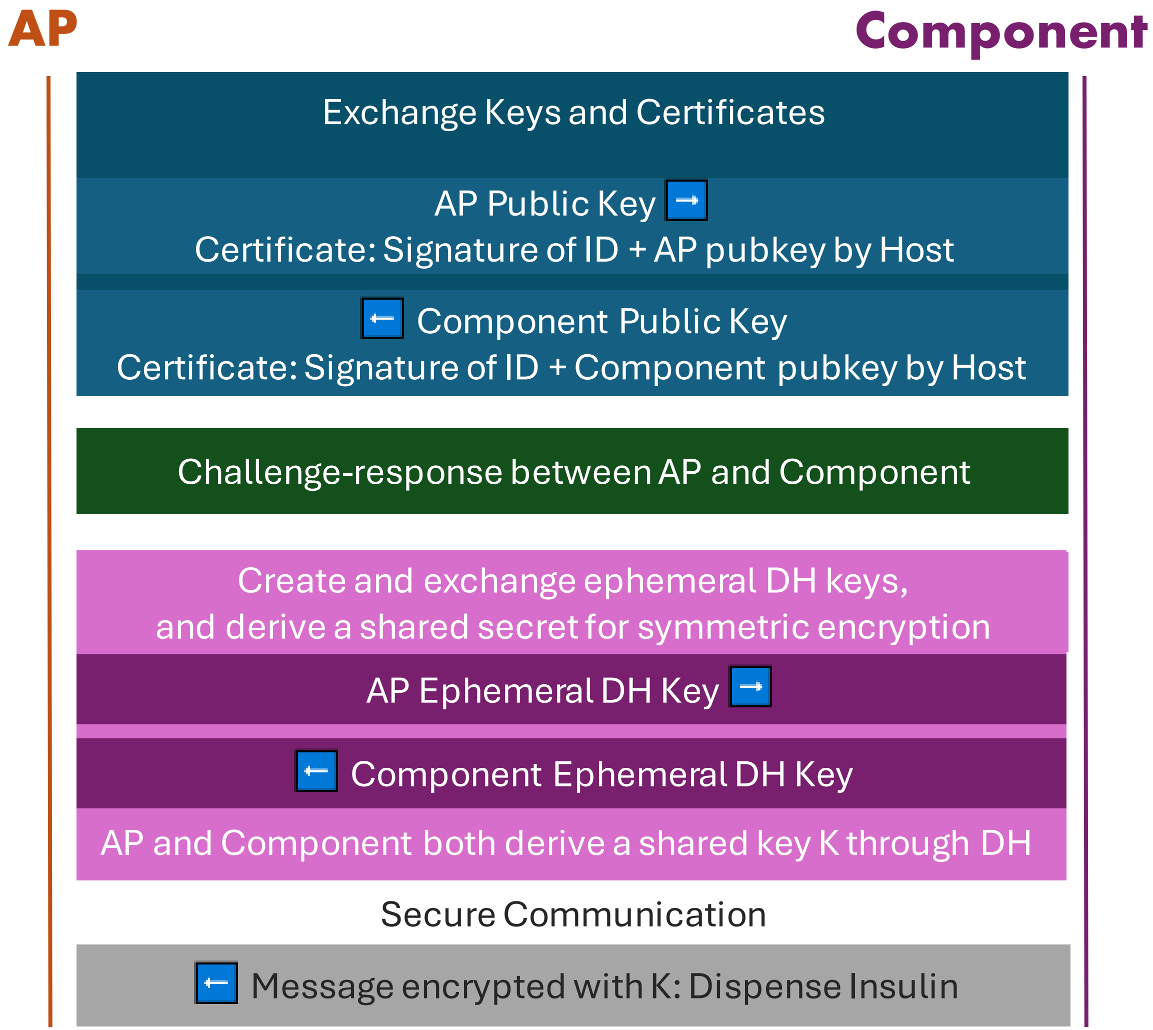
This offers perfect **forward-secrecy** of communications - which means that all communications are secure against a future compromise of either party's keys.



Idea 6

Use a **Diffie-Hellman Key Exchange** with ephemeral keys. In our design, both the AP and Component generate pairs of random **Curve25519** keys during the handshake. They then exchange the public points and combine them such that they both arrive at the same result, and no outside party observing this exchange can figure out the same key. This shared result is then used as the secret key for symmetric encryption, and the keys are then discarded after the session end (which is why they are called ephemeral).

This offers perfect **forward-secrecy** of communications - which means that all communications are secure against a future compromise of either party's keys.



An Unaddressed Flaw

We've ensured that messages between boards are encrypted and cannot be modified using authenticated encryption, but there is still a vulnerability that could lead to malfunction.

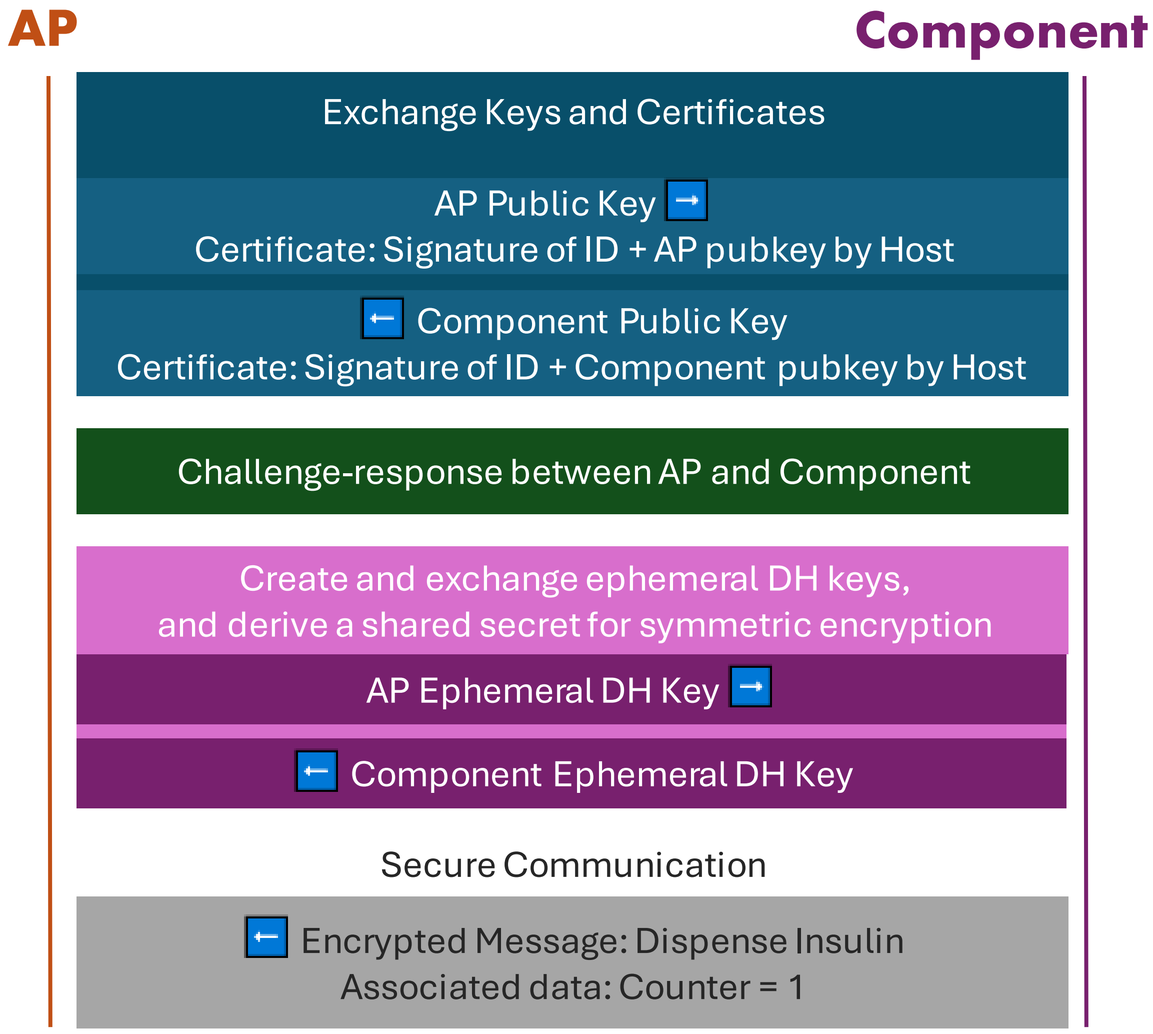
An attacker could mess with the functionality of an insulin pump component by receiving packets sent from an AP post-boot and resending it multiple times, causing it to dispense a dangerous amount of insulin. These packets would be valid and encrypted with the proper keys, but the component would have no way to know that they were only meant to be received once instead of multiple times!

This was an attack against a few teams that we unfortunately did not capture in time, as it required building a MITM board that could act as both an I2C controller and peripheral ☹️

Idea 7

To prevent this, we can encrypt a **counter** with each message and keep track of it on the receiving end. The counter increments for every message sent, and we make sure to reject any packets that are below the latest value of the counter.

