2024 Collegiate eCTF Award Ceremony
WELCOME!
WE WILL GET STARTED SHORTLY
WELCOME
2024 ECTF AWARD CEREMONY
Ben Janis
Senior Embedded Security Engineer
eCTF Technical Lead
Today's Agenda

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

<table>
<thead>
<tr>
<th>2023</th>
<th>2024</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>546 Students</strong></td>
<td><strong>773 Students</strong></td>
</tr>
<tr>
<td>450 COLLEGIATE STUDENTS</td>
<td>633 COLLEGIATE STUDENTS</td>
</tr>
<tr>
<td>96 high school students</td>
<td>140 high school students</td>
</tr>
<tr>
<td>317 undergraduate students</td>
<td>512 undergraduate students</td>
</tr>
<tr>
<td>64 graduate students</td>
<td>97 graduate students</td>
</tr>
<tr>
<td>69 PhD students</td>
<td>24 PhD students</td>
</tr>
</tbody>
</table>

Global Representation
- 773 Participating Students
- 9 COUNTRIES
- 30 US STATES

69 Schools 2023 vs 94 Schools 2024

- Year 1: 20 high school students, 60 undergraduate students
- Year 2: 100 undergraduate students
- Year 3: 140 high school students, 340 undergraduate students
- Year 4: 66 graduate students
- Year 5: 12 PhD candidates

© 2024 MITRE Engenuity, LLC. Approved for Public Release. Document number ME0132
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

© 2024 MITRE Engenuity, LLC. Approved for Public Release. Document number ME0132
Welcome
Ben Linville-Engler
Deputy Director
Chief Investment Strategist
Administering ~$530M in state and federal funds through the next 2 fiscal years.
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
Thank You!

Let’s #getCHIPSdone!
COMPETITION BRIEFING & HIGHLIGHTS

Fritz Stine
Senior Cyber Operations Engineer
Thank You, Participants!

<table>
<thead>
<tr>
<th>United States Coast Guard Academy</th>
<th>University of California, Irvine</th>
<th>University of Colorado, Colorado Springs</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States Air Force Academy</td>
<td>University at Buffalo</td>
<td>Carnegie Mellon University</td>
</tr>
<tr>
<td>Indian Institute of Technology Madras</td>
<td>University of Texas at Arlington</td>
<td>University of Illinois Urbana-Champaign</td>
</tr>
<tr>
<td>Government Polytechnic, Pendurthi</td>
<td>University of Arizona</td>
<td>Tufts University</td>
</tr>
<tr>
<td>University of California Los Angeles</td>
<td>Virginia Tech</td>
<td>University of New Haven</td>
</tr>
<tr>
<td>ISD 196</td>
<td>Oklahoma Christian University</td>
<td>Pace University</td>
</tr>
<tr>
<td>Texas A&amp;M University</td>
<td>University of North Dakota</td>
<td>Air Force Institute of Technology</td>
</tr>
<tr>
<td>Georgia Institute of Technology</td>
<td>University of Washington</td>
<td>United States Cyber Games</td>
</tr>
<tr>
<td>Springfield-Clark County Career Technology Center</td>
<td>University of California, Santa Cruz</td>
<td>Strayer University</td>
</tr>
<tr>
<td>North Carolina State University</td>
<td>Wilmington University</td>
<td>Florida International University</td>
</tr>
<tr>
<td>Delaware Area Career Center</td>
<td>The Ohio State University</td>
<td>Andhra Polytechnic High School</td>
</tr>
<tr>
<td>University of Nebraska Omaha</td>
<td>City College of San Francisco</td>
<td>Symbiosis Institute of Technology</td>
</tr>
<tr>
<td>Michigan Technological University</td>
<td>Western Michigan University</td>
<td>Morgan State University</td>
</tr>
<tr>
<td>The University of Alaska Anchorage</td>
<td>Summit Technology Academy</td>
<td>MITRE ENGenuity</td>
</tr>
<tr>
<td>Dakota State University</td>
<td>Worcester Polytechnic Institute</td>
<td>MITRE C2F2024</td>
</tr>
<tr>
<td>Virginia Tech</td>
<td>Ipnet Institute of Technology</td>
<td>CyberAegis</td>
</tr>
<tr>
<td>Kansas State University</td>
<td>Pace University</td>
<td>Sathyabama University of Science and Technology</td>
</tr>
<tr>
<td>San Francisco State University</td>
<td>United States Air Force Academy</td>
<td>East Tennessee State University</td>
</tr>
<tr>
<td>United States Coast Guard Academy</td>
<td>United States Air Force Academy</td>
<td>Indian Institute of Technology</td>
</tr>
</tbody>
</table>
Unique Competition Design

