PARED - Protective Automotive Remote Entry Device

Arun Krishna AMS (EE19B001), guided by Prof. Chester Rebeiro

1 Introduction

The **Embedded-Capture-The-Flag** competition conducted by MITRE is a 13-week-long two-phase competition consisting of the design-build & attack phases.

During the design phase(first phase), a secure embedded firmware for the Protected Automotive Remote Entry Device System (PARED) for the keyfob-car pair must be developed. The attack phase(second phase) involves analyzing and attacking the competitor's design to identify security flaws that enable unauthenticated unlocking of a car & enabling its features.

2 Challenge Overview

2.1 Motivation

The car's owner will receive a pre-paired fob that can unlock the car. The generic unpaired fobs produced by the manufacturer can be paired with the car when the paired fob and the pairing pin are available. If a car owner pays for an additional feature, the manufacturer will send an image file that must be uploaded into the existing paired fob, enabling the feature when it unlocks the car.

2.2 Security Requirements

- The car should be unlocked only by the authentic fob that is paired with the car. (`SR1`)
- Revoking an attacker's physical access to a fob should revoke their ability to unlock the car. (`SR2`)
- Snooping the communication between the paired fob and the car while unlocking must now allow the attacker to unlock the car. (`SR3`)
- An unpaired fob can be paired with a car only when both the car's paired fob and the pairing pin are available. (`SR4`)
- Owner cannot enable new features in the paired fob that the manufacturer has not packaged. (`SR5`)
- A feature packaged for one car must not allow an attacker to enable the same feature on another car. (`SR6`)

2.3 Attack Phase Objective

Firmware generated for the car and fobs will be run on **Texas Instruments Tiva-C Microcontrollers(TM4C123GH6PM)**. The car-pairedfob & pairedfob-unpairedfob interact through the UART interface. Generated binaries by the players are encrypted and Encrypted binaries can be run only on the keyed boards provided by the organizers.

During deployment, the firmware for the unpaired fob and fob-car pairs will be built. The unpaired fob can be paired with any car within the same deployment (but not possible across deployments).

Multiple car-paired-fob pairs are generated during each deployment. For each pair, the attacker has access to the following resources. (Rules have been simplified for this report. Refer to the original rules document)

- The attacker has access only to the encrypted binary of the unpaired fob.
- `Car 0`: The attacker has full access: Unprotected plaintext binaries of car 0, paired fob & its pairing pin.
- `Car 1`: Encrypted image of the car only(paired fob not available)
- `Car 2`: Permanent access to the encrypted image of the car and temporary access to the encrypted image of the fob
- `Car 3`: Encrypted image of car & paired fob is available

The objective of the attacker is to obtain the secret flags by

- Unlocking `Car 1` (paired fob not available) (`SR1`)
- Unlocking `Car 2` when the paired fob is disabled/not available (`SR2` & `SR3`)
- Find the pairing pin of `Car 3` to pair an unpaired fob (`SR4`)
- Enable a feature in `Car 3` that is not packaged for the car (`SR5` & `SR6`)

2.4 Threat Model

The secret flags are stored in the EEPROM of cars. The organizers encrypt the firmware of `Car 1,2 & 3` and they cannot be decrypted by the attacker. Thus, secrets cannot be directly obtained. Only the MITRE-provided secure bootloader in the keyed microcontrollers can decrypt and run the encrypted firmware. The secure bootloader is not in the scope of the attack. Malicious code cannot be injected into the keyed boards through direct means.

While the generic source code of the firmware is available to the attacker, the cryptographic secrets used in building the firmware are not available to the attacker. The attacker has physical access to the keyed microcontrollers running the firmware.

3 Design Phase

3.1 Protocol

The entire design philosophy revolved around the usage of symmetric encryption because of the memory and computation bottlenecks (TM4C123: ARM Cortex M4, 32kB SRAM & 256 kB Flash).

NIST recommended **Lightweight Cipher ASCON-128a with AEAD** encryption is used to provide confidentiality, integrity, and authenticity of the messages. Designs without them are prone to man-in-the-middle attacks. ASCON is chosen because of its high cryptanalytic security, lightweight, interleaving capabilities(for 32bit), software efficiency, and side-channel countermeasures. Encryption keys, and other cryptographic secrets are stored in the EEPROM.