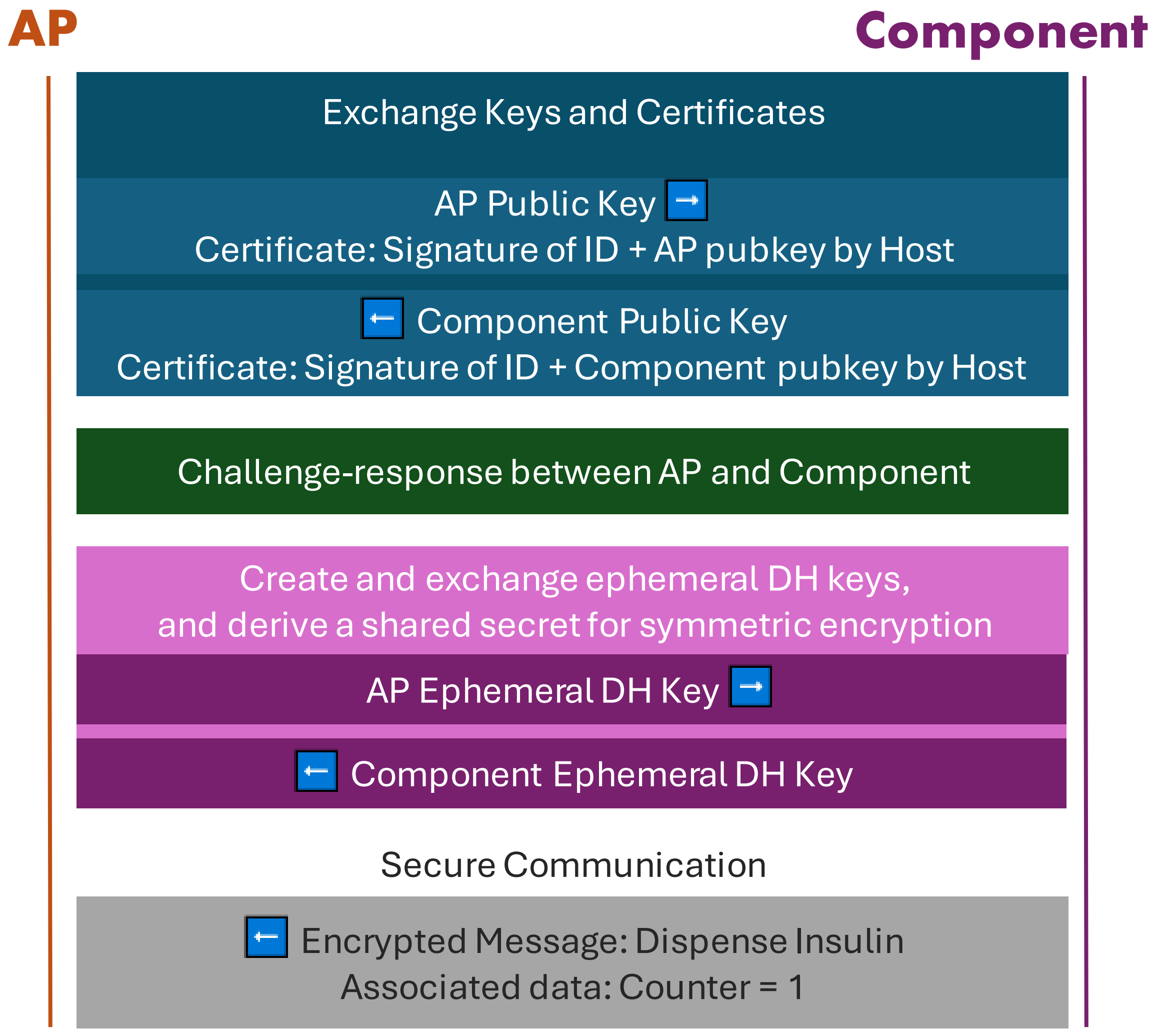
We can do this by adding **associated data** to the authenticated encryption (ChaCha20-Poly1305 in our case), using the counter value as additional data that is attached to every packet. If a device receives a packet that has an old or repeated value of counter, it refuses to accept it. This protects us against replay attacks.



Final design!

This is a simplified version of the cryptography in our final design.

In the implementation, some parts were condensed to make the handshake more efficient, such as using the ephemeral Diffie-Hellman keys as the unique nonces to be signed for the challenge-response, and sending them as part of the initial message including the public key and certificate.



Performance

MichState	75	88	1338	160	91	89	89

Leaves a lot to be desired but we successfully defended one flag!

Aiming to defend (and attack) more next year :)

the one singular unexploited flag hanging on for life

Thank You!

Sources

Diffie-Hellman Visualization

https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/Diffie-Hellman_Key_Exchange.svg/375px-Diffie-Hellman_Key_Exchange.svg.png

Today's Presentation Agenda

#eCTF2024

- 10:45 University of Illinois Urbana-Champaign
- 11:00 Delaware Area Career Center
- **11:15 BREAK**
- 11:25 A Word from NSTXL
- 11:35 Purdue University
- 11:50 Michigan State University
- 12:05 University of California, Irvine
- **12:20 LUNCH / NETWORK**
- 1:20 University at Buffalo
- 1:35 Carnegie Mellon University
- 1:50 A Word from Fortinet
- 2:05 Award Presentation
- 2:20 Closing Remarks / Student Dismissal

MITRE MITRE ENGENUITY

University of California, Irvine

#eCTF2024

Welcome
BugEaters

Currently 11th place with 3,265 points

The BugEaters

University of California, Irvine

Team Leader: Peter the Bugeater

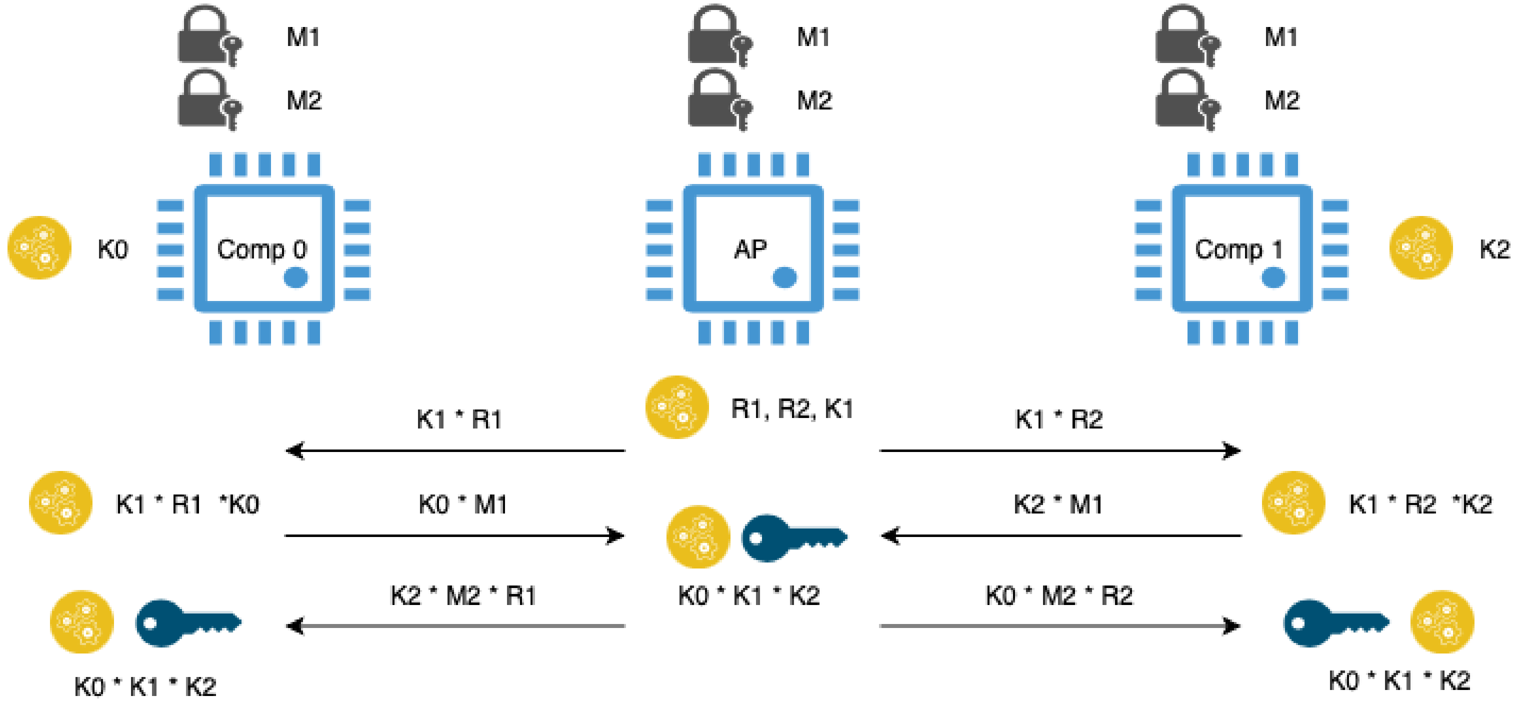
- *Presented by:* Jinyao Xu | Zuhair Taleb
  - Zhanhao Ruan | Yintong Luo
  - Richard Sima | Songhao Wang (Emeriti)
- *Advised by:* Professor Ian G. Harris

Outline

- **Towards a Secure Design**
  - Mask-on Key-Exchange-Verify
  - Random-Nonce-Based Communication
- **Towards a Robust Attack**
  - Weak Crypto & Weak Design
  - Brute Force Attack
- **2025-ECTF Directions**

Mask-On Key-Exchange-Verify

- A One-Time-Pad XOR-based Key Synthesis Protocol
  - Randomization of AES key for 3 devices: Prevent Board Switching.

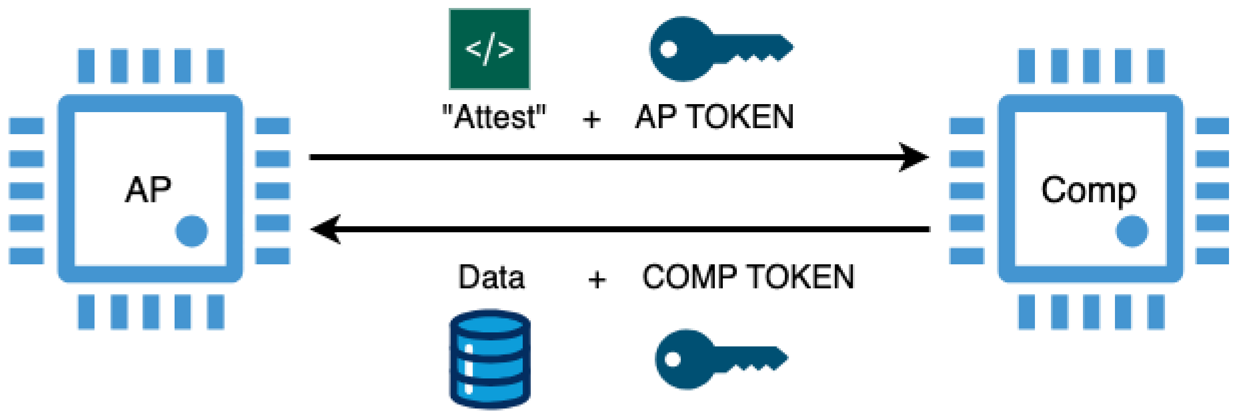


Nonce-Based Communication

- Communication protocols
  - Pre-boot: Two-way handshake
  - Post-boot: Three-way handshake
- Attacks we considered when developing our design
  - Brute force
  - Replay attacks
  - Man-in-the-middle (MitM)
- Attacks we didn’t consider
  - Side channel 😭

Weak Crypto & Weak Design

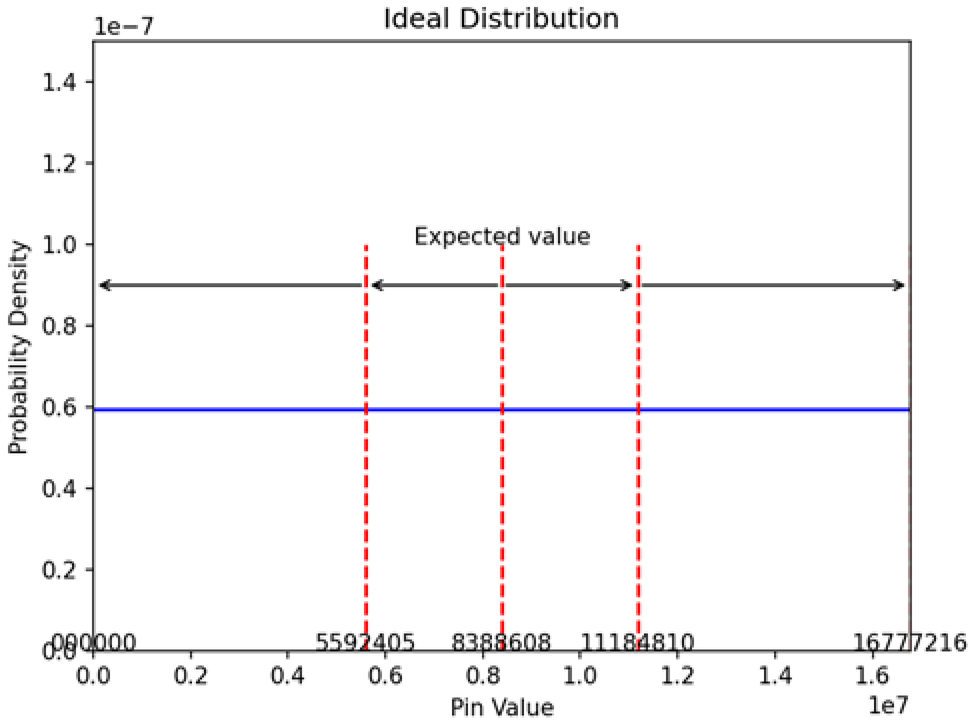
- Replay Attack to Get Firmware Token(Recognition Key)
  - The design validates the authenticity of the firmware by sending and validating tokens.
  - The tokens are static once built and are sent in plain text.
  - We built a malicious component with their codes and printed out the AP(Application Processor) token through the message sent from AP.
  - We used the acquired AP token to build a malicious AP and used it to trick the real Component to send the Component token back.