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
#eCTF2024

## Competition Overview

### 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

What Teams are Given

Functional Requirements

Hardware

Automated Testing

Security Requirements

Example Code (Reference Design)

Organizer Support
Medical Infrastructure Supply Chain Security #eCTF2024

Host Computer

Processor

Sensor

Actuator
Medical Infrastructure Supply Chain Security

Host Computer

Processor

Sensor

Actuator

Component Authenticity

Communication Integrity

Data Security

Component Replacement

The diagram illustrates the components of a medical infrastructure supply chain security system. It includes:

- **Host Computer**
- **Processor**
- **Sensor**
- **Actuator**

The system has phases labeled as Pre-Boot, Boot, and Post-Boot. The phases are:

- **Pre-Boot**
  - List Components
  - Attest Components
  - Replace Component

- **Boot**
  - Boot Medical Device

- **Post-Boot**
  - Secure Comms

The diagram is a visual representation of a system designed to ensure secure communications and component integrity during the boot process.
Security Requirements

The device should only boot if all components are present and valid.
The device should only boot if all components are present and valid.
The device should only boot if all components are present and valid.
Secure data and component replacement should only be able to be accessed with a valid PIN.
Security Requirements

The PINs should remain confidential
Secure communications post-boot should not be able to be forged or duplicated by an attacker.
#eCTF2024

## Attacker Resources

<table>
<thead>
<tr>
<th>Full Access</th>
<th>Used by Technician</th>
<th>Application Processor</th>
<th>Component A</th>
<th>Component B</th>
<th>Replacement Token</th>
<th>Attestation PIN</th>
<th>Physical Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="checkmark.png" alt="Checkmark" /></td>
<td><img src="checkmark.png" alt="Checkmark" /></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Device 0</strong></td>
<td>Operational Device</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><img src="checkmark.png" alt="Checkmark" /></td>
<td><img src="checkmark.png" alt="Checkmark" /></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Device 1</strong></td>
<td>Damaged Device</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><img src="checkmark.png" alt="Checkmark" /></td>
<td><img src="checkmark.png" alt="Checkmark" /></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Device 2</strong></td>
<td>Supply Chain Poisoning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><img src="checkmark.png" alt="Checkmark" /></td>
<td><img src="checkmark.png" alt="Checkmark" /></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Device 3</strong></td>
<td>Black Box</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Team Presentations

