Team Cacti

University at Buffalo

Gaoxiang Liu, Zheyuan Ma, Alex Eastman, Xi Tan, MD Armanuzzaman, Sagar Mohan, Afton Spiegel, and Sai Bhargav Menta
Advised by: Prof. Ziming Zhao and Prof. Hongxin Hu

eCTF
EMBEDDED CAPTURE THE FLAG

Design Overview

Attest

The AP signs a component's random number and sends back the signature for validation. After validation, the component sends encrypted attestation data to the AP for decryption.

Boot

The AP verifies each component by having it sign a random number; the components then validate the AP similarly. On successful validation, both the AP and components boot up.

Post-Boot Communication

To ensure message integrity, messages are signed with the slave device's I^2C address, and a random number provided by the recipient. If the recipient can verify the signature, it trusts the data.

PIN and Token Checking

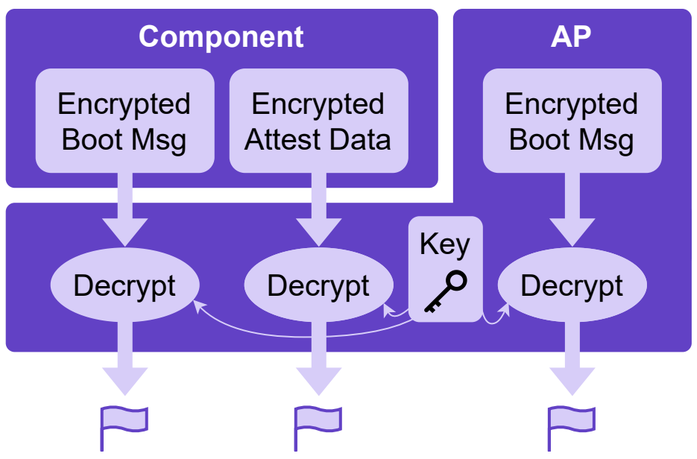
We only store the Argon2 keyed-hash values of the PIN and token. User input is also hashed, and a constant-time comparator is used to verify hash values.

Secrets	Key1_Public	Key1_Private	Key2_Public	Key2_Private	AEAD Key	Hash Key & Salt
Stored at	AP	Component	Component	AP	AP	AP
Description	Create signature for AP's message	Validate signature for AP's message	Create signature for AP's message	Validate signature for AP's message	Decrypt the attestation data and boot message	Secure attestation PIN and replace token

Defensive Highlight

Encrypted Sensitive Data

- Flags in the boot message and attestation data are encrypted and stored securely
- Only the AP holds the decryption key
- Even if attackers can read a component's flash memory and SRAM, the flags remain secure



Memory Wiping

- After using sensitive data like keys, fill the memory location with zeros to securely erase it

Mitigating Bruce-Force Attacks

- Use the Argon2 keyed-hash algorithm for the inputted PIN and Token; this method is intentionally slow
- Introduce a deliberate delay during the checking
- Longer delay after a failed attempt, the delay is still in effective even resetting the board
- A longer delay after a failed PIN or Token attempt, which remains effective even after resetting the board

Mitigating Fault-Injection Attacks

- Introduce random delays of several hundred CPU cycles
- Execute important conditional expression twice; for example, branching on authentication or password validity

Offensive Highlight

Attacks

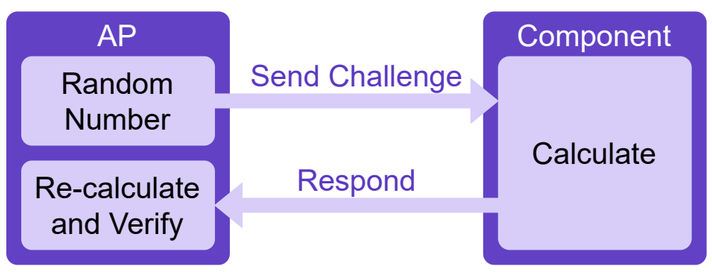
Brute-Force	Try all possible PIN codes
Shared Secrets	Reuse known keys between deployments
Replay Attack	Capture and replay authentic messages
Predictable Keys	Key value is 0 due to global variables

Exploiting Weak Validation Designs

- If a post-boot communication message has a fixed checksum for the same message, it can be captured and replayed, allowing the checksum validation to still succeed.
- Many teams use pre-boot validation to confirm the authenticity of the AP and components by sending a fixed secret value. However, this value can be intercepted during communication or extracted by malicious firmware, leading to a security breach as the secret is exposed.

How to Fix It

- Ensure that the checksum for a message is unique each time, even if the message itself is unchanged
- Do not reuse the same secret for different validation processes
- Ensure that any random number used in the validation process is unpredictable to enhance security
- Use the challenge-response mechanism to authenticate, rather than using a fixed value (See the figure below, where the AP tries to authenticate the component)



The research of Team Cacti is supported in part by a National Centers of Academic Excellence in Cybersecurity grant (H98230-22-1-0307).