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

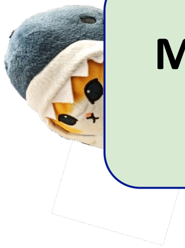
Carnegie Mellon University Computer Science Department
CyLab Carnegie Mellon University Security and Privacy Institute
Carnegie Mellon University Electrical & Computer Engineering
ini Carnegie Mellon University Information Networking Institute

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

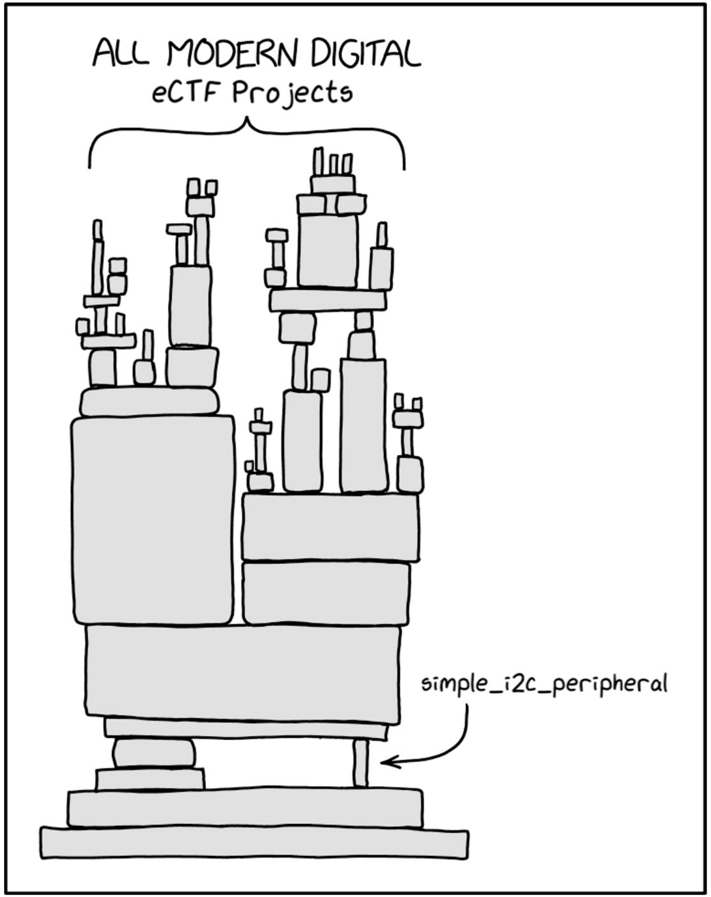
- **Thought Experiment:** Assume full hardware compromise
  - How to defend flags? Can we use fun crypto tricks?
- *BB Boot / BB Extract*: Encrypt comp. secrets w/ key stored in AP
- *Op. PIN Extract / SC Extract*: Encrypt keys inside AP w/ PIN
  - *Potential for offline brute-force if AP compromised
- *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	**Infineon**
AT&T	**Nokia Bell Labs**
AWS	**Rolls-Royce**
Cisco	**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!

Extra Slides