3.1.1 Unlock Process

A secure unlock protocol over - insusceptible to replay, roll-jam, and roll-back attacksis realized (satisfying `SR1`, `SR2`, and `SR3`) with ASCON AEAD encryption for message transfer.

A randomly generated challenge-response pair is used to authenticate the fob and the car for the unlock process.

- The car sends a random token encrypted using K_1 to the fob. The validity of the generated token is limited up to the threshold time to prevent roll jam attacks.
- Fob decrypts the received message (using K_1) and sends token and enabled feature information after encrypting the whole message using another K_2.
- Car decrypts the received message(using K_2); verifies and validates the received token & car-ID. If found correct, the car is unlocked.

The keys K_1 and K_2 are pre-shared to both car and fob during firmware build-time. The encryption keys differ for each fob-car pair (within and across deployments). Thus, for example, `fob 0` cannot unlock `car 1`.

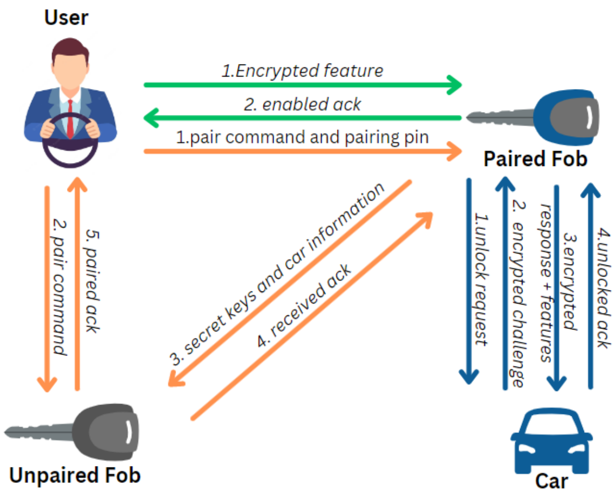
3.1.2 Pairing Process

The User sends the pairing pin to the paired fob. After verifying the pairing pin, the paired fob sends the pairing pin, car ID, and encryption keys used in unlocking & feature-enabling processes to the unpaired fob (satisfies `SR4`).

3.1.3 Packaging and Enabling Feature

The feature information and car-ID are encrypted(ASCON AEAD) and packaged into a binary when the manufacturer packages a feature(`SR5`). The key used for encryption is pre-shared to the fob during the firmware-built process. The package-encryption key is unique for each fob(`SR6`).

The encrypted package is sent to the paired fob, which decrypts, authenticates, compares the car-ID, and enables the feature.



Protocol

3.2 Implementation Specifications

3.2.1 UART Communication

The communication between paired fob-car, paired fob-unpaired fob & user-fob is through the UART interface. Significant challenges were faced while implementing the UART communication between the microcontrollers because of loss of synchronization resulting in break and frame errors.

Several methods were tried to resolve the issue but failed: decreasing the number of data bits, reducing the baud rate, and increasing the number of stop bits. The frame errors were because of weak-pull up strength of the GPIO pins and were finally resolved by attaching a weak internal pull-up resistor to the UART pins. This problem delayed our entrance to the attack phase by two weeks.

Each communication session is valid only within a specified timeout to avoid a deadlock situation. The session is exited if the specified number of bytes is not received within the timeout.

3.2.2 Random Number Generation

Challenge-Response authentication used in the unlock process requires random number generation. Since TM4C123GH6PM does not have hardware RNG, the seed for the RNG must be from other multiple high-entropy sources to prevent replay attacks(`SR2` and `SR3`).

- The seed for the srand() function is created using a combination of
  - Initial values in SRAM at the startup. Part of SRAM is de-initialized in the linker `.ld` file.
  - A random salt is generated while building the firmware that is unique to each built firmware.
- The value of rand() is combined with System Tick to generate the random number.