Brute Force Attack
— Simple but effective

- Target designs with no lock-out time delay for incorrect input

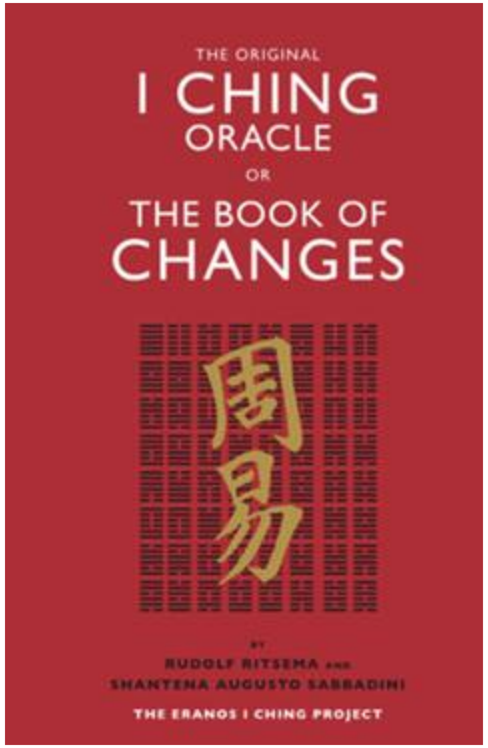
Strategy:



$$E[X] = \int_a^b x \; 1/(b-a) \; dx$$

$$= 1/(b-a) \int_a^b x \; dx$$

$$= (a+b)/2$$



Looking Ahead: 2025

- Secure designs against our attacks
  - Time delay (prevents brute force)
  - Nonced encryption: prevents replay attacks
- Would some of your attacks also be successful against your own system?
  - Our design was heavily designed with replay attacks in mind, as well as a secure key exchange algorithm and thus none of the attacks we conducted would've been successful
- With more time and resources, what other things would you have done?
  - We'd only realized that side channel attacks were much more feasible and effective than we expected once we entered the attack phase, so that's an aspect we'd like to learn to perform and defend against for next year. Additionally, we tried to but were unsuccessful in setting up a breadboard to perform a man-in-the-middle attack

Additional Acknowledgement

- **Additional Members**
- Xiaozheng Li
- Emma Xiao
- Yeseong Moon
- Zhengxuan Li

Criticism Or Questions?

LUNCH BREAK
12:20 PM– 1:20 PM

Restrooms, Refreshments

See you soon!

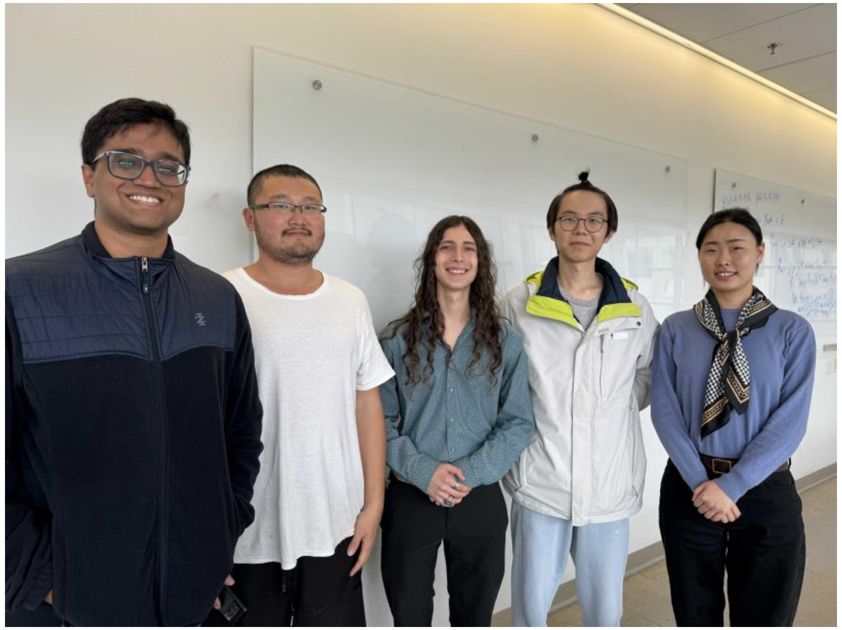
Today's Presentation Agenda

#eCTF2024

- 10:45 University of Illinois Urbana-Champaign
- 11:00 Delaware Area Career Center
- **11:15 BREAK**
- 11:25 A Word from NSTXL
- 11:35 Purdue University
- 11:50 Michigan State University
- 12:05 University of California, Irvine
- **12:20 LUNCH / NETWORK**
- 1:20 University at Buffalo
- 1:35 Carnegie Mellon University
- 1:50 A Word from Fortinet
- 2:05 Award Presentation
- 2:20 Closing Remarks / Student Dismissal

University at Buffalo

#eCTF2024



Welcome

BugEaters

Currently 4th place with 10,422 points

Team Cacti

University at Buffalo

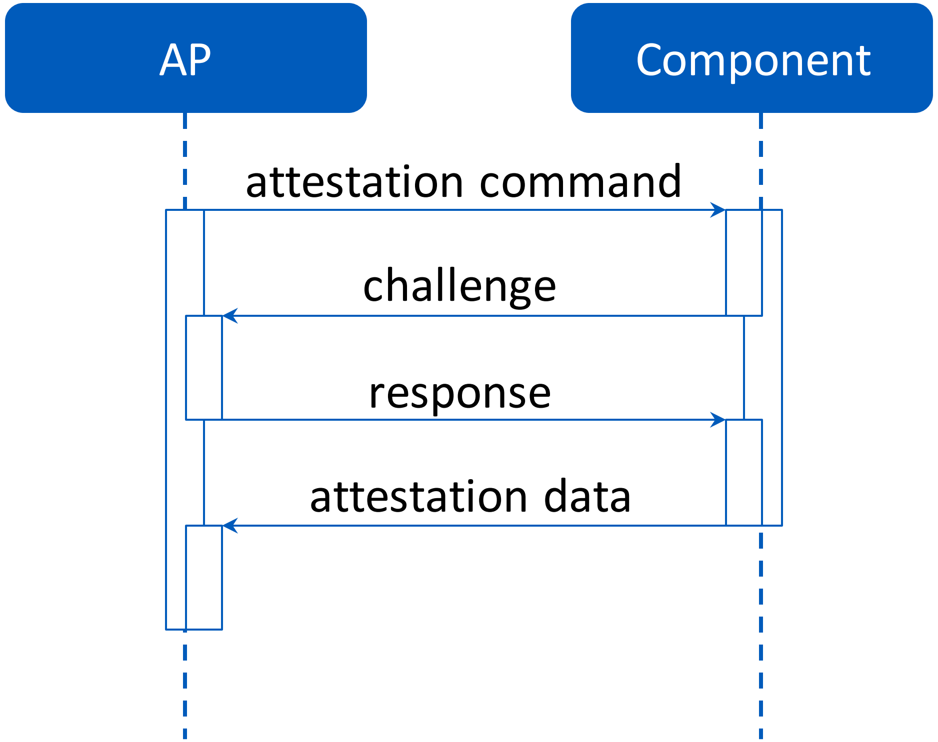
Gaoxiang Liu
Zheyuan Ma
Alex Eastman
Xi Tan
MD Armanuzzaman
Sagar Mohan
Afton Spiegel
Sai Bhargav Menta

Advised by: Prof. Ziming Zhao and Prof. Hongxin Hu

Design Overview

Design Overview

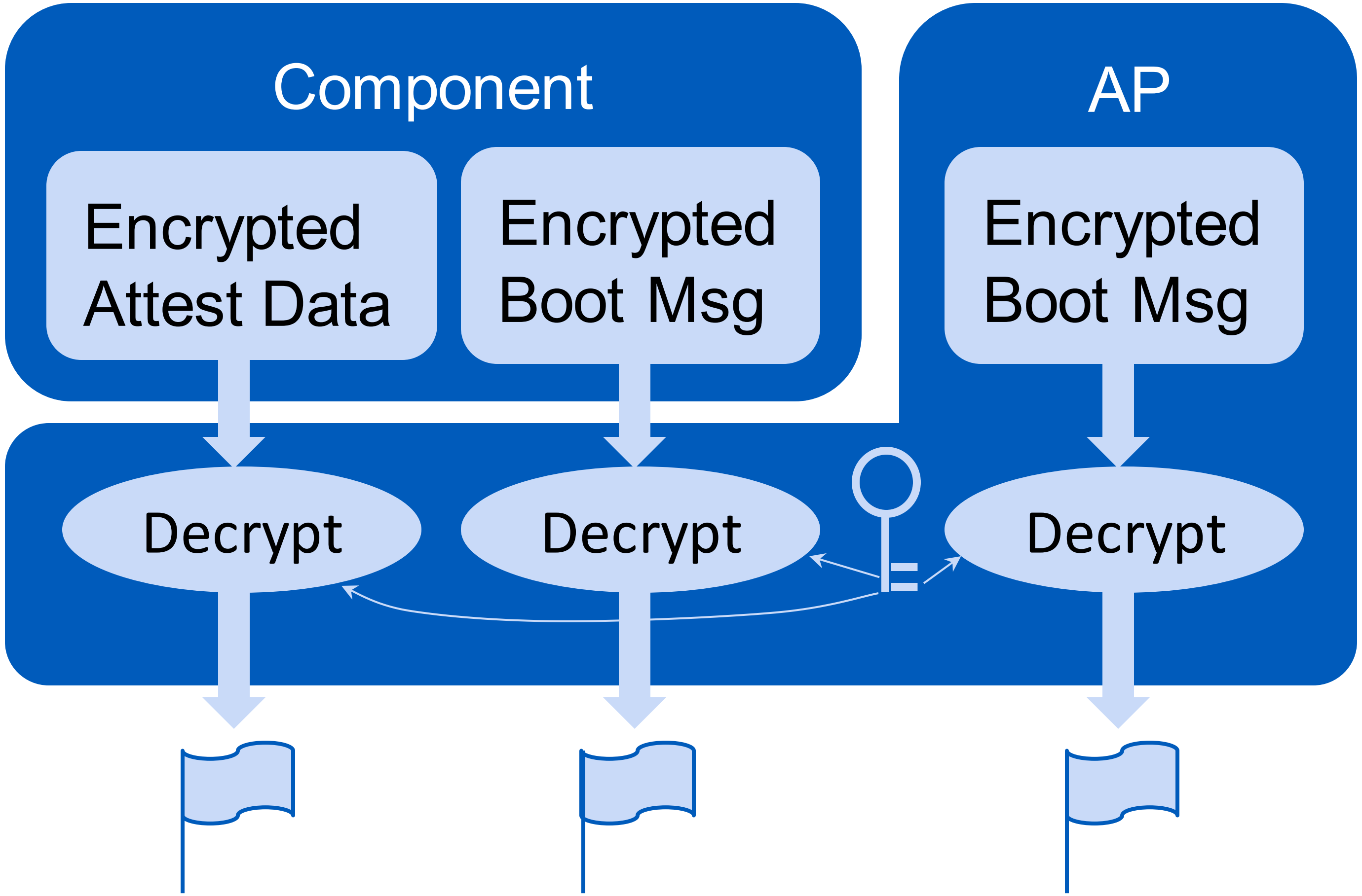
- Use the Monocypher library
- Use a challenge-response mechanism to authenticate
  - The challenge is a random number generated by the True Random Number Generator (TRNG)
  - It relies on a key shared between the AP and the component



