PARED - Protective Automotive Remote Entry Device

Vishnu, Arun, Antonson, Divya, Priyanka

Indian Institute of Technology, Madras

08-05-2023

Outline

- Challenge Overview
- System Architecture
- Requirements
  - Build PARED system
  - Load Device
  - Functional Requirements
  - Security Requirements
  - Technical Requirements
- Design Phase
- Attack Phase
- Results
- Improvements
- Individual Contributions

Competition Outline

eCTF-2023: The eCTF, organized by MITRE, is unique in two major ways.

1. The focus is on securing embedded systems, which present an entirely new set of challenges and security issues that are not currently covered by traditional CTFs.
2. This event balances offense and defense by including a significant secure-design phase in addition to an attack phase.

Organizers: MITRE is a not-for-profit organization that operates research and development centers sponsored by the federal government. MITRE works with industry and academia to apply science, technology, and systems engineering that enables the government and the private sector to make better decisions

Competition Timeline:

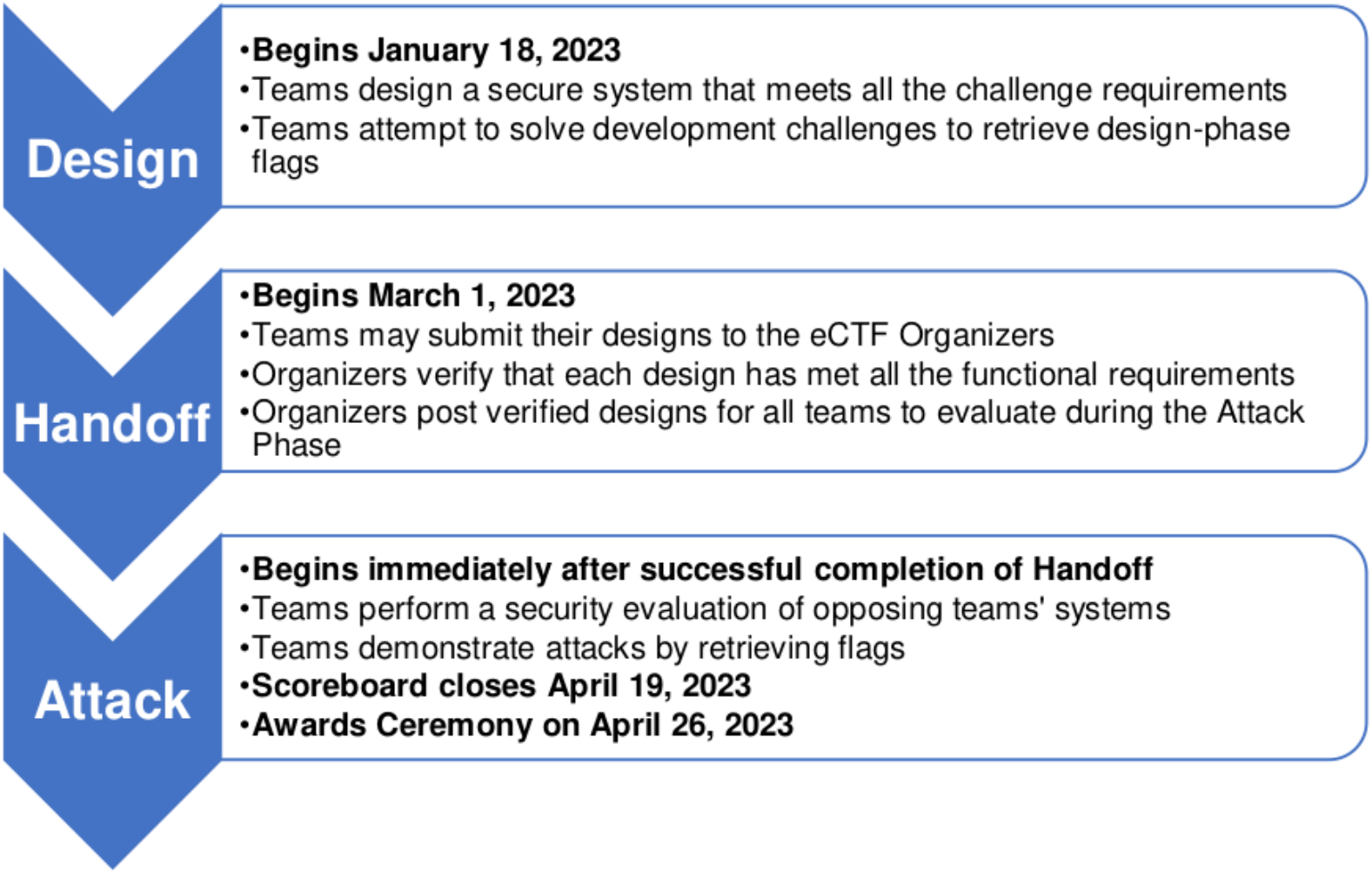
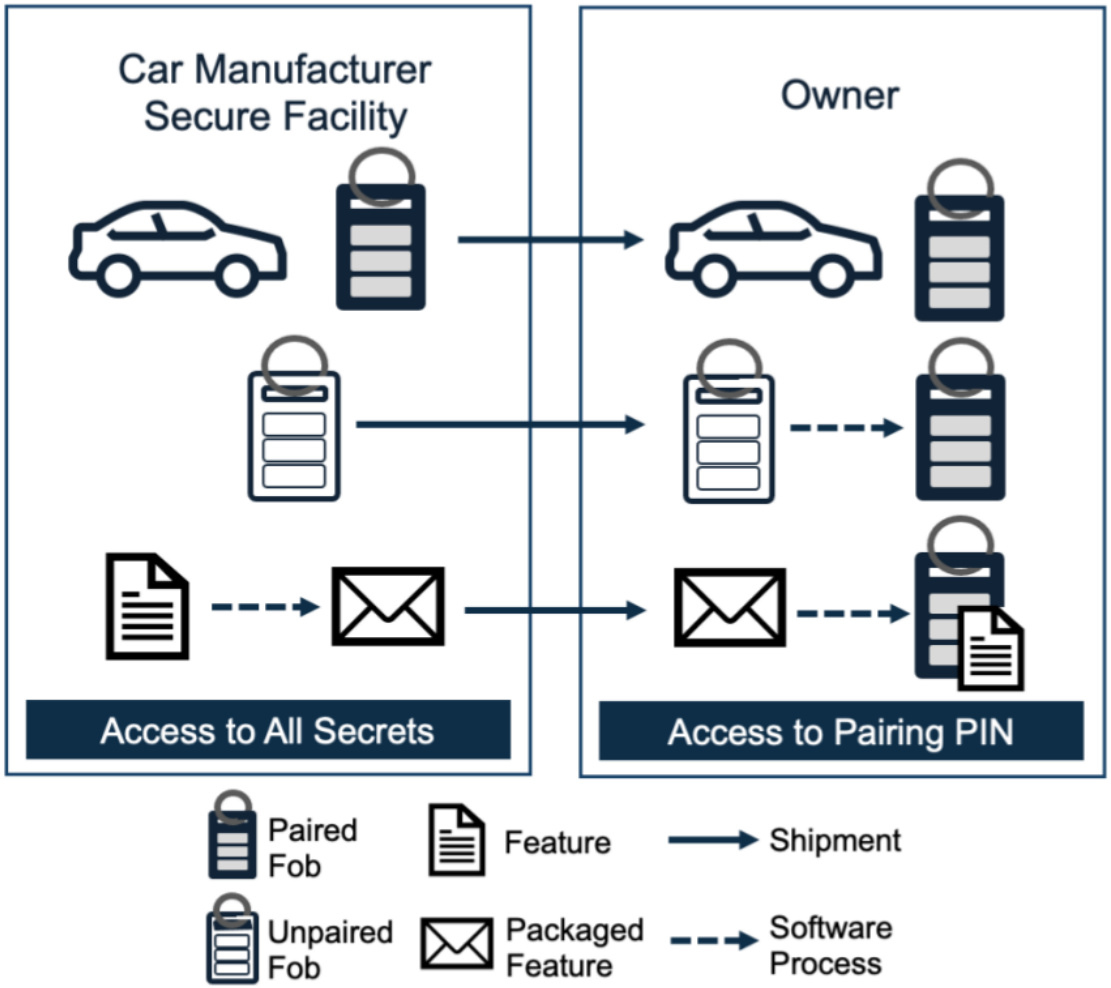


Figure 1: Competition Timeline

Challenge Overview



The owner of a car will receive pre-paired fobs that unlock their car.

The manufacturer will also produce unpaired fobs that any owner can pair with their car.

If a car owner pays for an upgraded feature, then the manufacturer will send them a file to install onto an existing paired fob, which will then enable the upgraded feature on their car.