Please be respectful during presentations and Q&A

Event is being recorded
Today’s Presentation Agenda

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

Welcome SIGPwny

Currently 2nd place with 26,318 points
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)
<table>
<thead>
<tr>
<th>Title</th>
<th>Team</th>
<th>Status</th>
<th>End date</th>
<th>Labels</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement List Components</td>
<td>Pre-Boot/Attest Subteam</td>
<td>Done</td>
<td>Mar 3, 2024</td>
<td>FR - List Components</td>
<td>Begin Testing</td>
</tr>
<tr>
<td>Implement Attestation</td>
<td>Pre-Boot/Attest Subteam</td>
<td>Done</td>
<td>Mar 3, 2024</td>
<td>FR - Attestation</td>
<td>Begin Testing</td>
</tr>
<tr>
<td>Implement Replacement</td>
<td>Pre-Boot/Attest Subteam</td>
<td>Done</td>
<td>Mar 3, 2024</td>
<td>FR - Replace Components</td>
<td>Begin Testing</td>
</tr>
<tr>
<td>Initial protocol for List Components</td>
<td>Pre-Boot/Attest Subteam</td>
<td>Done</td>
<td>Feb 10, 2024</td>
<td>documentation</td>
<td>Begin Implementation</td>
</tr>
<tr>
<td>Initial protocol for Attestation</td>
<td>Pre-Boot/Attest Subteam</td>
<td>Done</td>
<td>Feb 10, 2024</td>
<td>documentation</td>
<td>Begin Implementation</td>
</tr>
<tr>
<td>Initial protocol for Replacement</td>
<td>Pre-Boot/Attest Subteam</td>
<td>Done</td>
<td>Feb 10, 2024</td>
<td>documentation</td>
<td>Begin Implementation</td>
</tr>
<tr>
<td>Comms Subteam</td>
<td>Comms Subteam</td>
<td>Done</td>
<td>Mar 3, 2024</td>
<td>FR - Boot Verification</td>
<td>Begin Testing</td>
</tr>
<tr>
<td>Implement Boot Verification protocol using HIDE</td>
<td>Comms Subteam</td>
<td>Done</td>
<td>Mar 3, 2024</td>
<td>FR - Secure Comms</td>
<td>Begin Testing</td>
</tr>
<tr>
<td>Implement HIDE protocol</td>
<td>Comms Subteam</td>
<td>Done</td>
<td>Mar 3, 2024</td>
<td>FR - Secure Comms</td>
<td>Begin Testing</td>
</tr>
<tr>
<td>Initial protocol for HIDE secure communications layer</td>
<td>Comms Subteam</td>
<td>Done</td>
<td>Feb 10, 2024</td>
<td>documentation</td>
<td>Begin Implementation</td>
</tr>
<tr>
<td>Initial protocol for Boot Verification</td>
<td>Comms Subteam</td>
<td>Done</td>
<td>Feb 10, 2024</td>
<td>documentation</td>
<td>Begin Implementation</td>
</tr>
<tr>
<td>Build Subteam</td>
<td>Build Subteam</td>
<td>Done</td>
<td>Mar 5, 2024</td>
<td>Attack</td>
<td>Handoff</td>
</tr>
<tr>
<td>Implement fault-injection resistant patterns</td>
<td>Build Subteam</td>
<td>Done</td>
<td>Mar 5, 2024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add secure send/receive C interfaces for POST_BOOT code</td>
<td>Build Subteam</td>
<td>Done</td>
<td>Mar 4, 2024</td>
<td>FR - Build System</td>
<td>Begin Testing</td>
</tr>
<tr>
<td>Add mic, delay.h and led.h support to POST_BOOT code</td>
<td>Build Subteam</td>
<td>Done</td>
<td>Mar 4, 2024</td>
<td>FR - Build System</td>
<td>Begin Testing</td>
</tr>
</tbody>
</table>
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

AP
- Request to Send Message
- Decrypt and Solve Challenge Nonce

Component
- Encrypt Random Challenge Nonce
- Validate Response and Execute Message
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 **complete compromise** of the Component
– We developed an exploit for this vulnerability to extract Component flags and carry out attacks against the AP as well
– **85% of teams were vulnerable to this exploit** since the bug originated from the reference implementation
Attack #1

Attacking boot process with a compromised supply chain
Here is a typical device configuration!
Component B becomes damaged!
An authorized technician orders a new Component...
... and runs the replacement routine on the AP.
The device should be able to boot!
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 I²C Component exploit, we can extract secrets!