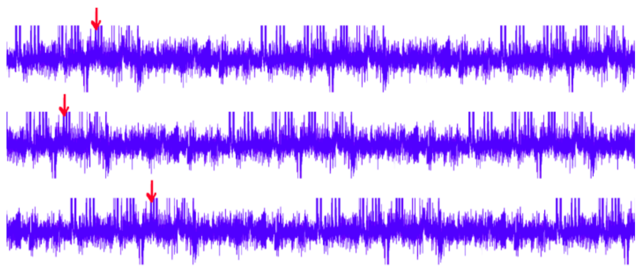
3.3 Attack Prevention Measures

3.3.1 Preventing Stack-based attacks

- **Zeroing of the stack**: Overwrites the local variables stored in the stack once they are not required to prevent *stack leaks*
- **Non-Executable stack**: SRAM permissions were set to read + write to prevent *malicious stack payload execution*
- **Fixed message size**: The protocol is implemented such that messages transferred through UART are of pre-defined fixed size, i.e., inputs are of a fixed number of bytes to prevent *buffer overflow attacks.*

3.3.2 Preventing Side-Channel attacks

Addition of Random Delays: Random delays in the execution will introduce desynchronization when collecting traces, increasing the complexity of mounting a side-channel attack.



Desynchronization of traces*

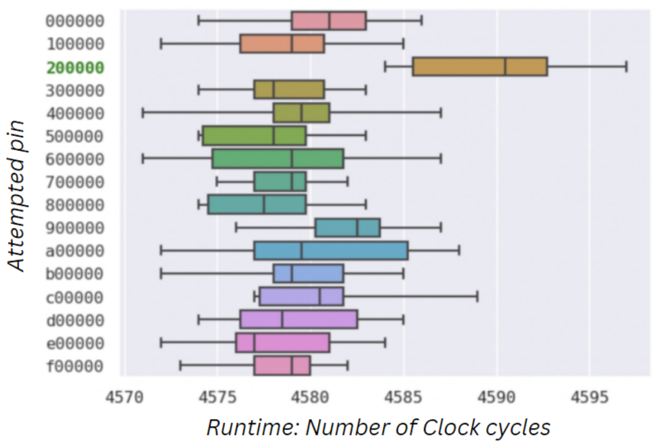
3.3.3 Preventing Fault Injection-based attacks

- **Addition of Random Delays**: Because of random delays, the target point in time is randomly changing its position. Thus the attacker has to inject faults numerous times until the target & fault coincide.
- **Use of Internal Oscillator**: The external oscillator on the evaluation board is prone to tampering to inject fault through *clock glitching.* Clock glitching is prevented using a precision internal oscillator, which cannot be tampered externally.
- **Brownout Protection**: Any brownout/tampering with voltage to cause *voltage glitching* is detected and will result in a system reset using TM4C123's internal features. The efficacy of this method is not yet tested.

* *Figure for illustrative purposes only*

3.3.4 Preventing Timing-based attacks

- **Addition of Random Delays**: Timing attacks are difficult to implement since accurate time measurement is prevented.
- **Constant-time Comparisons**: Use of timing-attack vulnerable 'memcmp' and 'strcmp' functions were avoided. These functions compare one character at a time and return when the first incorrect character is encountered, making the execution time vary based on the number of correct characters in the given input. By using constant-time comparisons for array, these attacks are avoided



Timing attack: 1st character of pairing pin*

3.4 Design Improvements

3.4.1 Random Number Generation

Only one of the competitors retrieved a flag from our design from `Car 2` through replay attacks because of low entropy RNG. Unexpectedly the MITRE-provided secure bootloader zeroes the SRAM entirely on startup before loading the image, which went unnoticed by us. Thus initial values for SRAM at startup cannot be used as a source of entropy. Instead, the seed for RNG must be chosen from sources like

- Internal temperature sensor of ADC.
- Clock drift between Hibernation Real Time Clock and CPU Clock

We can build upon this by putting the output of each entropy source through a Von Neumann Extractor to debias and hashing them together to obtain the seed for RNG.

3.4.2 Brute Force Prevention

According to competition rules, the pairing pin used in the pairing process of an unpaired fob with a car must be a 6-digit hexadecimal password. Thus the pairing pin has an entropy of 24 bits and can be broken by brute force. Intentional delays(within the time requirements specified by the organizers: 1 second for the correct pairing pin and 5 seconds delay for the incorrect pin) were added to increase the time for computation - thereby preventing brute-force attacks within the competition timeframe.