Figure 2: Overview

System Architecture

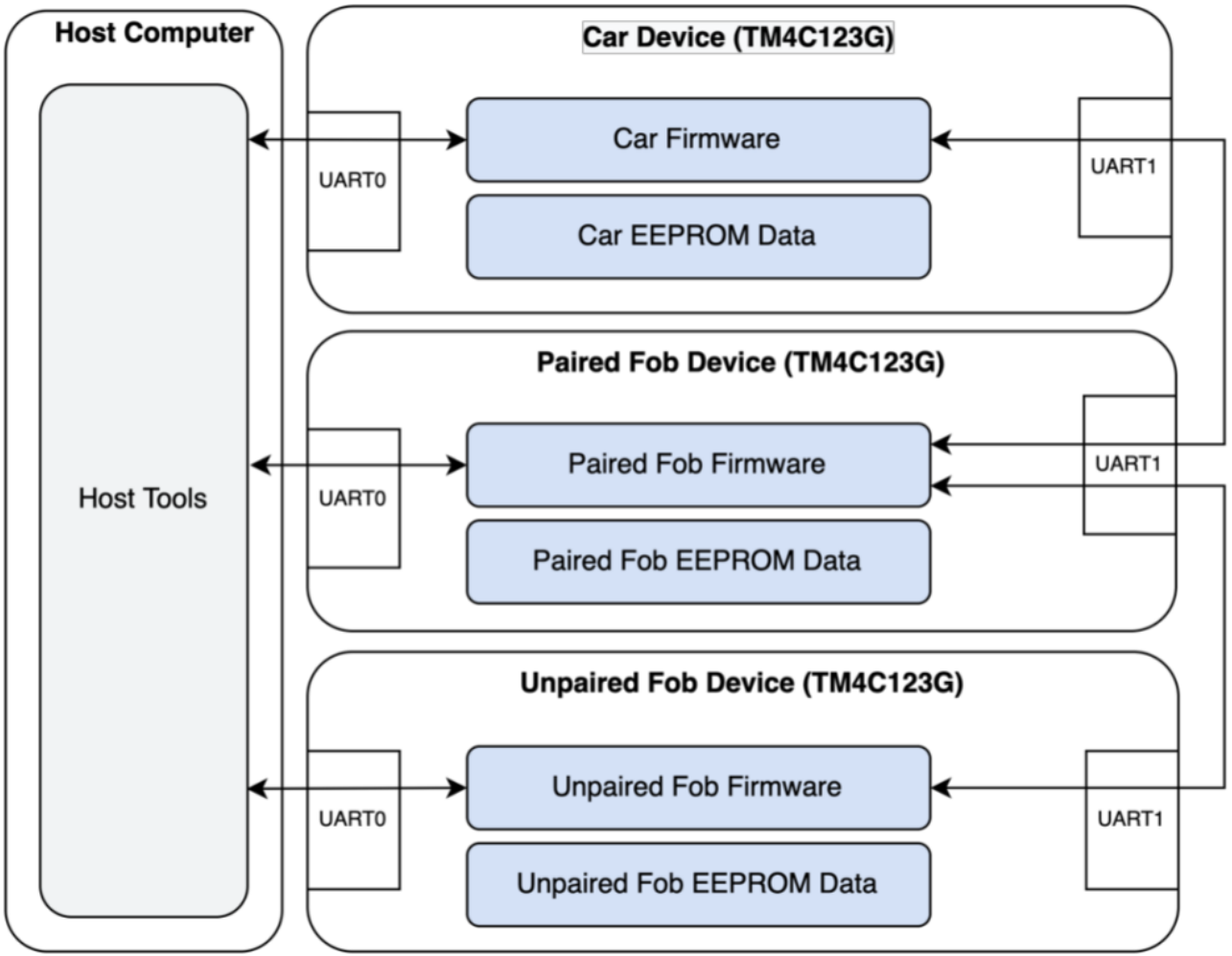


Figure 3: System Components

System Architecture

r

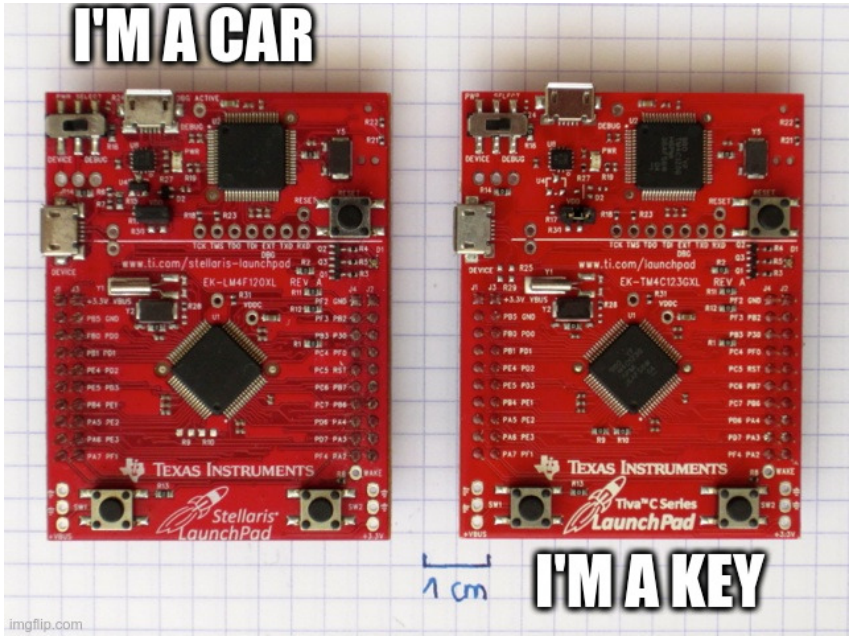


Figure 4: Tiva-C TM4C123G

- Un-keyed boards used for Development
- Keyed boards used for Attack Phase

Requirements

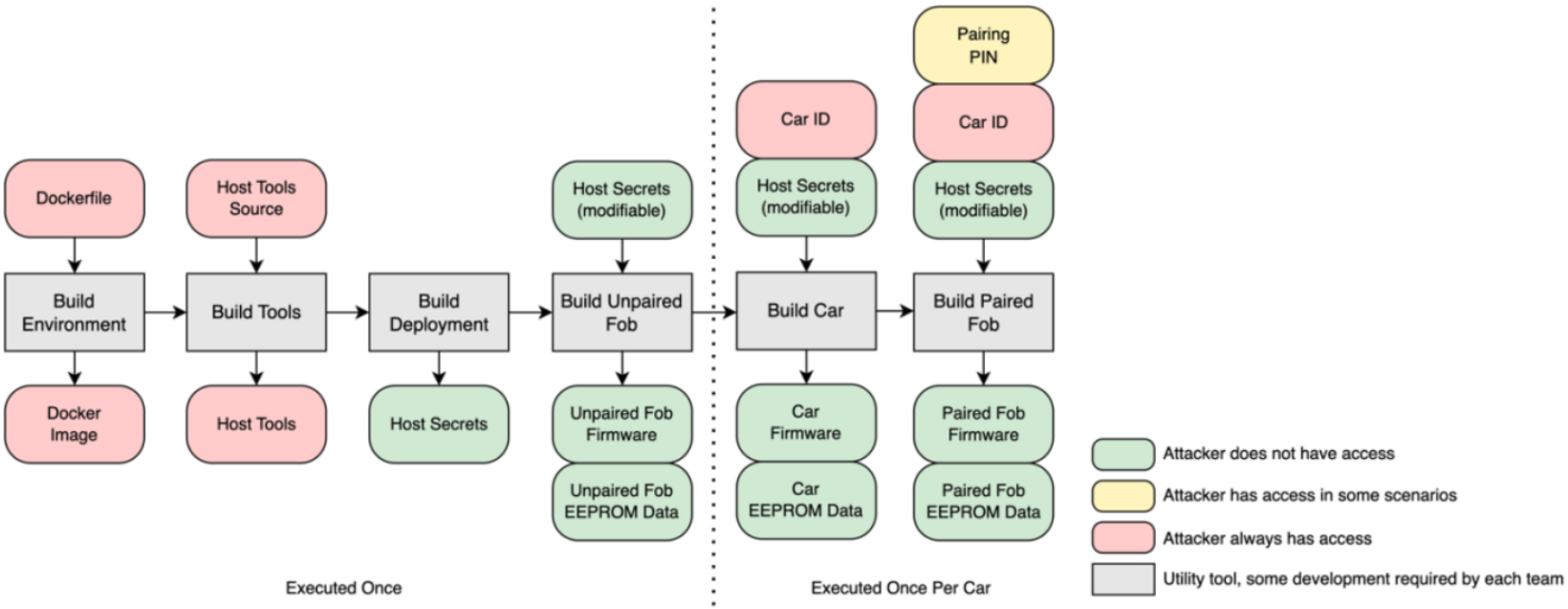


Figure 5: Overview of Build process

Build Steps

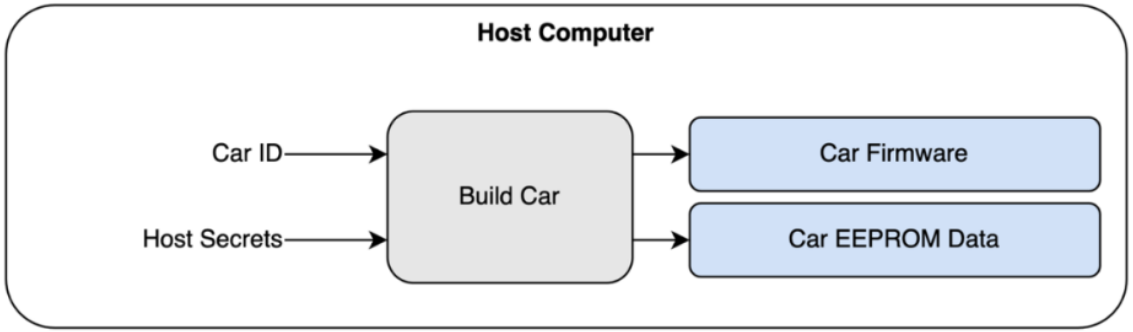


Figure 6: Build Car Steps

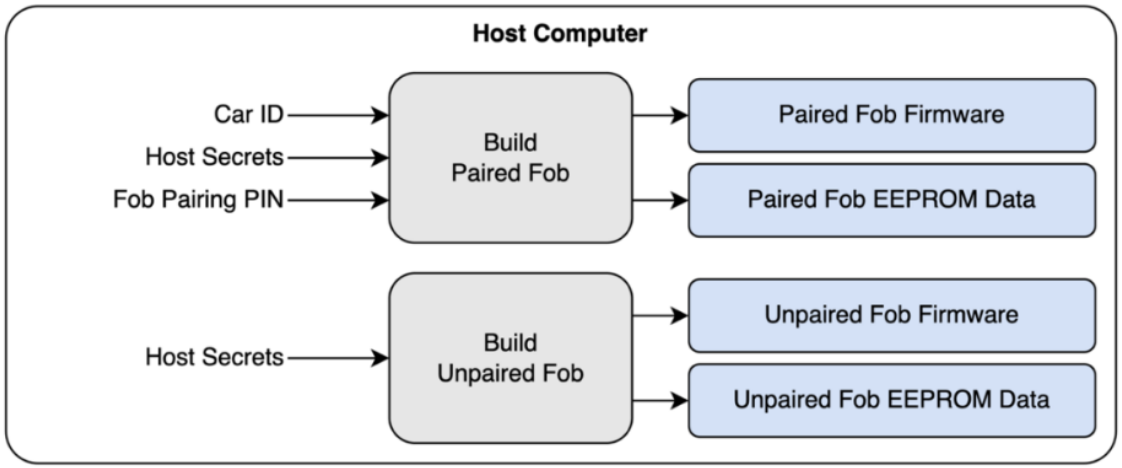


Figure 7: Build Unpaired Fob and Build Paired Fob Steps

Load Devices

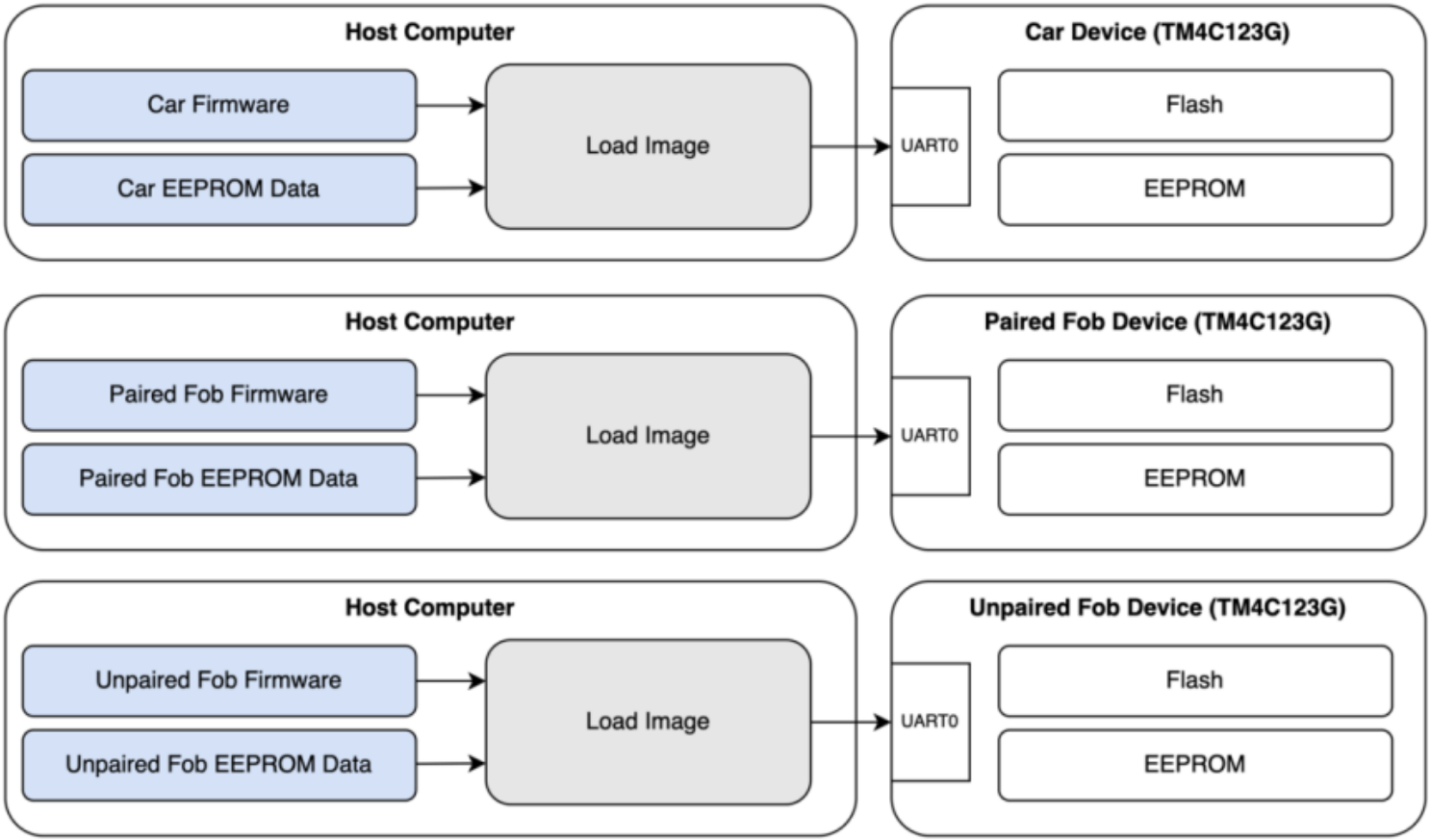


Figure 8: Device Load Step

Functional Requirements

Figure 9: An overview of available tools for the PARED system

Security Requirements

1. A car should only unlock and start when the user has an authentic fob that is paired with the car
2. Revoking an attacker's physical access to a fob should also revoke their ability to unlock the associated car
3. Observing the communications between a fob and a car while unlocking should not allow an attacker to unlock the car in the future
4. Having an unpaired fob should not allow an attacker to unlock a car without a corresponding paired fob and pairing PIN
5. A car owner should not be able to add new features to a fob that did not get packaged by the manufacturer
6. Access to a feature packaged for one car should not allow an attacker to enable the same feature on another car