Send I²C exploit with payload to extract secret key

Component A
0x11111111

AP
0x11111111
0x33333333

Evil Component
0x33333333

Kₚₛₚₘ

Kₚₛₚₘ
Using the I\textsuperscript{2}C Component exploit, we can extract secrets!
Better Solution: Adding a validation step with unique 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 already provisioned!
In other words: an AP can have two provisioned Components with same ID!
Attack #1: Exploiting Replace Code

- New problem: two same Component IDs means that they share the same I²C address, which will cause bus errors
  - Attacker’s fix: use the simple I²C exploit to disable Component A
  - This is done by changing Component A’s I²C address to 0x00
  - Our Evil Component will handle both validate and boot requests from the AP
Use the \( I^2C \) 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 – fffffff)
  - 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 I²C addresses
Thank you! Any questions?
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

Welcome 0xDACC

Currently 5th place with 6,225 points
<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beau Schwab</td>
<td>Project Manager</td>
</tr>
<tr>
<td>Samuel Goodman</td>
<td>Comms. Director</td>
</tr>
<tr>
<td>Andrew Langan</td>
<td>Lead Developer</td>
</tr>
<tr>
<td>Eli Cochran</td>
<td>Team Advisor</td>
</tr>
<tr>
<td>David Nunley</td>
<td>Developer</td>
</tr>
<tr>
<td>Ezequiel Flores</td>
<td>Developer</td>
</tr>
<tr>
<td>Grayson Seger</td>
<td>Developer</td>
</tr>
<tr>
<td>Henry Reid</td>
<td>Developer</td>
</tr>
<tr>
<td>Cameron Crossley</td>
<td>Developer</td>
</tr>
<tr>
<td>Tyler McColeman</td>
<td>Developer</td>
</tr>
<tr>
<td>Ethan Martindale</td>
<td>Video Editor</td>
</tr>
<tr>
<td>Seth Tydings</td>
<td>Video Editor</td>
</tr>
</tbody>
</table>
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
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

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

template<packet_type_t T> struct __packed payload_t;

// Encrypted packet payload
template<packet_type_t T> struct __packed payload_t<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;
};
```
ATTEST

Strengths

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

Weaknesses

- No delays
- Small key space
- Signature reuse
REPLACE

Strengths

- Hashed token

Weaknesses

- Could not get validation to function

```cpp
mitre - application_processor.cpp

1  tc_sha256_init(&sha256_ctx);
2  tc_sha256_update(&sha256_ctx, buf, 16);
3  tc_sha256_final(hash, &sha256_ctx);

4  if (memcmp(hash, REPLACEMENT_HASH, 32) == 0) {
5      return error_t::SUCCESS;
6  } else {
7      return error_t::ERROR;
8  }
```
**BOOT**

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

### Weaknesses
- Only 1 key pair
- Fault injection during RNG

```cpp
mitre-component.cpp
1 #include "mitre.h"
2
3 // Packet checks and re-construction omitted for brevity.
4
5 uECC_make_key(public_key, private_key, uECC_secp256r1());
6 uECC_shared_secret(rx_packet.payload.material, private_key, shared_secret,
7 uECC_secp256r1());
8
9 #define SHA256_CTX SHA256_CTX
10
tc_sha256_init(&sha256_ctx);
11 tc_sha256_update(&sha256_ctx, shared_secret, 32);
12 tc_sha256_final(hash, &sha256_ctx);
13
14 memcpy(ctr, "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00", 8);
15 memcpy(&ctr[8], &hash[16], 0x8); // AES-128-CTR nonce utilized secure send and receive
16 memcpy(aes_key, hash, 16); // AES-128-CTR key
```
**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
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

Solution

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