Example: the component authenticates the AP during the attestation process

Defense Mechanisms

Sensitive Data (Flags) Encryption



Mitigating Brute-Force Attacks

- Use the Argon2 keyed-hash algorithm for the attestation PIN and replace token
    - Argon2 is for password hashing
    - Computing speed is deliberately slow
- Introduce delays in the PIN and token validation processes
- A longer delay is introduced after an unsuccessful attempt
    - The delay remains effective even after resetting the board

Mitigating Fault Injection Attacks

- Remove debugging messages and turn off the LED
  - They can be used as triggers for fault injection attacks
- Introduce random delays of several hundred CPU cycles
- Execute important conditional expressions twice
  - E.g., branching on PIN code validation

Additional Defenses

- Memory wiping
  - Zero out the memory area which contained sensitive data, such as keys, after each use
- Communication timeout
  - A timer is started after sending a message, and the response must be received before the timer expires
- Constant time comparator for the PIN code checking
  - Mitigates timing side-channel attacks

Attacks

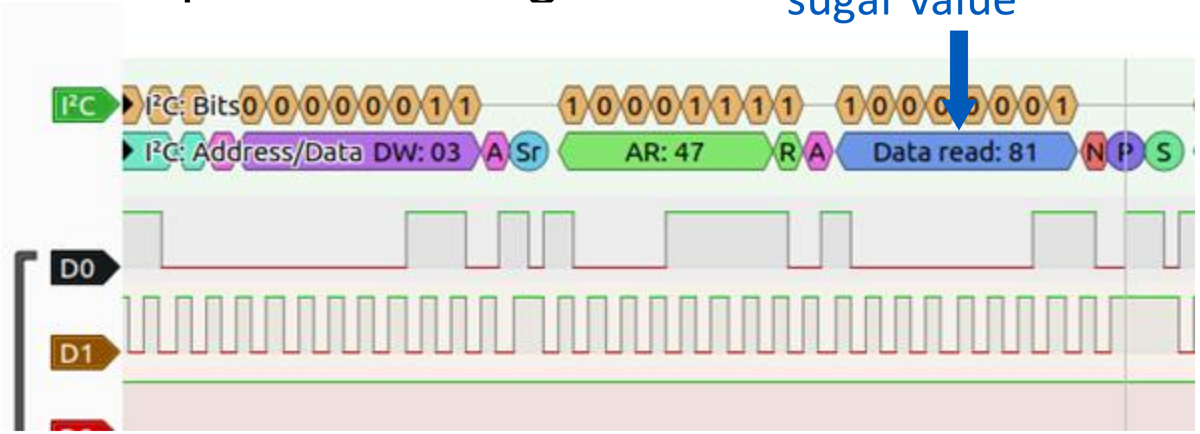
Brute-Force Attack

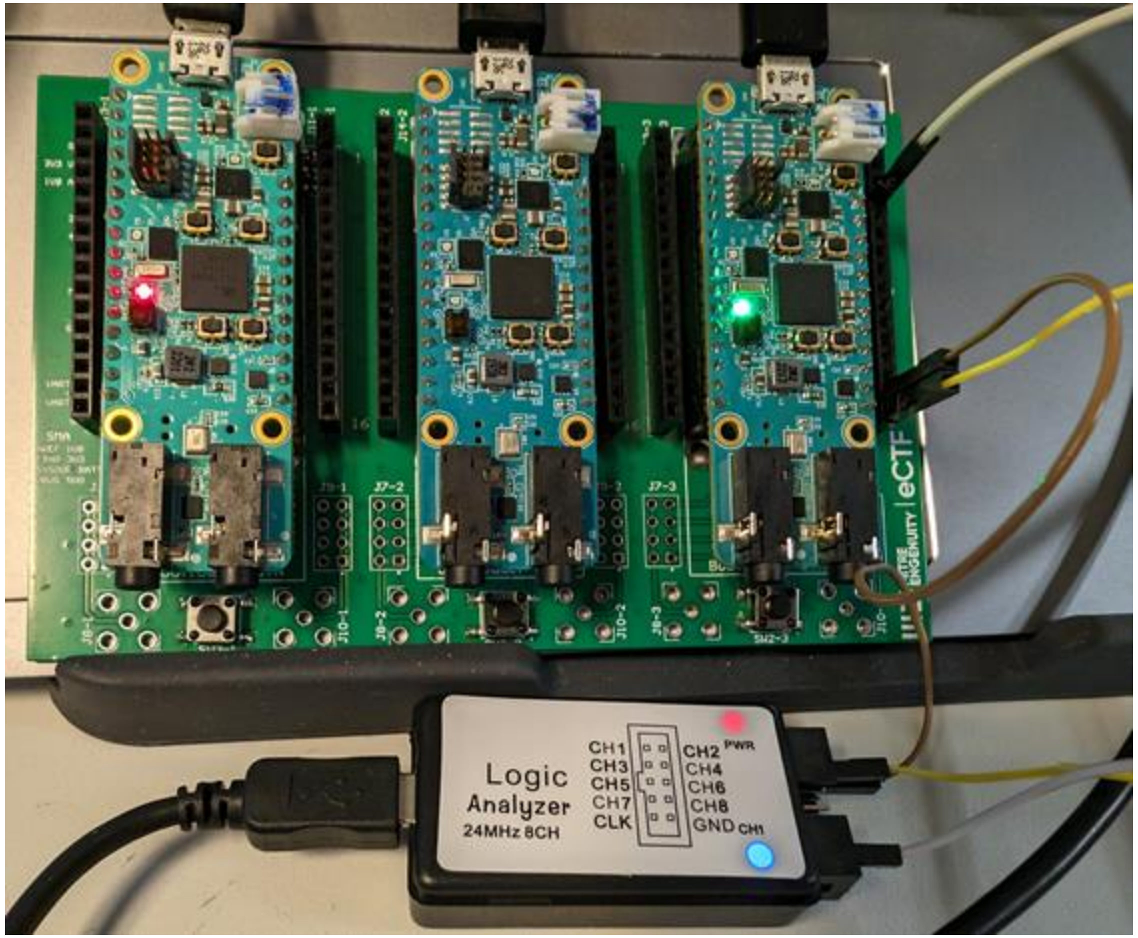
- Try all the possible PIN codes
- Use the Python UART library
- Utilize the three attack boards (3-threaded)
- Debug messages, such as “PIN Accepted!” tell when the correct PIN is found
- Finds the correct PIN within 15 hours for designs without delays

Replay Attack

- Use a logic analyzer to capture traffic on the I2C bus
- Replay specific captured messages
- The attack works if there is no message integrity check or if the checksum remains constant for a specific message

A valid high blood sugar value





Exploiting Other Design Flaws

- Same/no secrets for all deployments
  - Self-built firmware will be valid for any attack scenario
- Predictable keys
  - Global variables are 0
  - The keys, intended to be random by design, are not actually random.
- Weak/no validation
  - Sending a fixed value as the authentication token
    - The value can be captured and then replayed

Thoughts and Tips

Thoughts and Tips

- Don’t rush to submit
  - We had a buffer overflow bug last year
  - The defense points helped a lot this year
- Always encrypt sensitive data
- Use Elliptic Curve Cryptography (ECC) instead of RSA for asymmetric encryption
  - RSA will slow down the system
- Check the disassembly code from your firmware to make sure it works as expected
  - Use Ghidra or objdump
- Use multiple entropy sources for generating random numbers
- Utilize the hardware resources
  - E.g., the temperature sensor on the board last year and the TRNG on the board this year

Thank you!

Q & A

Today's Presentation Agenda

#eCTF2024

- 10:45 University of Illinois Urbana-Champaign
- 11:00 Delaware Area Career Center
- **11:15 BREAK**
- 11:25 A Word from NSTXL
- 11:35 Purdue University
- 11:50 Michigan State University
- 12:05 University of California, Irvine
- **12:20 LUNCH / NETWORK**
- 1:20 University at Buffalo
- 1:35 Carnegie Mellon University
- 1:50 A Word from Fortinet
- 2:05 Award Presentation
- 2:20 Closing Remarks / Student Dismissal

Carnegie Mellon University

#eCTF2024

Welcome

Plaid Parliament of Pwning

Currently 1st place with 39,052 points

Plaid Parliament of Pwning 2024 eCTF Team
Carnegie Mellon University

Akash Arun, Andrew Chong, Aditya Desai, Nandan Desai, Quinn Henry, Sirui (Ray) Huang, Tongzhou (Thomas) Liao, David Rudo, John Samuels, Anish Singhani (lead), Carson Swoveland, Rohan Viswanathan, and Gabriel Zaragoza

Advised by Anthony Rowe, Patrick Tague, and Maverick Woo

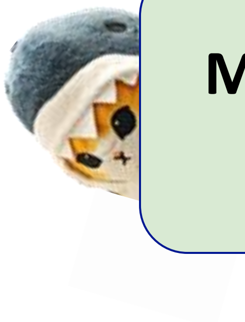
Carnegie Mellon University Computer Science Department
CyLab Carnegie Mellon University Security and Privacy Institute
Carnegie Mellon University Electrical & Computer Engineering
ini Carnegie Mellon University Information Networking Institute