Attack Cars

✓ = Full Access ✓ = Temporary Access	Car	Paired Fob	Unpaired Fob	Logic Analyzer Capture of Unlock	Pairing PIN	Packaged Feature 1	Packaged Feature 2
Car 0 Your Car (No Flags)	✓	✓	✓		✓	✓	✓
Car 1 New Car	✓		✓			✓	✓
Car 2 Temporary Fob Access	✓	✓	✓			✓	✓
Car 3 Passive Unlock	✓		✓	✓		✓	✓
Car 4 Leaked Pairing PIN	✓		✓		✓	✓	✓
Car 5 PIN Extraction, Enable Feature	✓	✓	✓			✓	

Figure 10: Attack Phase Cars

Attack Phase Cars

In addition to the Protected Unpaired-Fob binary, for each car, the attackers will have access to different resources:

- **Car 0: Unprotected Car:** The attacking team is provided with:
  - Plaintext car binary
  - Plaintext paired fob binary
  - Pairing PIN
- **Car 1: New Car:** The attacking team is provided with:
  - Protected car binary
  - Protected unpaired fob binary

  This car can be unlocked by compromising SR1 and doing so will release the new car unlock flag

Attack Phase Cars

- **Car 2: Temporary Fob Access:** The attacking team is provided with:
  - Protected car binary
  - Protected paired fob binary (Temporary)
  - Protected unpaired fob binary

  This car will only release a dummy flag during unlocking unless the paired fob device has been disabled using the secure MITRE bootloader. At that point, the car can be unlocked by compromising SR2 and doing so will release the temporary fob access flag.