Mounting the brute-force attack can be made impossible within the competition timeframe by ensuring that the 5-second delay always happens if a wrong pin is entered. Our current design is vulnerable because if the attacker realizes the pin is incorrect, he can reset the system, bypassing the 5-second delay.

We can use a counter value stored in the flash memory to prevent this. If three failed attempts are detected, we delay for atmost 5 seconds on each further attempt. This counter value is set to 0 only when the pairing process is run successfully. Since it is stored in flash memory, the count will persist even when the device is reset.

3.4.3 Pairing Pin: Single Point of Failure

During the pairing process in our current design, if the pairing pin is incorrect, the paired fob openly shares the cryptographic secrets with the unpaired fob without any encryption. While all the security requirements of the competition are satisfied, they cannot be realized in real life since replay attacks could result in unauthenticated pairing.

The pairing pin is also the single point of failure in the current design. If the pairing pin is obtained, the attacker can obtain all the cryptographic secrets directly. Thus, the cryptographic secrets must be encrypted to avoid snooping while sending them to the unpaired fob.

Since the unpaired fob is common across the entire deployment, symmetric encryption with a fixed key cannot be used because the attacker can obtain the key from the paired fob of `Car 0`(Refer section 2.3). Instead, a session key for the pairing process must be created on the fly for each session(preventing replay attacks) using authenticated secure key exchange protocol for the encrypted sharing of the cryptographic secrets. Since the private keys are unknown to other parties, confidentiality can be ensured.

4 Attack Phase

Once the functionality of the design we created was verified for its adherence to the competition rules, we moved to the attack phase on March 21st. We performed *Man-in-the-middle attacks, Brute-Force attacks, Replay attacks,* & *stack-smashing attacks* to obtain most of the flags.

4.1 Man-In-The-Middle Attacks

In the attack phase, the devices were placed in an adversarial environment where the attacker could act as a 'Man-in-the-middle,' intercepting and maliciously modifying the packets during transmission. The received message packets must be checked for their integrity and authenticity through HMAC or authenticated encryption. Designs without them were prone to 'Man-in-the-middle' attacks.

As an attacker, our role was to convince the controller to accept a maliciously modified packet. Designs which had message packets without supplied hash were prone to tampering. We enabled additional features in some designs through this method.

```
packet = read_input()
malicious_packet = modify_packet(packet)
write_output(malicious_packet)
```

Certain designs had message packets with supplied hash that the receiver could use to perform integrity checks. But as long as the attackers can perform the hash calculation on their own, they can determine the correct value and send the modified message + hash convincing the controller to accept the flag as valid. HMAC message construction must be used to prevent such tampering.

```
packet = read_input()
message, hash = parse_packet(packet)
malicious_message = modify_message(message)
new_hash = compute_hash(malicious_message)
generate_packet(malicious_message, new_hash)
write_output(malicious_packet)
```

4.2 Brute-Force Attacks

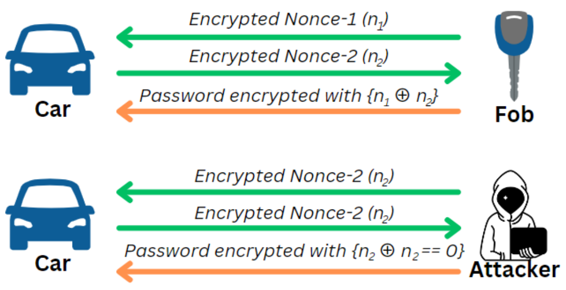
As mentioned in section 3.4.2, Designs without delays are vulnerable to brute-force attacks to obtain the pairing-pin since the number of permutations is just 2^{24}. These attacks can be avoided by implementing the measures mentioned in section 3.4.2

4.3 Replay Attacks

Designs which performed Challenge-Response authentication with poor entropy source for PRNG were susceptible to replay attacks in the unlock process(`Car 2: SR2 & SR3`). The following vulnerabilities were observed in the designs:

- Only Hardcoded PRNG seed was used. Compile-time entropy is insufficient since the same sequence of random numbers will be observed every time the system is reset.
- Certain designs had hardcoded PRNG seed stored in EEPROM which gets updated every time the system is reset/powered on. The system is vulnerable because the attacker can flash the image and capture the communication traces. Once sufficient traces are captured, the attacker reflashes the image again, and the captured communication traces can be replayed to unlock the car.
- Designs that use only the system tick timer or interrupt-based timers are prone to replay attacks. Communication traces can be captured by starting the unlock process at the precise time known to the attacker. The attacker can unlock the car by simulating the exact conditions in the adversarial environment. Lowering the resolution of the timer increased the difficulty of mounting the attack.