```c
1. bzero(g_nKey, BLOCK_SIZE);
2. MXC_TRNG_Init();
3. MXC_TRNG_Random(g_nKey, KEY_SIZE);
4. MXC_TRNG_Shutdown();
```
BINARY LEAK

- Exploit binary leaked in public channel “ucccon_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

© 2024 MITRE Engenuity, LLC. Approved for Public Release. Document number ME0132
Today’s Presentation Agenda

- 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
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

Host Computer

<table>
<thead>
<tr>
<th>UART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Computer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Processor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component 2</td>
</tr>
</tbody>
</table>

Medical Device
Attest Protocol

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

1. PIN
2. $H(PIN \| \text{salt}) \equiv h_{PIN}$
3. Send $chal$
4. $\tau = \text{HMAC}(CID \| \text{chal})$
5. Verify $\tau$ & send $\text{AuthEnc}(adata)$
Replace Protocol

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

![Diagram showing the Replace Protocol process with two steps: 1. Token, CID → CID', and 2. $H(\text{Token}) \neq h_{C1}$]
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.

1. Send chal & Verify AuthEnc(Sign(…))
2. Send chal & Verify AuthEnc(Sign(…))
3. Send boot resp cmd.
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.

1. Send `chal`
2. \( \sigma = \text{Sign}(\text{CID} || \text{chal} || \text{msg}) \)
3. Verify \( \sigma \)
Security Features

Passwords (PINs)

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

Hardware Abstraction Layer (HAL)

- Re-wrote firmware in Rust.
- Memory safe by default!
Build-time Encrypted Data

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

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.

```c
// Transaction over interrupt
if (Flags & MXC_F_I2C_INTFLO_STOP) {
```
I2C Exploit (Cont.) Transaction Threshold

1. Transaction start sets WRITE_START register to be TRUE.

```
if (WRITE_START) {
    REGISTER = (volatile unsigned char*)
```

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

```
&ACTIVE_REG, 1); // Read remaining data
if (ACTIVE_REG <= MAX_REG) {
    available = MXC_I2C_GetRXFifo(MXC_I2C_INTERFACE, (volatile unsigned char*)
    if (available < (I2C_REGS[ACTIVE_REG][WRITE_INDEX],
    FOAvailable(I2C_INTERFACE));
```

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

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

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

---