Attack Phase Cars

- **Car 3: Passive Unlock:** To provision this car, the organizers will perform the following steps after building and protecting the binaries.
  1. Load the car and paired fob binaries on two keyed boards
  2. Record the UART communications during three unlock transactions between the paired fob and the car using a logic analyzer

  The attacking team is provided with:
  - Protected car binary
  - Recording of the UART communications during multiple unlock transaction

  This car can be unlocked by compromising SR3 and doing so will release the passive unlock flag.

Attack Phase Cars

- **Car 4: Leaked Pairing PIN:** The attacking team is provided with:
  - Protected car binary
  - Leaked pairing PIN

  This car can be unlocked by a compromise of SR4 and doing so will release the leaked pairing pin flag

Attack Phase Cars

- **Car 5: PIN Extraction:** The attacking team is provided with:
  - Protected car binary
  - Protected paired fob binary
  - Feature 1 packaged for Car 5
  - Feature 2 packaged for Cars 0-4

  This car will not release any flags from an unlock. However, two flags will be available to capture from this system. The first flag available is PIN Extraction, which proves the compromise of SR4. The other flag will be awarded for enabling features on the car. Feature 1 is already packaged for this car, so enabling Feature 1 will not reveal a flag. However, enabling Feature 2 will release a flag during an unlock, proving the compromise of SR5 and/or SR6.

Information with to Attackers

For each car discussed above, the additional files below will be made available to attacking teams.

- All source code
- The most recent design documentation provided to the eCTF organizers
- The built host tools for interacting with your design
- Car ID Numbers
- Feature Numbers

Time Requirements

OPERATION	MAXIMUM TIME FOR COMPLETION
Boot	1 second
Pair Fob	1 second
Package Feature	1 second
Enable Feature	1 second
Unlock Car	1 second

Size Requirements

COMPONENT	SIZE
Car Firmware	Max 110 KB
Car EEPROM Data	Max 1792 bytes
Fob Firmware	Max 110 KB
Fob EEPROM Data	Max 1792 bytes
Pairing PIN	6 hexadecimal digits
Feature Number	Can be 1, 2, and 3
Feature Message	Max 64 bytes
Unlock Message	Max 64 bytes
Car ID	8-bit unsigned integer

Memory Layout

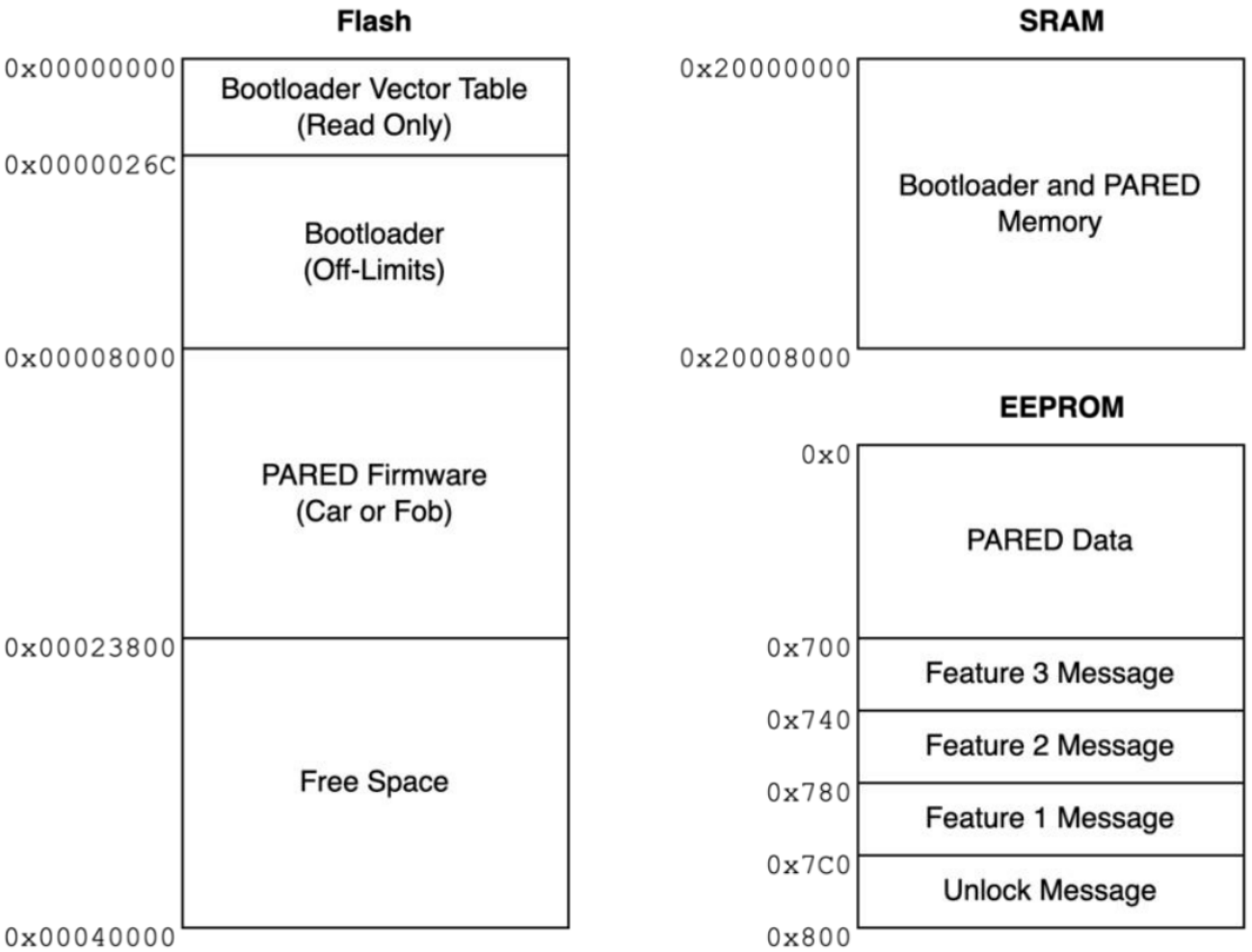


Figure 11: Memory Layout

Design Phase

- Environment

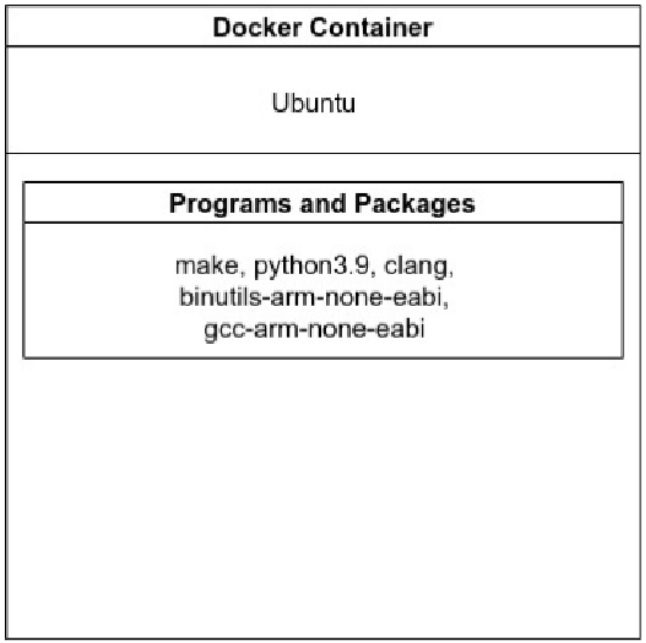


Figure 12: Docker container

- Deployment
  - system-wide secrets - can be a potential threat
  - car 0 binaries are unprotected and can be used to extract keys
  - unique key for each car and fob pair