To prevent replay attacks, one of the teams designed their protocol to generate session keys for each unlocking session. To create the session key, the fob generates an encrypted nonce N_1 and shares it with the car. Following that, the car also generates an encrypted nonce N_2 and communicates it with the fob. Both the car and fob decrypt the received nonces, XORs them (N_1 `XOR` N_2), and use the result as the session key.



By forcing the car to choose the same nonce as the fob sends, the attacker can cause the session key to 0. Since the message and key are known, the attacker can unlock the car. Their design is vulnerable because the generated nonce was not completely random.

Their nonces were generated from a constant SEED which was stored in EEPROM. Although it gets updated every time a random number is generated, this method can be broken because the same nonce is observed after reflashing the firmware, thus making the entire design vulnerable.

4.4 Buffer Overflow attacks

Certain designs while having secure protocol, had buffer overflow vulnerabilities while reading unbounded data through UART. A common vulnerable function that was commonly used was:

```
uint32_t uart_readline(
    uint32_t uart, uint8_t* buf)
{
    uint32_t read=0; uint8_t c;
    do {
        c = (uint8_t)uart_readb(uart);
        if ((c!=’\r’)&&(c!=’\n’)&&(c!=’0x0D))
            buf[read++] = c;
    } while ((c!=’\n’)&&(c!=’0x0D))
    return read;
}
```

By sending more data than the size of the buffer, the buffer was overflowed to point the function's return address to the payload location. The execution of the plaintext firmware of `Car 0` was observed and analyzed using `OpenOCD debugger` to generate the payload. The absence of ASLR enabled us to use the same payload on the other cars to obtain secret flags. This can be avoided by following the measures mentioned in section 3.3.1

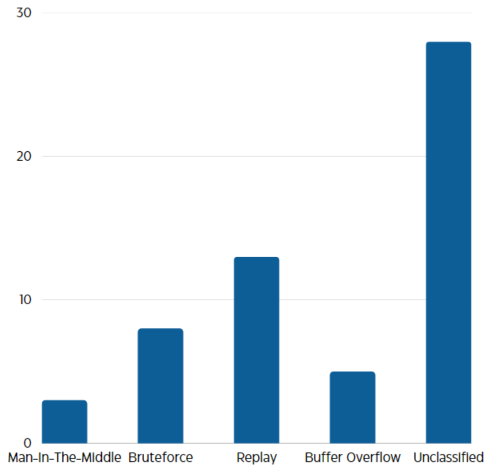
4.5 Miscellaneous attacks

Certain designs didn't create a unique link between cars and fobs. This was exploited by using paired fobs associated with a different car to obtain flags. For example, in some cases `fob 0` can be used to unlock `cars 1 & 2` and obtain their flags despite using secure channels.

4.6 Attacks to be implemented

Following attacks were considered but weren't implemented because of time constraints

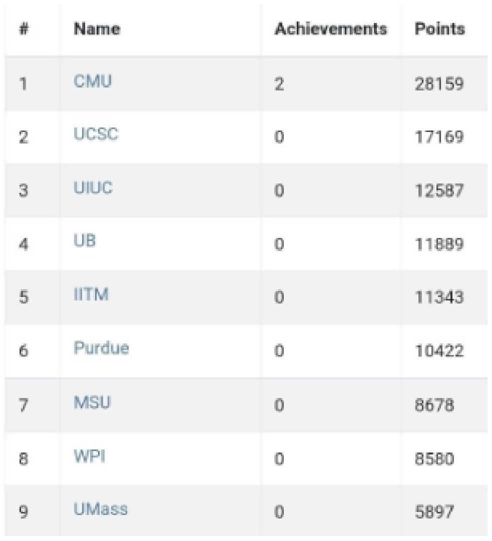
- Timing based attacks on non-constant-time array comparisons utilizing `memcmp` & `strcmp` functions. (Refer to section 3.3.4)
- Power - Side Channel attacks
- Clock Glitching - Fault Injection attacks