Presentation Outline

- **Design Phase**
  - Overview
  - Design Highlight: Zero-Trust Architecture
- **Attack Phase**
  - I2CBleed Exploit
  - Supply Chain I2CBleed
  - Other Attacks + Interesting Defenses
- **Project Management + Lessons Learned**

Our Design Highlights

Encryption At-Rest of *Everything*	Custom Hardened Physical Link Layer	Encrypted Link Layer Wrapper
Random Nonces to Prevent Replays	ChaCha-Poly AEAD for encryption	Board RNG + von-Neumann
Minimal External Code Surface	Avoid Interrupts & Async Code	Random Delays + Redundant Checks



Design Highlight: Zero-Trust Architecture

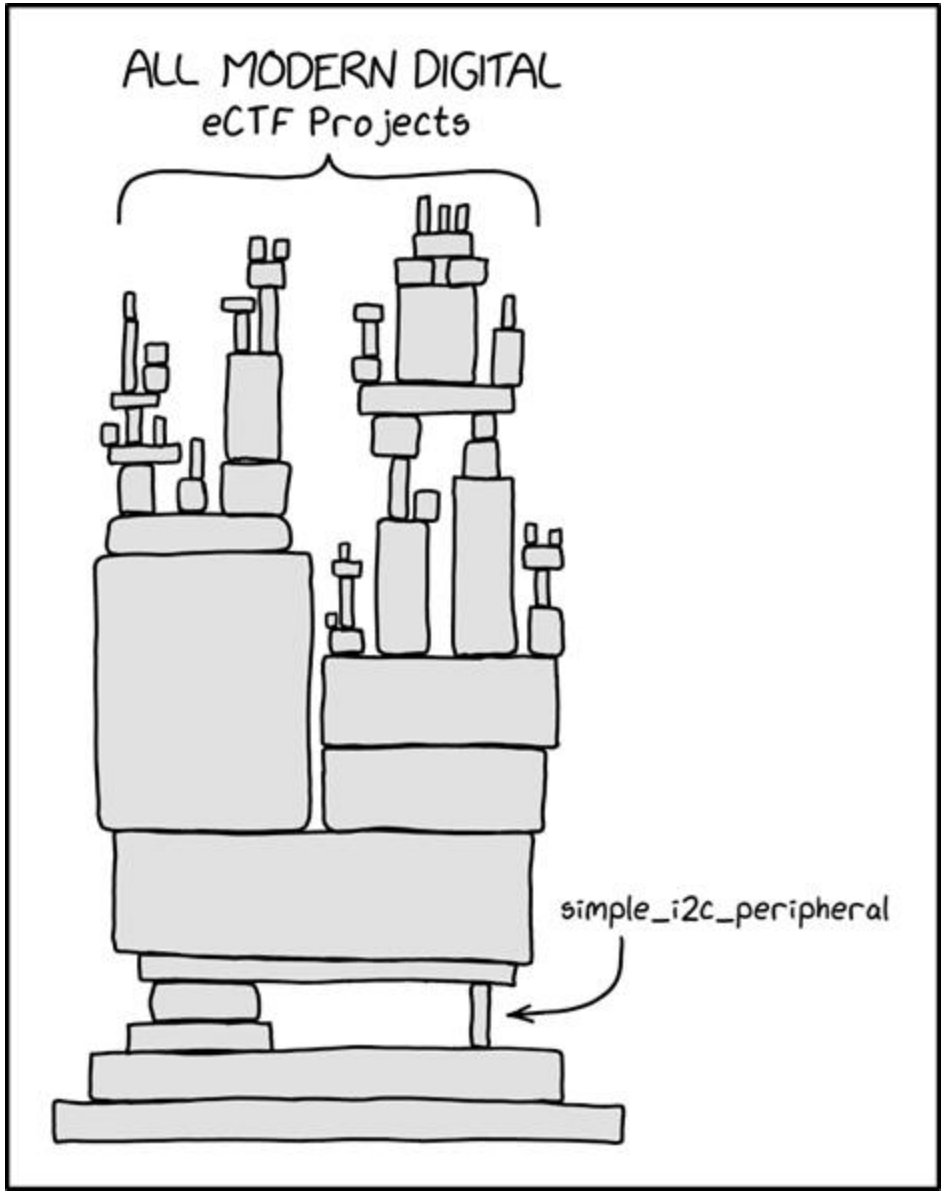
- **Thought Experiment:** Assume full hardware compromise
  - How to defend flags? Can we use fun crypto tricks?
- *BB Boot / BB Extract*: Encrypt comp. secrets w/ key stored in AP
- *Op. PIN Extract / SC Extract*: Encrypt keys inside AP w/ PIN
  - *Potential for offline brute-force if AP compromised
- *Op. Pump Swap:* Not defensible, but encrypt the code to make it harder
- *SC Boot / Damaged Boot:* ?????
  - How to require both components to be present in order to boot?

Design Highlight: Zero-Trust Architecture

- **Damaged Boot: Require all components be present in order to boot?**
- “Russian Encryption Doll”: Encrypt AP boot data with all component keys
- How to distribute component keys?
  - Comp Key = Hash(Root Key || Comp ID)
- How to do **replace component?**
  - Keep Root Key encrypted with Replace Token
  - RT is long enough to not be brute-able

Attack Highlight: I2CBleed

- **Three vulnerabilities in starter code**
  - Read/write indices not reset on repeated start
  - Read index checked for == instead of >=
  - Write index casted to unsigned (overflows)
- **Result: Arbitrary Read/Write (!!!)**
  (of anything past `I2C_REGS`)
- **Straightforward Attack Process**
  1. Write in malicious shellcode
  2. Write a bunch of padding
  3. Overwrite vector table to jump to shellcode



Attack Highlight: Fully-Automated I2CBleed

- **Q: What to do with a near-universal arbitrary-code-execution exploit?**
  - A: Make it *full auto: <u>4-5 flags in 90 seconds from ZIP download</u>*
- **Step 1: Determine I2C Address of Victim**
  - Scan all addresses, see which ones ACK (like insecure `list_components`)
- **Step 2: Determine `I2C_REGS` address (shellcode address)**
  - Use arbitrary read until the component crashes (stops ACKing)
- **Step 3: Inject shellcode**
  - Step 3.5 (SC only): Scan until we find the string "`ctf{`"
  - Locally: Dump all of flash to the UART (including keys and plaintext flags!!)
- **Step 4 (SC only): Bitbang SPI data back to malicious component**
  - Malicious component receives SPI and dumps anything transmitted over UART

Interesting Defenses

- **Defending against I2CBleed**
  - Certificate Chain: Provide each component with a ID-unique certificate signed using a deployment-time CA
  - Encrypt component attestation data / boot message with key stored in AP
  - Key pinning to assign unique component keys (bypass deployment hash check...)
- **Other unique defenses**
  - Challenge-response handshake on every message in the system
  - Custom I2C implementation (don't trust provided libraries...)
  - Use of hardware features / PUFs to prevent emulation

Project Management + Lessons Learned

- **Design Phase**
  - Get everyone set up with insecure example in the first week
  - Design security protocol *before* starting implementation, but can start generic tasks (scripting, infra, comms, crypto library) simultaneously
  - Secure By Design: Drive out the attacker in every possible way

- **Attack Phase**
  - Balance between optimizing conventional attacks and developing novel attacks
  - Track red-team availability for executing rapid attacks for first bloods
  - Be willing to operate at strange hours (sadly)

Project Management + Lessons Learned

- **Overall**
  - Earning course credit helps offset the time investment
  - Cross-Training: EEs studied crypto, Security students studied electronics
  - If viable, hardware setup for each team member to individually play with

- **Lessons Learned**
  - Sustainability of having most of the work be done by a few team members?
  - Redundancy to avoid single points of failure (esp. for design phase timeline)
  - Novel attacks require *a lot* more human-hours than estimated, fine-tuning "standard" attacks can be better

Sponsors Acknowledgement

We acknowledge the generous support of the following sponsors to our team:

CyLab IoT Initiative
AT&T
AWS
Cisco
Infineon
Nokia Bell Labs
Rolls-Royce
Siemens

(Any opinions, findings, and conclusions or recommendations expressed in this material are those of our team and do not necessarily reflect the views of our sponsors.)

Carnegie Mellon University Computer Science Department
CyLab Carnegie Mellon University Security and Privacy Institute
Carnegie Mellon University Electrical & Computer Engineering
ini Carnegie Mellon University Information Networking Institute

Thank you!

#eCTF2024

Welcome

DOUG SANTOS

Director, Advanced Threat Intelligence

FORTINET

MITRE ENGENUITY

Douglas Santos : Who, What, Why

Douglas Santos

Director, Advanced Threat Intelligence

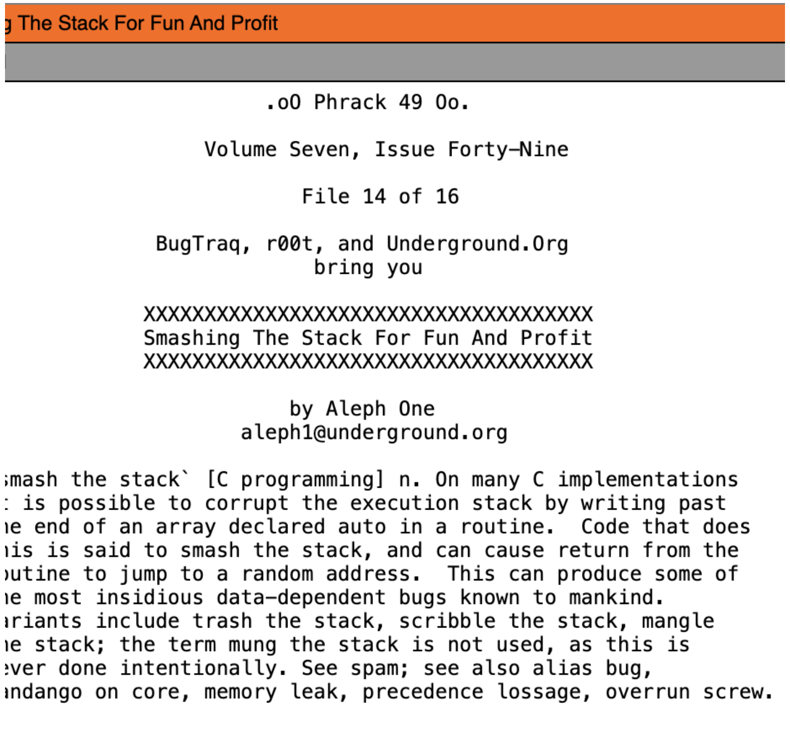
Who IS this guy ?

My first computer

Hackers Movie

First contact with ‘The Web’

Who IS this guy ?



g The Stack For Fun And Profit

.oO Phrack 49 Oo.