Protocols

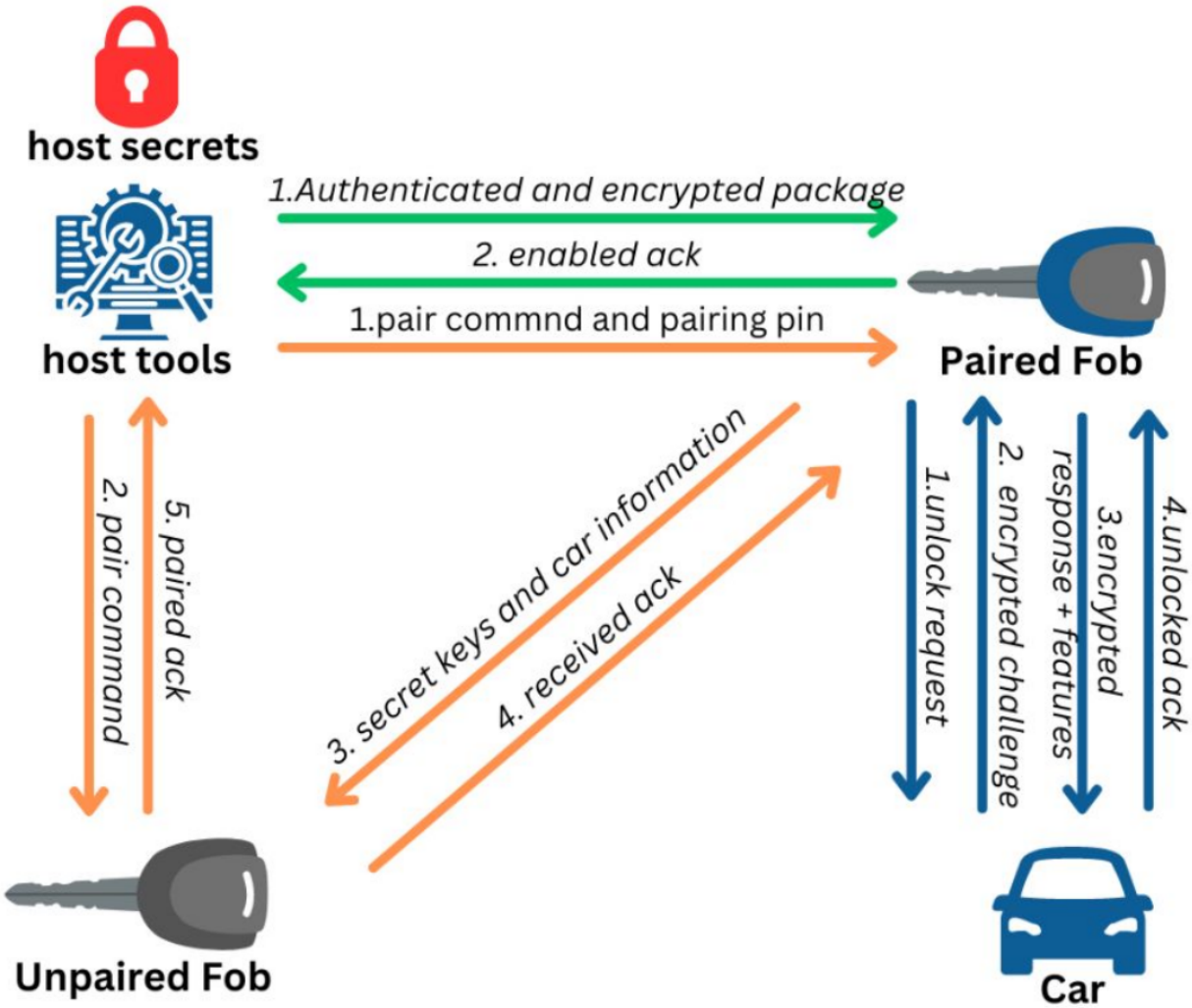
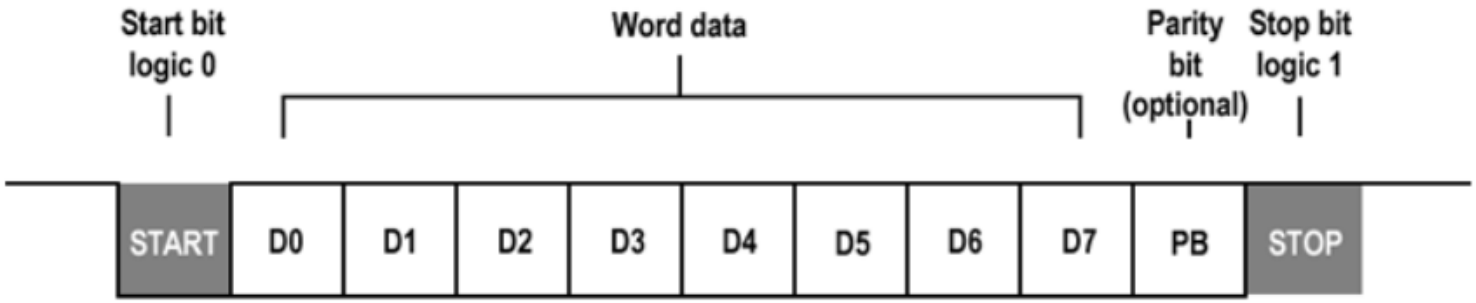


Figure 13: Overview of Protocols

Implementation - UART Communication



- Communication between paired fob-car, paired fob-unpaired fob, and user-fob is through the UART interface.
- Each communication session has a specified timeout to avoid deadlock situations, and the session is exited if the specified number of bytes is not received within the timeout.

Implementation - Challenges with UART

- Significant challenges were faced during UART because of frame errors.
- Following methods were tried but resulted in failure:
  - Decreasing number of data bits
  - Reducing baudrate
  - Increasing the number of stop bits
- Frame errors were caused by weak-pull up strength of GPIO pins
  - Attaching a weak internal pull-up resistor to the UART pins solved the problem
- This problem delayed our entrance to the attack phase by two weeks

Implementation - RNG

- Challenge-Response authentication used in the unlock process requires random number generation
- Absence of a hardware Random Number Generator (RNG): Major challenge
- RNG must have high entropy source for seed generation to avoid replay attacks
- 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.

Implementation - Encryption Scheme

- Entire design philosophy revolved around using symmetric encryption
  - memory and computation bottlenecks (TM4C123: ARM Cortex M4, 32kB SRAM & 256 kB Flash).
  - time requirements
- Encryption scheme must ensure confidentiality, integrity and authenticity of the message
- **NIST recommended ASCON-128 Lightweight Cipher** with AEAD(Authenticated Encryption with Associated Data) is used
  - Lightweight
  - High cryptanalytic security
  - interleaving for 32-bit systems
  - side channel countermeasures

Implementation - Generating Secrets

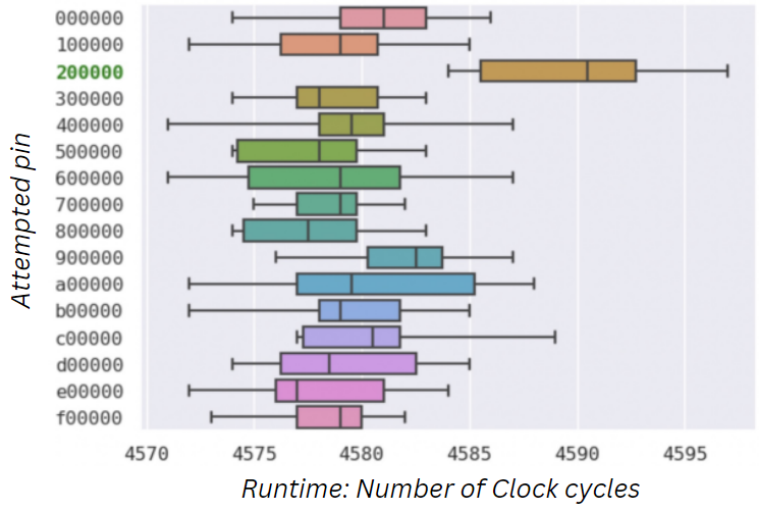
- Car
  - Generated 12 pairs of keys for the unlocking process and an additional package encryption key.
  - Keys stored in host secrets and copied to paired fob firmware.
  - Unlock keys stored in the EEPROM file of the car firmware.
- Fob
  - Retrieved keys from host secrets and modified EEPROM file.
  - Added car ID, car PIN, and package encryption key to the fob's EEPROM.
  - Filled EEPROM of unpaired fob with garbage data.

Attack Prevention: Stack

- Stack Execution Prevention
  - The SRAM is configured as read-write only, whereas it was previously configured as read-write-execute.
  - helps in preventing the execution of any potentially malicious code from the process stack.
- Buffer overflow attacks
  - The communication through UART is standardized with a fixed message size and format.
  - This helps prevent buffer overflow attacks and other types of input-based vulnerabilities.
- Zeroing of Stack
  - It is used to prevent sensitive data stored in the function's local variables from being left behind in memory where it could potentially be accessed by an attacker.
  - By overwriting the data with meaningless values before the function returns, the sensitive data is effectively erased from memory.