Attack phase flags

5 Results

Over 60 teams from different universities around the globe participated in the competition. At the end of the competition, the following were the standings:



#	Name	Achievements	Points
1	CMU	2	28159
2	UCSC	0	17169
3	UIUC	0	12587
4	UB	0	11889
5	IITM	0	11343
6	Purdue	0	10422
7	MSU	0	8678
8	WPI	0	8580
9	UMass	0	5897

We stood **fifth overall**. By snatching 56 secret flags(out of 120), We had the **fourth highest ranking in the attack phase**. Our design was **one of the most secure implementations** of the PARED system - *with only UCSC capturing one of our six secret flags.* Only two other teams were better than us: CMU and UCSC, with no flags broken.

6 Contribution

6.1 Design Phase

- Contributed to the development of the conceptual design of the secure protocol.
- Implemented UART Communication between microcontrollers. Resolved the break and frame errors.
- Implemented System tick functionality and timeout functionality for the UART communication.
- Modified ASCON base code to suit the current software stack for the 32-bit microcontroller.
- Developed individual modules for `unlocking` & `pairing` processes, ensuring stack-protection measures.
- Implemented measures against fault injection(Refer to section 3.3.3): Random delay, Use of internal oscillator, and Brownout protection.
- Assisted in implementing EEPROM Password protection to prevent unauthorized access.
- Conducted multiple code reviews and dependency audits to screen for security vulnerabilities.

6.2 Attack Phase

- Screened through other designs for security vulnerabilities involving buffer overflow, brute-force, man-in-the-middle, replay, and other attacks.
- Majorly attacked designs through brute force, main-in-the-middle, replay, and miscellaneous attacks.
- Assisted in developing a parser tool to understand and analyze the output from the logic analyzer in a human-readable format using MATLAB to implement replay attacks.
- Implemented an automatic replay script for collecting communication traces, reflashing the firmware, and replaying the captured traces for replay attacks.

7 Acknowledgement

I express my sincere gratitude to Professor Chester Rebeiro for his invaluable guidance and support throughout the course of my thesis. His expertise and insightful feedback played a crucial role in shaping our ideas and enhancing the quality of our work. I also extend my heartfelt appreciation to my fellow teammates, Antonson, Divya Prajapati, Priyanka, and Shree Vishnu, for their unwavering dedication and collaboration. I would also like to express my appreciation to Mahesh, Reetwik Das, and Ritwik for their valuable assistance and advice during the competition. Our team's collective efforts and cooperation enabled us to overcome various challenges and achieve our research goals.

Furthermore, I would like to acknowledge the support provided by the Department of Computer Science and Engineering and the Department of Electrical Engineering at IIT-Madras. Lastly, I am grateful to MITRE for organizing this competition & our competitors who made this journey a wonderful learning experience.

8 References

Following links are references to ASCON ciphers and SRAM-based Random Number Generator

- Ascon Lightweight Authenticated Encryption and Hashing
- Aging-Resilient SRAM-based True Random Number Generator for Lightweight Devices

Following links contain `Tiva-C TM4C123GH6PM` Peripheral Library Manual & Datasheet:

- TivaWare Peripheral Driver Library
- Tiva TM4C123GH6PM Microcontroller - data sheet

Our Implementation:

- https://github.com/dv1702/vadap-av/tree/v1.0

The following links contain references to the tools used for development and attack purposes.

- Cross-platform PlatformIO IDE and Unified Debugger
- OpenOCD Debugger

The following links contain the eCTF rule book and the technical requirement document.

- https://ectf.mitre.org/wp-content/uploads/2023/02/2023-eCTF-Rules-v1.1-lr.pdf
- https://ectf.mitre.org/wp-content/uploads/2023/02/2023-eCTF-Technical-Specifications-v1.1.pdf