Volume Seven, Issue Forty-Nine

File 14 of 16

BugTraq, r00t, and Underground.Org
bring you

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Smashing The Stack For Fun And Profit
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

by Aleph One
aleph1@underground.org

smash the stack` [C programming] n. On many C implementations
: is possible to corrupt the execution stack by writing past
ie end of an array declared auto in a routine. Code that does
iis is said to smash the stack, and can cause return from the
butine to jump to a random address. This can produce some of
ie most insidious data-dependent bugs known to mankind.
iriants include trash the stack, scribble the stack, mangle
ie stack; the term mung the stack is not used, as this is
ever done intentionally. See spam; see also alias bug,
indango on core, memory leak, precedence lossage, overrun screw.

the paper that changed it all

#phrack @ freenode

Internet in the late 90's was like

What ARE you doing ?

$ whatis dsantos

Threat Intelligence

CTID Research Sponsor

Tell me again WHY are you doing this ?

$ man dsantos

FORTINET®

2024 eCTF Final Results

Special Awards

$2,500

Special Award
Top High School

Awarded to the high school team with the most points at the end of the competition

Special Award: Top High School

#eCTF2024

Delaware Area Career Center

0xDACC

Samuel Goodman, Ezequiel Flores, Tyler McColeman, Cameron Crossley, Beau Schwab, Seth Tydings, David Nunley, Grayson Seger, Andrew Langan, David Nunley, Tyler McColeman, Jake White, Henry Reid, Ethan Martindale

Advised by: Eli Cochran

MITRE MITRE ENGENUITY

Special Award
Best Poster

Awarded to the team that was given the highest-scoring poster as judged by a panel of experts

Special Award: Best Poster

#eCTF2024

Purdue University

Team b01lers

Nicholas Andry, Han Dai, Philip Frey, Jorge Hernandez, Ji Hun Hwang, Siddharth Muralee, Jaxson Pahukula, Mihir Patil, Adrian Persaud, Vinh Pham Ngoc Thanh, Jack Roscoe, Gabriel Samide, Lucas Tan, Vivan Tiwari, Neil Van Eikema Hommes, Jacob White, Tianze Wu, Kevin Yu

Advised by: Christina Garman, Mohammadkazem Taram, Santiago Torres-Arias

MITRE MITRE ENGENUITY

#eCTF2024

Special Award
Responsible Disclosure

Awarded to a team that identified a potential vulnerability that could have undermined the security of the competition and went through the responsible disclosure process to report the finding

Special Award: Responsible Disclosure

#eCTF2024

CARNEGIE MELLON UNIVERSITY

Team Plaid Parliament of Pwing

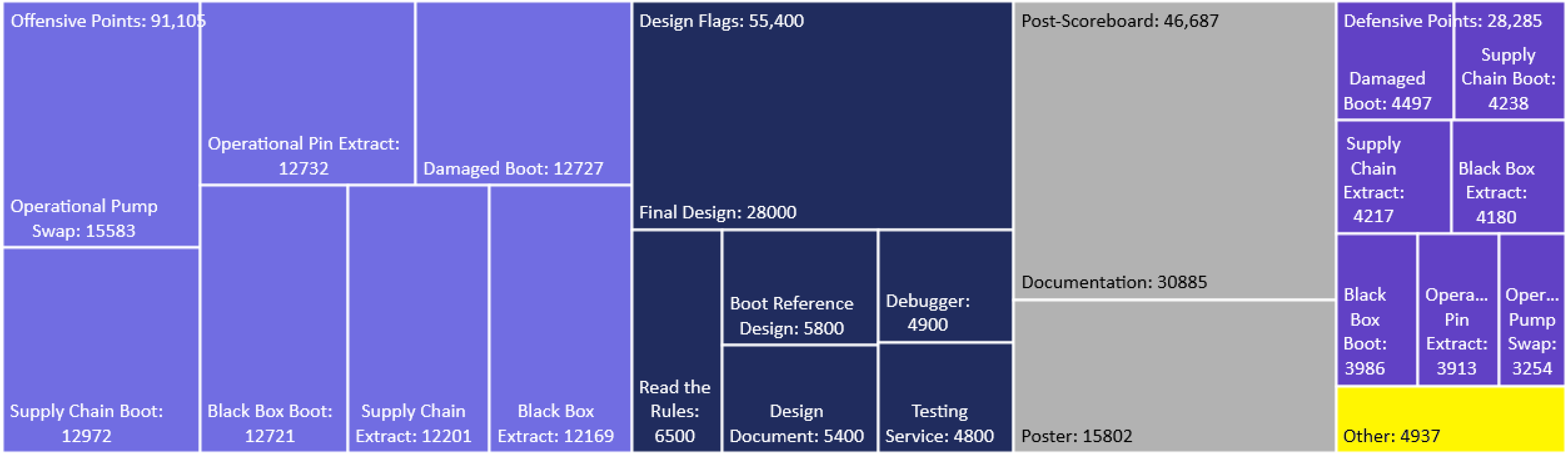
Akash Arun, Andrew Chong, Nandankumar Desai, Aditya Desai, Quinn Henry, Sirui Huang, Tongzhou Liao, David Rudo, John Samuels, Anish Singhani, Carson Swoveland, Rohan Viswanathan, Gabriel Zaragoza

Advised by: Anthony Rowe, Patrick Tague, Maverick Woo

MITRE MITRE ENGENUITY

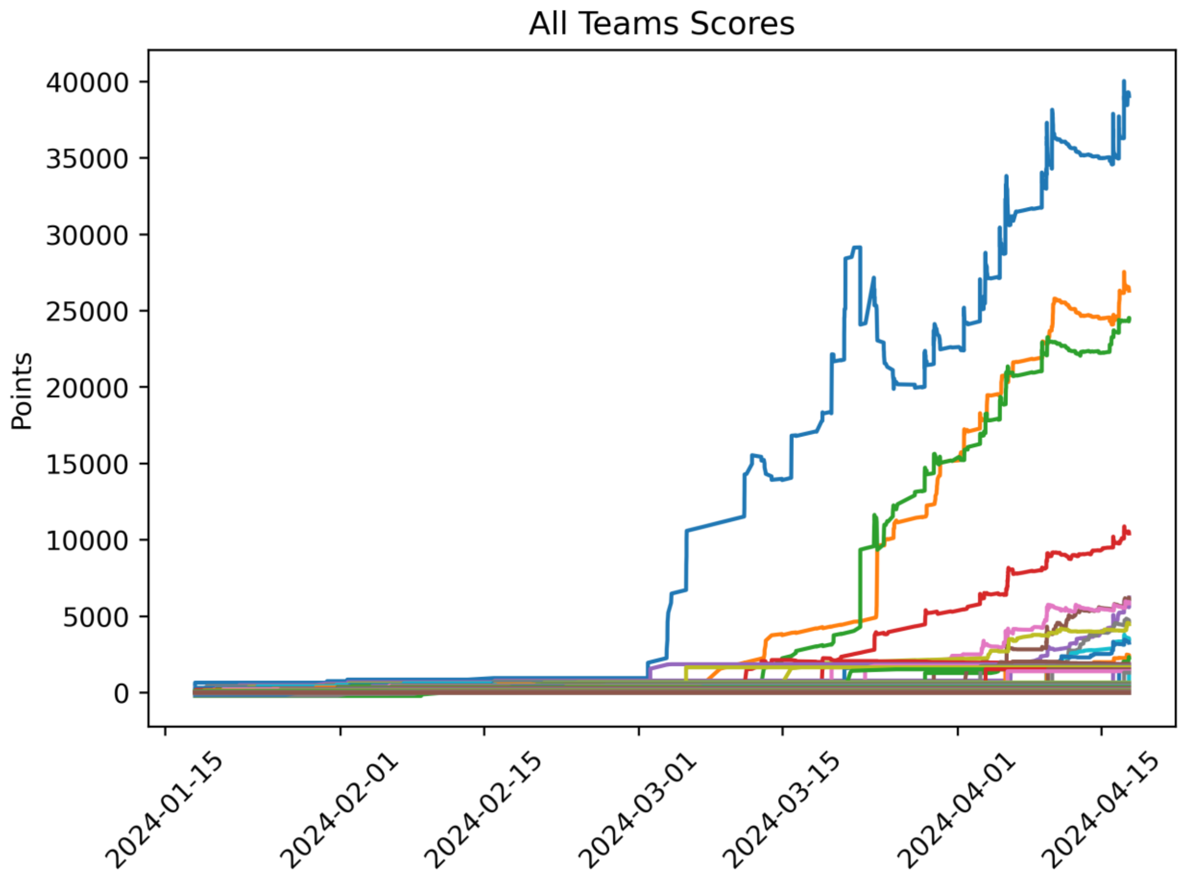
Final Scoring Breakdown

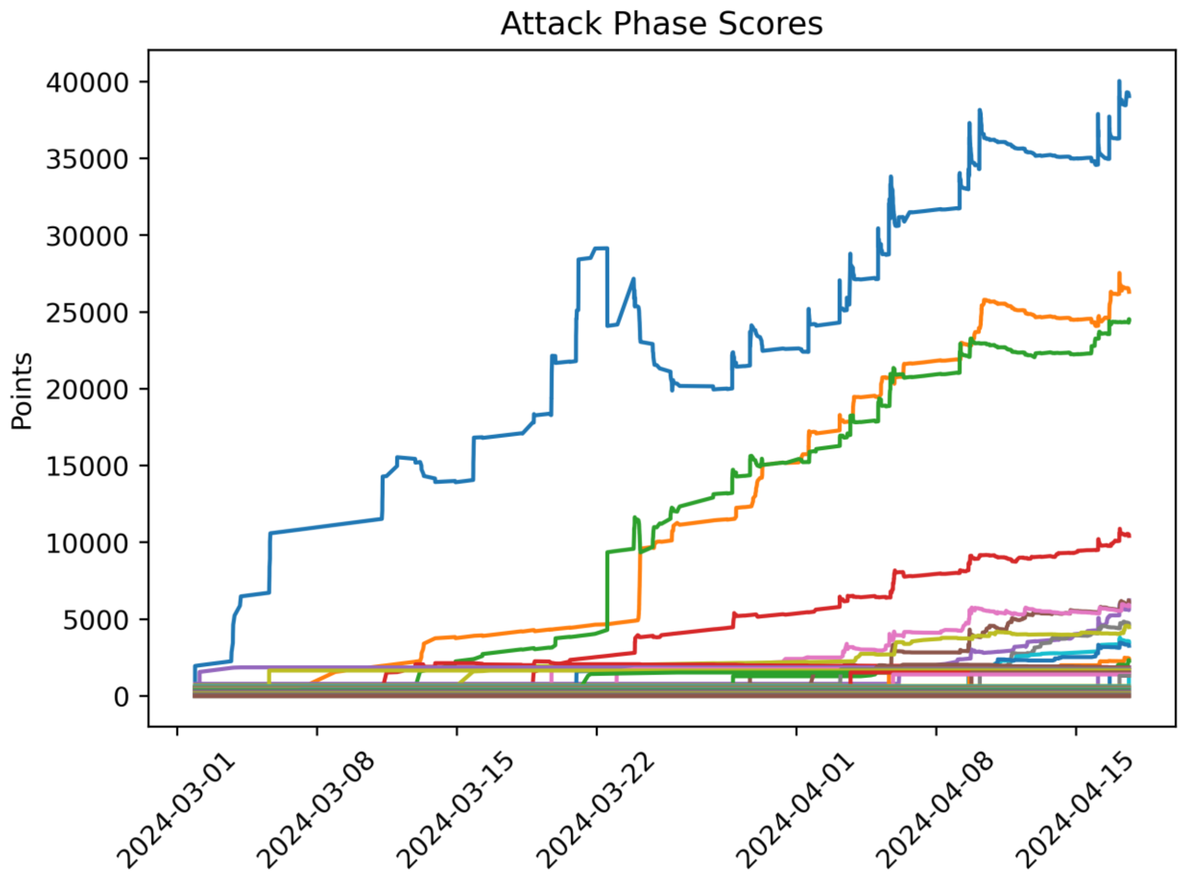
- Final scores are a combination of:
  - Design Phase flags
  - Defensive points
  - Offensive points
  - Documentation points
  - Poster points
- Documentation points and poster points are not shown on the scoreboard