Attack Prevention: Timing Based

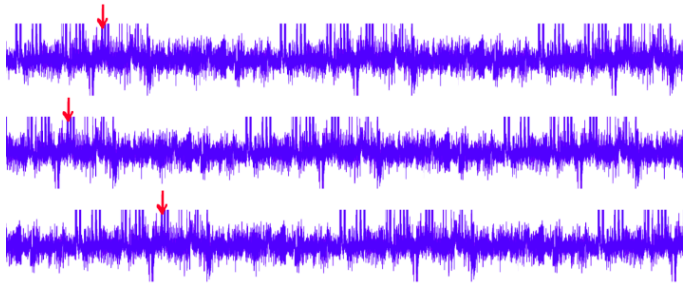
- Random Delays
  - The implementation of random delays makes it difficult to accurately measure time, thus preventing timing attacks.
- Constant-time Comparisons



  - Functions that 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 are vulnerable
  - By using constant-time comparisons for array, these attacks are avoided

Attack Prevention: Fault Injection & Side Channel

- Addition of Random Delays
  - Because of random delays, the target point in time is randomly changing ts position. Thus the attacker has to inject faults numerous times until the target & fault coincide

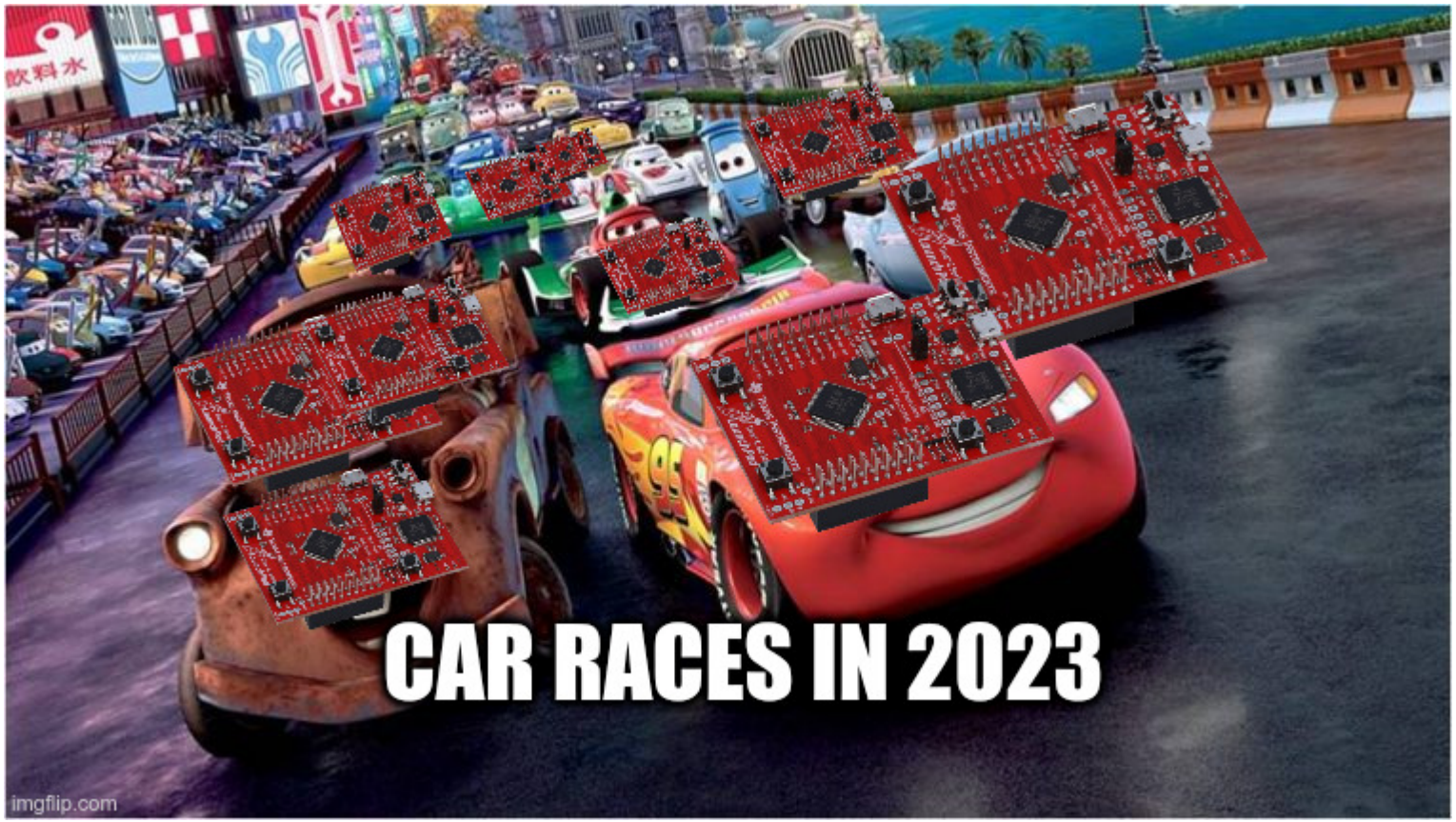


- Use of Internal Oscillator
  - The external oscillator on the evaluation board is prone to tampering to inject fault through clock glitching. Clock glitching is prevented using a precision internal oscillator, which cannot be tampered with externally
- Brownout Protection
  - Any brownout/tampering 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

Attack Prevention: Misc

- Brute Force
  - Adding intentional delays of 5 seconds in the implementation is a countermeasure to prevent attacks. Whenever the system is found to be under attack
  - These delays help in slowing down the attack
- EEPROM Block Hiding
  - Individual password protection is implemented for each block of the EEPROM containing sensitive data, such as keys and secrets, for both the car and the fob.
  - This ensures the confidentiality of sensitive information stored in it. Additionally, any modification to the EEPROM can only be made through the proper authentication process, further enhancing the security of the system.

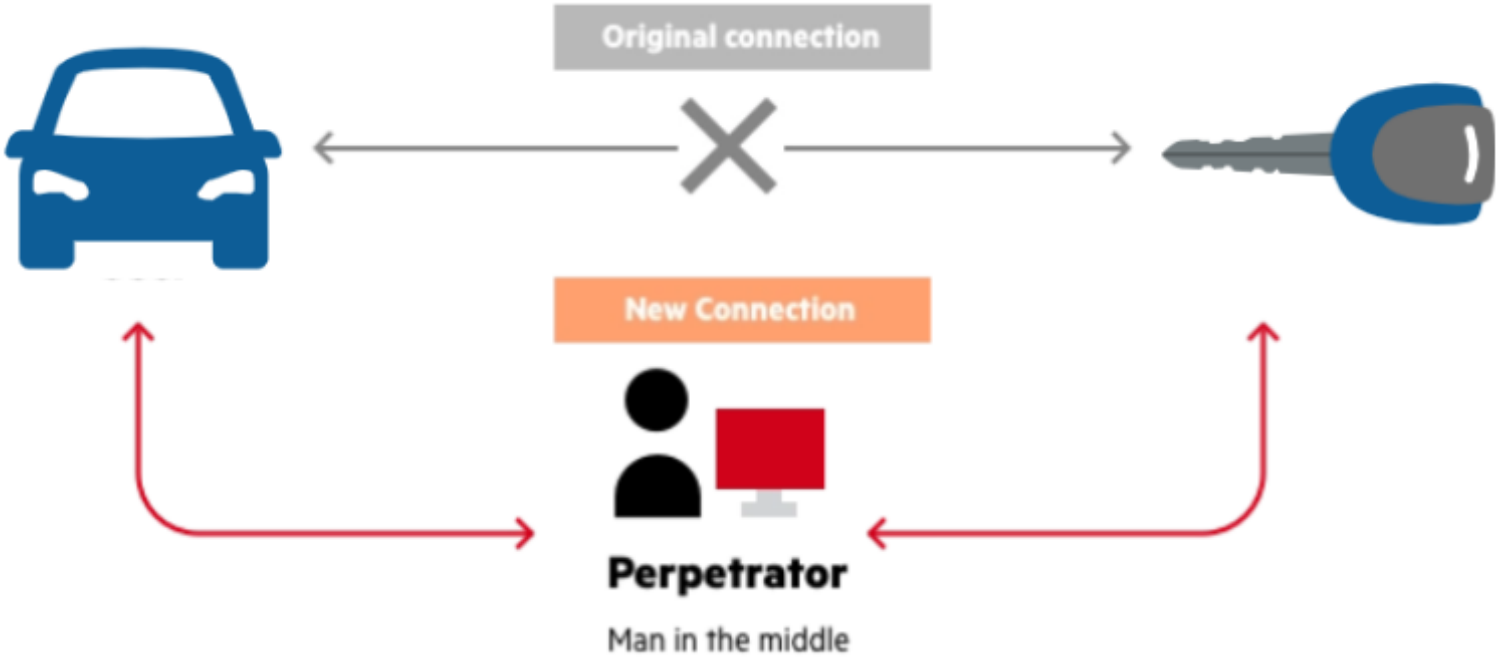
Attack Phase



Attack Phase

- A total of 80 teams participated in the competition out of which only 20 teams were able to move to the attack phase. The timing of their entry into this phase varied based on when they completed their design and qualified in the functional requirement round.
- To start attacking the other teams' design we require the attack board that was sent by the Mitre organization to each team. But due to customs issues, we received our boards after two weeks and that impacted our score majorly.