```
Receive Done Register
```

```
Nop Sled  >>>>> Shellcode  Shellcode Address
```

```
Buffer Overflow
```

```
Code Execution Redirected
```

---

```
Interrupt Vector Table
```

105
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
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
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

**Diagram:**

- High Insulin Values
  - Forwarded to AP
- Low Insulin Values
  - Discarded

**Nodes:**

- **Application Processor**
- **I2C Relay**
- **Insulin Sensor**

**Address Codes:**

- 0xf4
- 0xb8
- 0x09
- 0x24
- 0x94
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
devolution / 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
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 & 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

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

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.

Challenge-response between AP and Component (AP sends the component a random challenge to sign with its public key, and vice-versa).

Success! Boot 😊
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.

---

Challenge-response between AP and Component

AP checks against list to make sure Component was provisioned for it

Challenge-response between AP and Component

Success! Boot 😊
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.

Challenge-response between AP and Component

- AP checks against list to make sure Component was provisioned for it
- Challenge-response between AP and Component
- Success! Boot 😊
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.

AP

Component

Exchange Keys and Certificates

AP Public Key
Certificate: Signature of ID + AP pubkey by Host

Component Public Key
Certificate: Signature of ID + Component pubkey by Host

Both sides verify the certificate signatures

Challenge-response between AP and Component

Success! Boot 😊
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.
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.

**Secure Communication**

AP decrypts this with its secret key and obtains K

- Message encrypted with K: Dispense Insulin
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

Today’s Presentation Agenda  

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
Welcome

BugEaters

Currently 11th place with 3,265 points
The BugEaters
University of California, Irvine

Team Leader: Peter the Bugeater

Presented by:

- Jinyao Xu
- Zhanhao Ruan
- Richard Sima
- Zuhair Taleb
- Yintong Luo
- 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:

\[
E[X] = \int_a^b \frac{1}{(b-a)} \, dx
\]

\[
= \frac{1}{(b-a)} \int_a^b \, dx
\]

\[
= \frac{a+b}{2}
\]
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

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

#eCTF2024
University at Buffalo

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
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

Component
- Encrypted Attest Data
- Encrypted Boot Msg
  - Decrypt

AP
- Encrypted Boot Msg
  - Decrypt

Flags
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

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
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
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

<table>
<thead>
<tr>
<th>Encryption At-Rest of <em>Everything</em></th>
<th>Custom Hardened Physical Link Layer</th>
<th>Encrypted Link Layer Wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Nonces to Prevent Replays</td>
<td>ChaCha-Poly AEAD for encryption</td>
<td>Board RNG + von-Neumann</td>
</tr>
<tr>
<td>Minimal External Code Surface</td>
<td>Avoid Interrupts &amp; Async Code</td>
<td>Random Delays + Redundant Checks</td>
</tr>
</tbody>
</table>
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: *4-5 flags in 90 seconds from ZIP download*
- **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.)
Thank you!
Welcome

DOUG SANTOS

Director, Advanced Threat Intelligence
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?

The Stack For Fun And Profit

.o0 Phrack 49 00.
Volume Seven, Issue Forty-Nine
File 14 of 16
BugTraq, r0t, and Underground.Org
bring you

The paper that changed it all

#phrack @ freenode

Internet in the late 90’s was like
What ARE you doing?

$ whatis dsantos
Tell me again WHY are you doing this?

$ man dsantos

THE LACK OF AWARENESS ABOUT MITRE CTID
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

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
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

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

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

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 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

<table>
<thead>
<tr>
<th>Offensive Points: 91,106</th>
<th>Design Flags: 55,400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational Pump Swap: 15583</td>
<td>Final Design: 28000</td>
</tr>
<tr>