Preliminary Scoreboard Results

#eCTF2024

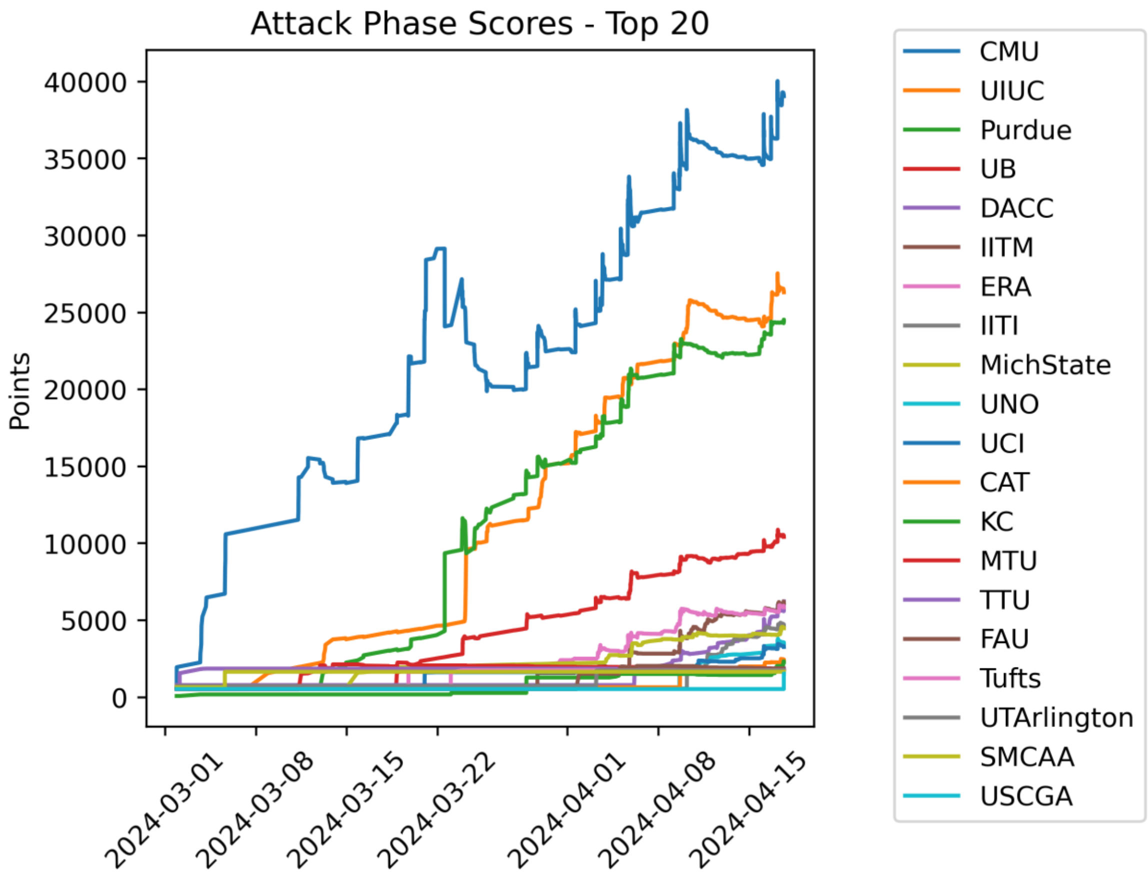


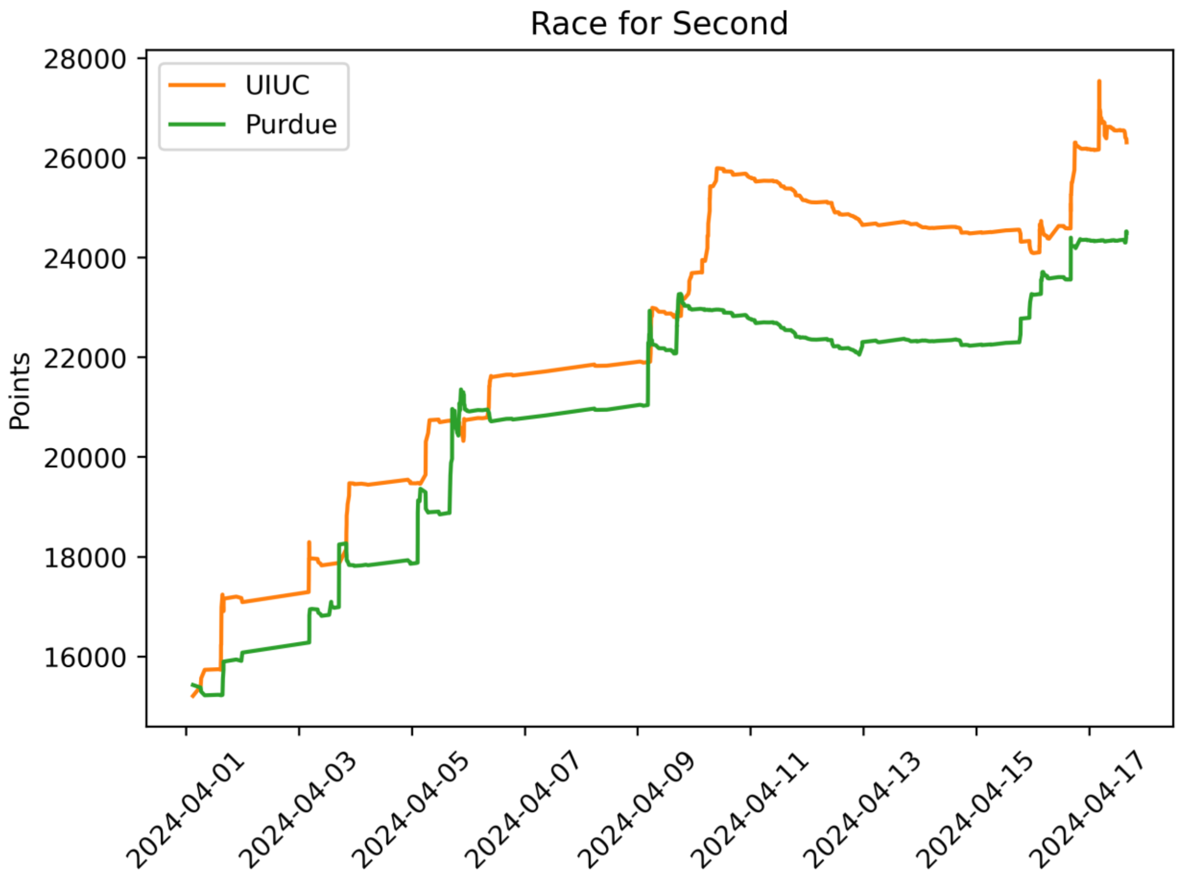


MITRE MITRE ENGENUITY

Attack Phase Scores Over Time

#eCTF2024

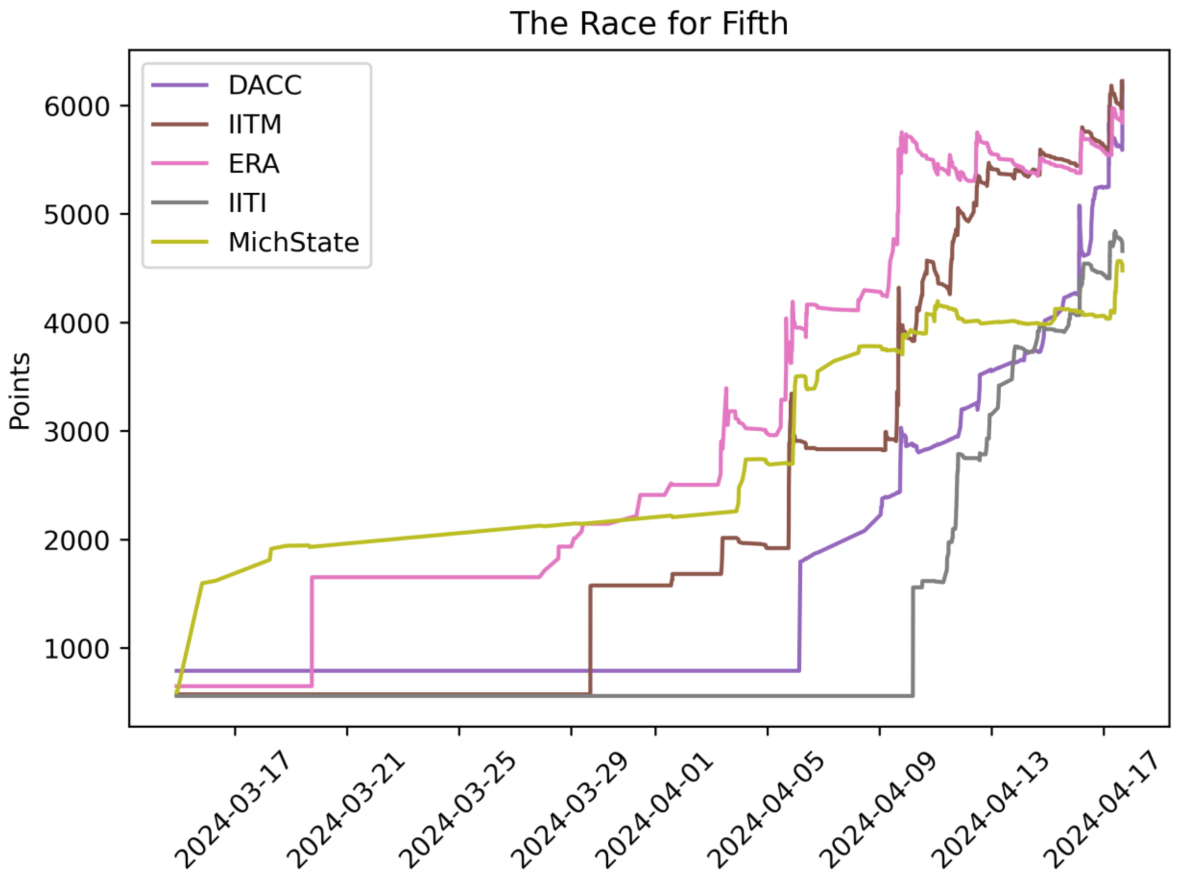


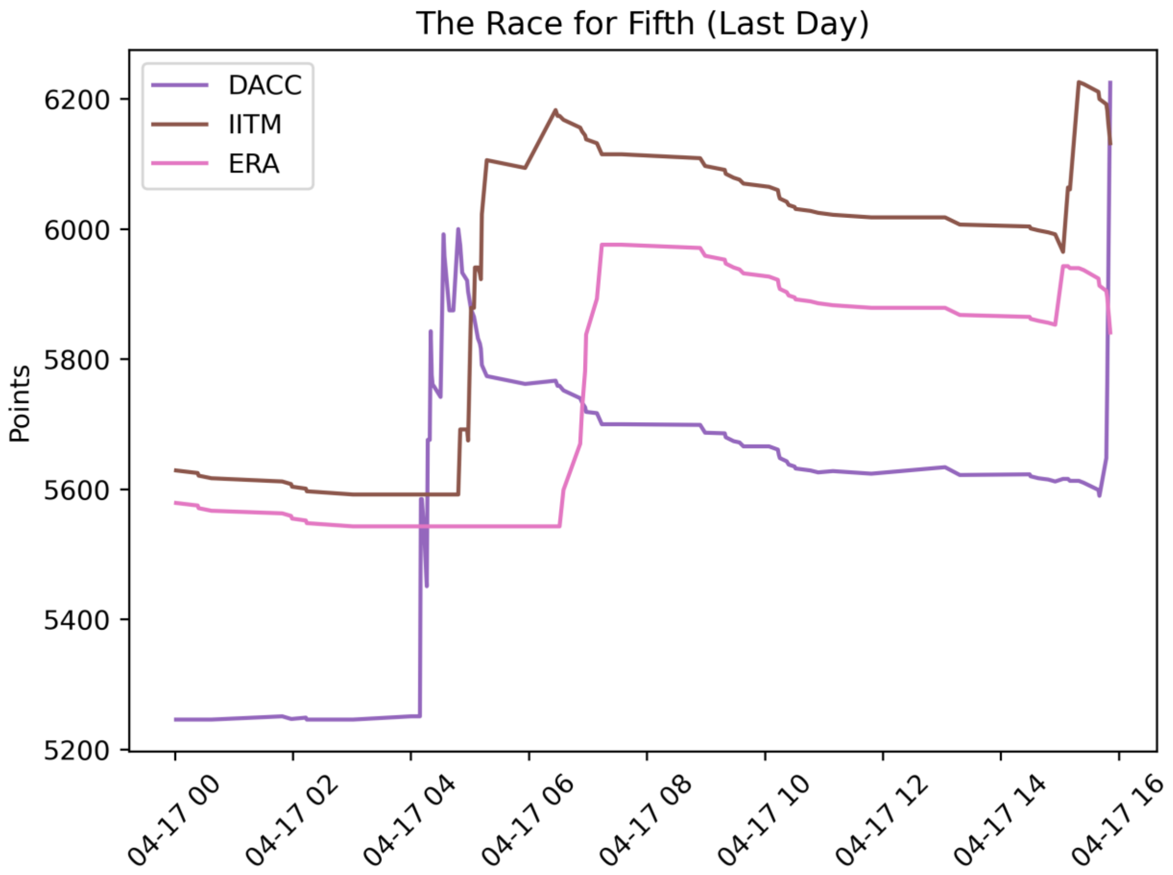


MITRE | MITRE ENGENUITY

Race for Fifth

#eCTF2024





MITRE | MITRE ENGENUITY

Third Place
$2,500

Third Place

#eCTF2024

26,926 Final Points
166 Flags Captured

PURDUE UNIVERSITY

Team b01lers

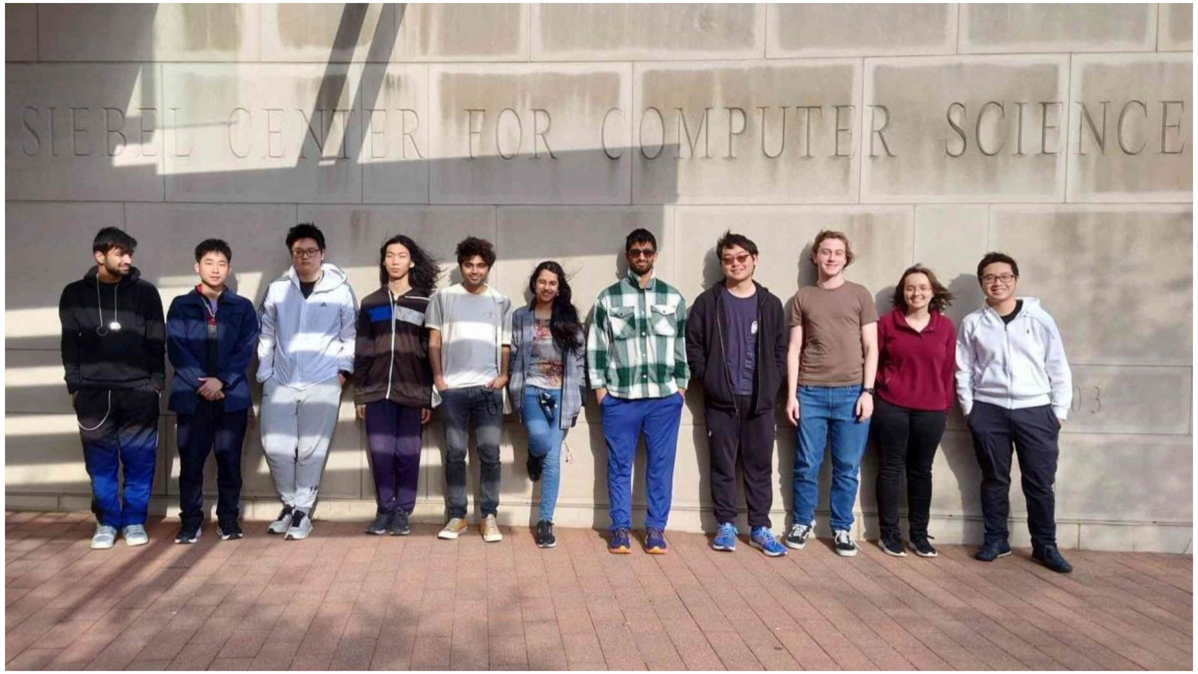
Nicholas Andry, Han Dai, Philip Frey, Jorge Hernandez, Ji Hun Hwang, Siddha rth Muralee, Jaxson Pahukula, Mihir Patil, Adrian Persaud, Vinh Pham Ngoc T hanh, Jack Roscoe, Gabriel Samide, Lucas Tan, Vivan Tiwari, Neil Van Eikema Hommes, Jacob White, Tianze Wu, Kevin Yu

Advised by: Christina Garman, Mohammadkazem Taram, Santiago Torres-Arias

Second Place
$5,000

Second Place

#eCTF2024



28,660 Final Points
173 Flags Captured

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Team SIGPwny

Team Leads: Minh Duong, Jake Mayer, Emma Hartman, Hassam Uddin
Team Members: Juniper Peng, Timothy Fong, Krish Asher, Adarsh Krishnan, Liam Ramsey, Yash Gupta, Suchit Bapatla, Akhil Bharanidhar, Zhaofeng Cao, Ishaan Chamoli, Tianhao Chen, Kyle Chung, Vasunandan Dar, Jiming Ding, Sanay Doshi, Shivaditya Gohil, Seth Gore, Zexi Huang, George Huebner, Haruto Iguchi, Parithimaal Karmehan, Jasmehar Kochhar, Arjun Kulkarni, Julia Li, Jingdi Liu, Richard Liu, Theodore Ng, Stefan Ninic, Henry Qiu, Neil Rayu, Ram Reddy, Sam Ruggerio, Naavya Shetty, Arpan Swaroop, Raghav Tirumale, Yaoyu Wu

Advised by: Professor Kirill Levchenko, PhD

MITRE | MITRE ENGENUITY

First Place
$10,000

First Place

#eCTF2024

41,581 Final Points
175 Flags Captured

CARNEGIE MELLON UNIVERSITY

Team Plaid Parliament of Pwing

Akash Arun, Andrew Chong, Nandankumar Desai, Aditya Desai, Quinn Henry, Sirui Huang, Tongzhou Liao, David Rudo, John Samuels, Anish Singhani, Carson Swoveland, Rohan Viswanathan, Gabriel Zaragoza