Man-in-the-middle Attacks



- Man-in-the-middle intercepts & maliciously modifies message packets during transmission
- Message packets must be checked for integrity and authenticity through MAC or authenticated encryption
- Exploited this vulnerability to enable features maliciously in `Car 5`

Man-in-the-middle Attacks

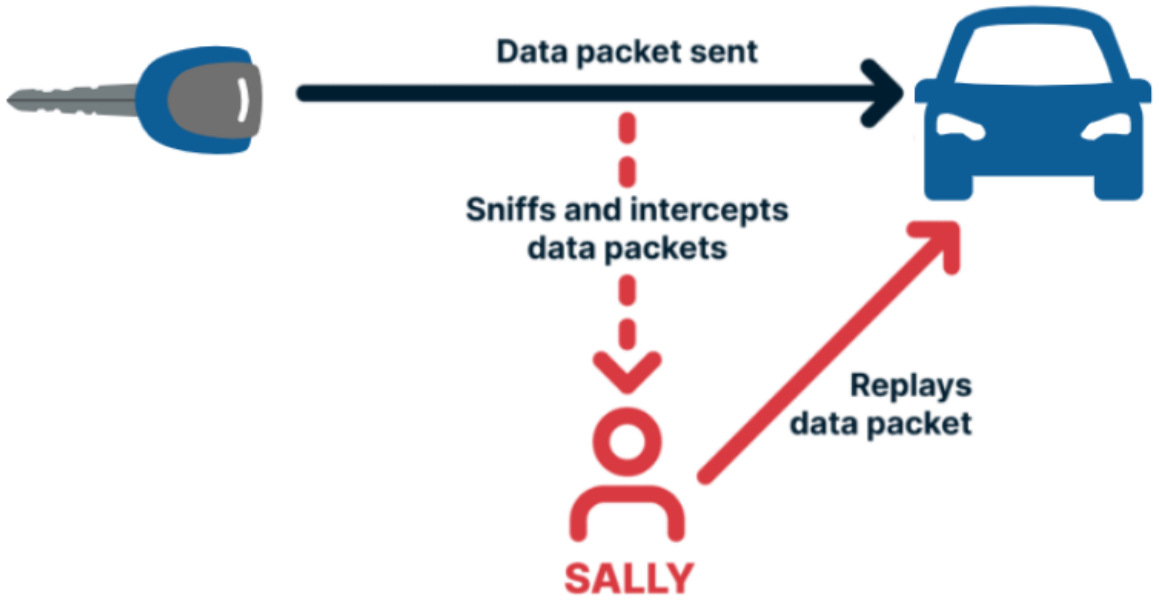
- Attacker's role: convince the receiver to accept a maliciously modified packet
- Message packets without integrity checks prone to tampering

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

- Certain designs had integrity checks; but no authenticity checks
- Attacker modifies packet, calculates the new hash, and convinces receiver to accept the malicious packet

```
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)
```

Replay attacks



- Designs involving Challenge-Response with poor entropy RNG prone to replay attacks.
- Exploited this vulnerability to obtain flags from `Car 2` and `Car 3`

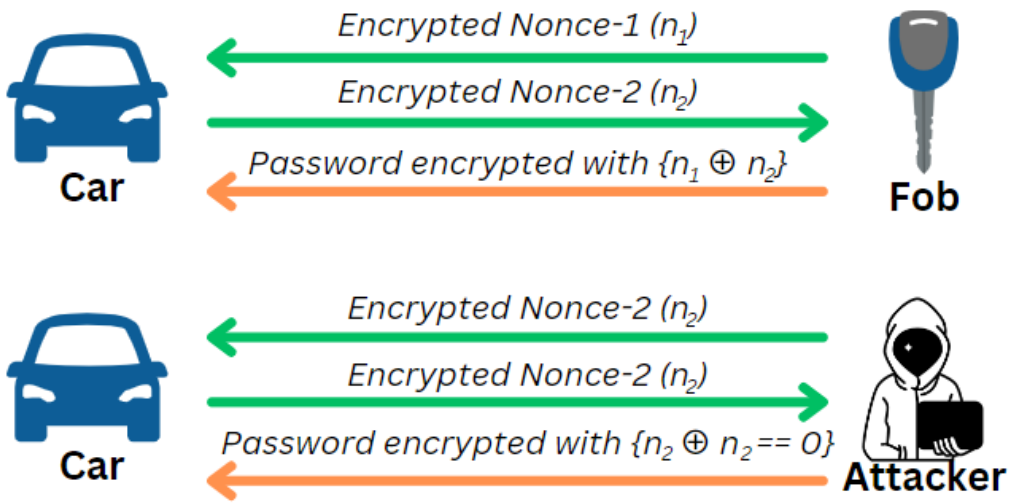
Replay attacks

Following vulnerabilities were found:

- Only Hardcoded PRNG seed was used
  - Compile time entropy insufficient
  - Same sequence of random numbers observed every time system is reset
- PRNG seed stored in EEPROM gets updated every time the system is reset/powered on
  - Attacker flashes image and captures communication traces
  - Once sufficient traces captured, Reflash the image and replay the captured traces
- Designs using system tick counter or interrupt-based timers instead of RNG
  - Attacker captures communication traces by starting the process precisely at a known time
  - Replay the traces & unlock the car by simulating exact conditions
  - Lowering resolution of timer increases difficulty

Replay attacks: Interesting Attack

- To prevent replay attacks, FAU 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.
- 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



Replay attacks: Interesting Attack

- Force the car to choose the same nonce as the fob sends, the attacker can cause the session key to 0
- Password message already known. Only the session key changes.
- Design vulnerable because nonces were not random
- Nonces were generated from seed stored in EEPROM which gets updated evertime system is reset.
  - System can be broken because same nonce is observed when firmware is reflashed

Brute Force Attacks

- Pairing PIN used in pairing process: 6-digit hex value - 24-bit entropy
  - Pairing pin: not strong enough to sustain brute-force attacks
  - Depending on implementation, average time to brute-force ranged from 2-10 hours
- Exploited this vulnerability to obtain pin extraction flags from `Car 5`
- Mounting the brute-force attack can be made impossible within the competition timeframe by ensuring fob enters `DEAD STATE` for 5-seconds if a wrong pin is entered
  - time requirements specified by the organizers: 1 second for the correct pairing pin and 5 seconds delay for the incorrect pin

Buffer Overflow Attacks

- Secure protocol doesn't matter if buffer-overflow vulnerabilities exist.
- Common vulnerable function that was used:

```
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;
}
```

Buffer Overflow Attacks

- Send more data than buffer size, overflow buffer
- Modify the return address to appropriate payload location
  - Can be redirected to skipping fob authentication in unlock process or into unauthenticated pairing of a fob
- Execution of plain-text firmware of `Car 0` was analyzed to design custom inputs for buffer specific to each design using `OpenOCD Debugger`
  - As there is no ASLR, same payload can be employed to unlock all cars.
- Used to extract flags from all 5 cars.
- We were able to break cars of UB and Tufts using buffer overflow.

Identical Encryption Secrets

- Identical encryption secrets or unintentional generation of the same keys were used by some teams, allowing exploitation.
- Keys were extracted from the unsecured binaries of Car0, and duplicate fobs were created for each car, allowing for unlocking.
- One team concatenated the car secret with the car ID and feature ID, encrypting the result using symmetric keys for packaging the feature. They used this same car secret to unlock the car, allowing for successful key replication.