<td>Operational Pin Extract: 12732</td>
<td>Boot Reference Design: 5800</td>
</tr>
<tr>
<td>Damaged Boot: 12727</td>
<td>Debugger: 4900</td>
</tr>
<tr>
<td>Supply Chain Boot: 12972</td>
<td>Read the Rules: 6500</td>
</tr>
<tr>
<td>Black Box Boot: 12721</td>
<td>Design Document: 5400</td>
</tr>
<tr>
<td>Supply Chain Extract: 12201</td>
<td>Testing Service: 4800</td>
</tr>
<tr>
<td>Black Box Extract: 12169</td>
<td>Poster: 15802</td>
</tr>
<tr>
<td></td>
<td>Post-Scoreboard: 46,687</td>
</tr>
</tbody>
</table>

Preliminary Scoreboard Results

All Teams Scores

Attack Phase Scores

#eCTF 2024

**Attack Phase Scores Over Time**

**Attack Phase Scores - Top 20**

![Chart showing the points for top 20 teams over time.]

**Race for Second**

![Chart showing the points for UIUC and Purdue over time.]

Race for Fifth

The Race for Fifth

The Race for Fifth (Last Day)
Third Place
$2,500
Third Place

#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

26,926 Final Points
166 Flags Captured
Second Place
$5,000
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, Parithmaal 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 Tirumala, Yaoyu Wu

Advised by: Professor Kirill Levchenko, PhD

28,660 Final Points
173 Flags Captured
First Place
$10,000
First Place

#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

41,581 Final Points
175 Flags Captured
### #eCTF2024

#### Final Scores: Top Teams

<table>
<thead>
<tr>
<th>Rank</th>
<th>Change</th>
<th>Team</th>
<th>Scoreboard Score</th>
<th>Final Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Carnegie Mellon University</td>
<td>39052</td>
<td>41581</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>University of Illinois Urbana-Champaign</td>
<td>26318</td>
<td>28660</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Purdue University</td>
<td>24501</td>
<td>26926</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>University at Buffalo</td>
<td>10422</td>
<td>12558</td>
</tr>
<tr>
<td>5</td>
<td>+1</td>
<td>Indian Institute of Technology Madras</td>
<td>6135</td>
<td>8275</td>
</tr>
<tr>
<td>6</td>
<td>-1</td>
<td>Delaware Area Career Center</td>
<td>6225</td>
<td>8216</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>Ecole Royale de l'Air</td>
<td>5844</td>
<td>7447</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>Indian Institute of Technology Indore</td>
<td>4660</td>
<td>6614</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>Michigan State University</td>
<td>4483</td>
<td>6393</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>University of Nebraska Omaha</td>
<td>3506</td>
<td>5644</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>University of California, Irvine</td>
<td>3265</td>
<td>4964</td>
</tr>
<tr>
<td>12</td>
<td>+4</td>
<td>Florida Atlantic University</td>
<td>1850</td>
<td>3974</td>
</tr>
<tr>
<td>13</td>
<td>+2</td>
<td>Tennessee Tech University</td>
<td>1863</td>
<td>3843</td>
</tr>
<tr>
<td>14</td>
<td>+3</td>
<td>Tufts University</td>
<td>1757</td>
<td>3803</td>
</tr>
<tr>
<td>15</td>
<td>+6</td>
<td>University of Connecticut</td>
<td>1530</td>
<td>3698</td>
</tr>
<tr>
<td>16</td>
<td>-4</td>
<td>Colombe Academy of Technology</td>
<td>2417</td>
<td>3540</td>
</tr>
<tr>
<td>17</td>
<td>+7</td>
<td>East Tennessee State University</td>
<td>1527</td>
<td>3376</td>
</tr>
<tr>
<td>18</td>
<td>-5</td>
<td>Kilgore College</td>
<td>2290</td>
<td>3368</td>
</tr>
<tr>
<td>19</td>
<td>+1</td>
<td>United States Coast Guard Academy</td>
<td>1533</td>
<td>3316</td>
</tr>
<tr>
<td>20</td>
<td>+6</td>
<td>Virginia Tech</td>
<td>1519</td>
<td>3268</td>
</tr>
<tr>
<td>21</td>
<td>-7</td>
<td>Michigan Technological University</td>
<td>1867</td>
<td>3088</td>
</tr>
<tr>
<td>22</td>
<td>+1</td>
<td>Massachusetts Institute of Technology</td>
<td>1528</td>
<td>3020</td>
</tr>
<tr>
<td>23</td>
<td>-4</td>
<td>Shawnee Mission Center for Academic Achievement</td>
<td>1656</td>
<td>2572</td>
</tr>
<tr>
<td>24</td>
<td>-2</td>
<td>United States Air Force Academy</td>
<td>1529</td>
<td>2445</td>
</tr>
<tr>
<td>25</td>
<td>-7</td>
<td>University of Texas at Arlington</td>
<td>1751</td>
<td>2412</td>
</tr>
<tr>
<td>26</td>
<td>-1</td>
<td>University of Colorado, Colorado Springs</td>
<td>1521</td>
<td>2386</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>Oklahoma Christian University</td>
<td>1313</td>
<td>2195</td>
</tr>
<tr>
<td>28</td>
<td>-1</td>
<td>CyberAegis</td>
<td>1401</td>
<td>2012</td>
</tr>
<tr>
<td>29</td>
<td>+12</td>
<td>ISD 196</td>
<td>509</td>
<td>1198</td>
</tr>
<tr>
<td>30</td>
<td>+10</td>
<td>Pace University</td>
<td>512</td>
<td>1192</td>
</tr>
<tr>
<td>31</td>
<td>+6</td>
<td>University of New Haven</td>
<td>522</td>
<td>1044</td>
</tr>
</tbody>
</table>

CLOSING REMARKS

Dan Walters
Senior Principal Microelectronics Solution Lead

© 2024 MITRE Engenuity, LLC. Approved for Public Release. Document number ME0132
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!