Advised by: Anthony Rowe, Patrick Tague, Maverick Woo

Final Scores: Top Teams

#eCTF2024

Rank	Change	Team	Scoreboard Score	Final Score
1	0	Carnegie Mellon University	39052	41581
2	0	University of Illinois Urbana-Champaign	26318	28660
3	0	Purdue University	24501	26926
4	0	University at Buffalo	10422	12558
5	+1	Indian Institute of Technology Madras	6135	8275
6	-1	Delaware Area Career Center	6225	8216
7	0	Ecole Royale de l'Air	5844	7447
8	0	Indian Institute of Technology Indore	4660	6614
9	0	Michigan State University	4483	6393
10	0	University of Nebraska Omaha	3506	5644
11	0	University of California, Irvine	3265	4964
12	+4	Florida Atlantic University	1850	3974
13	+2	Tennessee Tech University	1863	3843
14	+3	Tufts University	1757	3803
15	+6	University of Connecticut	1530	3698
16	-4	Colombe Academy of Technology	2417	3540

Rank	Change	Team	Scoreboard Score	Final Score
17	+7	East Tennessee State University	1527	3376
18	-5	Kilgore College	2290	3368
19	+1	United States Coast Guard Academy	1533	3316
20	+6	Virginia Tech	1519	3268
21	-7	Michigan Technological University	1867	3088
22	+1	Massachusetts Institute of Technology	1528	3020
23	-4	Shawnee Mission Center for Academic Achievement	1656	2572
24	-2	United States Air Force Academy	1529	2445
25	-7	University of Texas at Arlington	1751	2412
26	-1	University of Colorado, Colorado Springs	1521	2386
27	1	Oklahoma Christian University	1313	2195
28	-1	CyberAegis	1401	2012
29	+12	ISD 196	509	1198
30	+10	Pace University	512	1192
31	+6	University of New Haven	522	1044

MITRE MITRE ENGENUITY

#eCTF2024

CLOSING REMARKS

Dan Walters

Senior Principal Microelectronics Solution Lead

MITRE

MITRE ENGENUITY

Stay Connected

#eCTF2024

SCAVENGER HUNT!

eCTF Alumni Page

Winning Teams

Please stay in your seats after the ceremony for photos!

Enjoy the museum!

Thank You!

2024 MITRE eCTF Award Ceremony

Need help? Seek individuals with purple lanyards for help!

MITRE
SOLVING PROBLEMS FOR A SAFER WORLD

MITRE ENGENUITY
A Foundation for Public Good