Failures


study sidechannel analysis and learn how to use the ChipWhisperer for 2 weeks
Break AES in the demo lab and get a working trace capture setup just in the nick of time
idk how to use the attack package host tools
File "/home/iv/VT/host_tools/./unlock_tool", line 25, in unlock
car_sock.connect(("ectf-net", int(car_bridge)))
socket.gaierror: [Errno -3] Temporary failure in name resolution

Attacks to be implemented:

1. Timing attacks on non-constant-time array comparisons utilizing `strcmp` and `memcmp` functions
   - Unsuccessful because of coarse measurement of time
   - Because of no-precise pivot point from which the execution time can be measured. Ending of UART communication session was used as a pivot point which introduced high-variance errors in the timing measurement
2. Due to time constraints we were not able to implement Side Channel and Fault injection attacks

Design Improvements: RNG

- MITRE-provided secure bootloader zeroes the SRAM entirely on startup before loading the image - went unnoticed
  - initial values for SRAM at startup cannot be used as a source of entropy
  - Resulted in low-entropy RNG
  - One of our competitors used this vulnerability to retrieve flag from `Car 2`
- Obtain seed for PRNG by hashing other high-entropy sources:
  - De-biased noise for internal temperature sensors of the ADC(Analog-to-Digital Converters) & CPU
  - Clock drift between Hibernation Clock and Clock obtained from precision internal oscillator

Design Improvements: Encrypted pairing process

- In current design, during pairing process, paired fob openly shares the secrets to the unpaired fob without ensuring confidentiality
  - Cannot be used in real-life since replay attacks could result in unauthenticated pairing.
  - If pairing pin is known, attacker can obtain all cryptographic secrets for all the processes - **Single Point of Failure**
- Encrypt the transfer of pairing information to the unpaired fob after PIN verification
- Symmetric encryption using fixed keys is vulnerable
  - Attacker can obtain the key from `fob 0` of `Car 0`
- Encryption keys must be created on the fly for secure communication for each session using authenticated secure key exchange protocol
  - Private keys are not known to other parties, thus confidentiality can be ensured

Other Design Improvements

- **Secure Handling of Pairing Process:**
  - Store the pairing PIN on the boards after hashing to protect its confidentiality.
  - Ensure that even if the pairing PIN is correct, the execution time for PIN verification is consistent (e.g., close to 1 second) to account for brute-force attacks by resetting the attempts.
- **Randomization of Key Addresses:**
  - Randomize the addresses of various keys stored in the EEPROM during the build process.
  - Store the mapping of randomized addresses in a secure secret header, preventing predictable access to the keys.
- **Feature Enabling: Verification at car**
  - In current design, packaged feature is authenticated and enabled in the fob.
  - If the paired fob is compromised, all the features can be enabled.
  - Thus the enabled feature must also be verified both by car and fob.

Results

#	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

Figure 14: Final Standings

- We stood 5^{th} position overall in the eCTF-2023 competition scoring 11343 points.
- Snatched 56 flags.
- We had the 4^{th} highest ranking in the attack phase

Results

Teams	New Car Unlock	Temporary Fob Access	Passive Unlock	Leaked Pairing PIN	PIN Extract	Enable Feature
TAMU	46	58	54	47	938	61
VT	198	59	74	198	52	52
UB	57	57	67	67	90	21
SMU	1010	199	1010	1010	93	1010
MorganState	100	167	100	100	167	100
UCI	52	65	57	52	72	57
UMass	1032	78	65	1032	1032	1032
WPI	1028	84	84	1028	1028	1028
MSU	1418	69	1418	1418	86	1418
UCCS1	52	56	48	52	159	65
IITM	1082	243	1082	1082	1082	1082
UNHaven	94	81	94	94	762	81
Purdue	120	229	120	120	112	403
UCSC	1394	1394	1394	1394	1394	1394
Washington	94	64	66	97	173	370
CMU	1552	1552	1552	1552	1552	1552
UCCS2	65	65	65	65	71	66
FAU	68	59	70	74	63	62
Tufts	256	82	256	256	71	594
UIUC	102	121	112	112	1394	1394

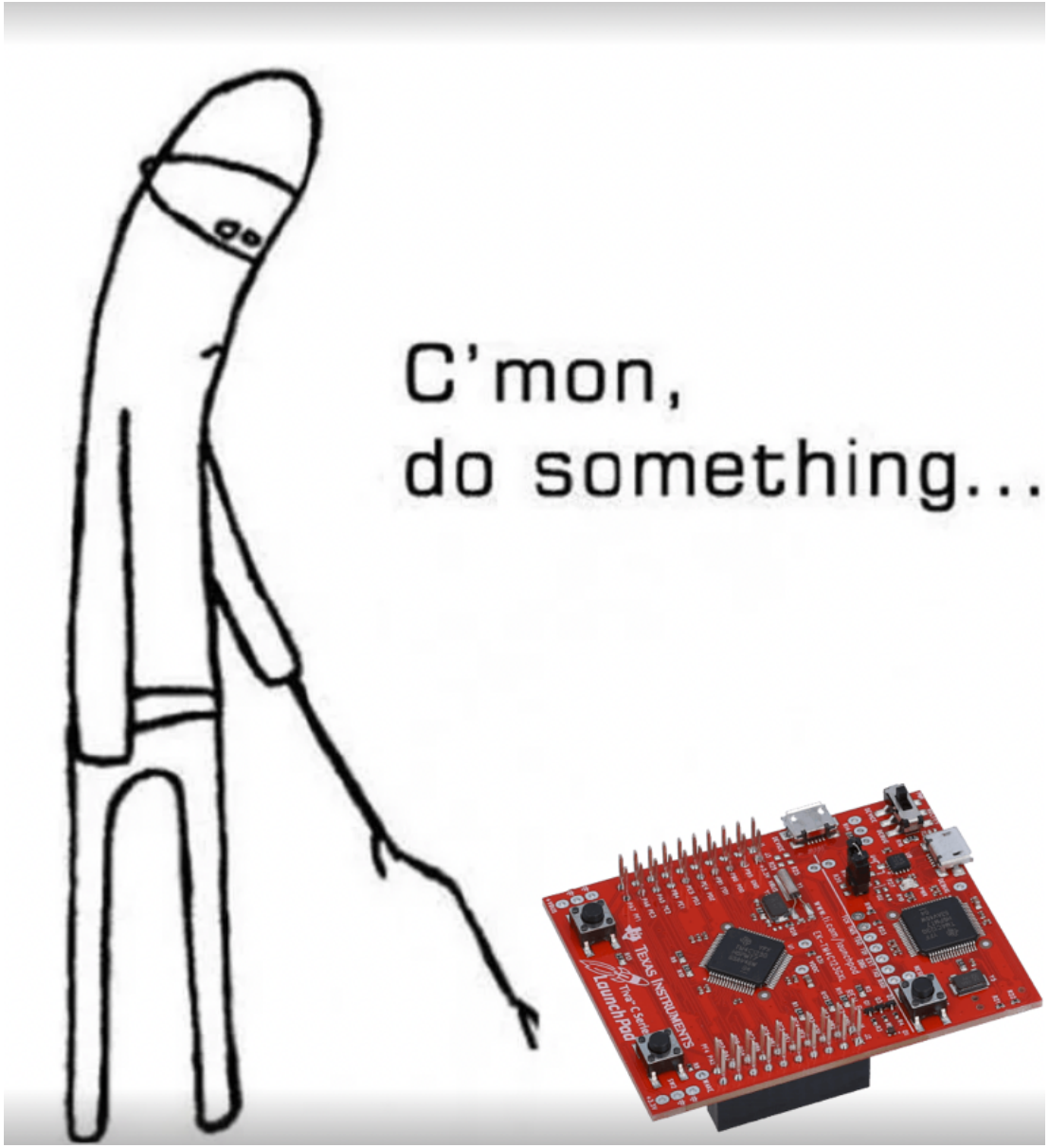
Legend

Dark Blue Unsolved by any team
Blue Solved by one team
Light Blue Solved by multiple teams
Green Solved by your team
Gray Unavailable
Red Locked

Results

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.

Individual Contributions



Individual Contributions

Divya Prajapati

- During the design of the protocol:
  - Assisted in the conceptual development of the protocols
  - came up with a suggestion of using multiple key pairs for encryption and randomly selecting one for each unlock process
  - discovered a functional bug in the insecure example provided by the organizers and provided modification to solve it
- During the implementation:
  - worked on understanding the flow of the build process
  - Responsibilities included understanding and implementing a unique key generation process for each car and fob pair
  - Implemented the host tools
  - Integrated the developed modules.
- During the attack phase:
  - worked on identifying the vulnerabilities in the protocol, implementation, and RNG.
  - Implemented the stack-smashing attack described in the attack phase
  - mounted some of the brute-force attacks on the pairing pin.

Individual Contributions

Divya Prajapati

- During the attack phase:
  - worked on identifying the vulnerabilities in the protocol, implementation, and RNG.
  - Implemented the stack-smashing attack described in the attack phase
  - mounted some of the brute-force attacks on the pairing pin.

Individual Contributions

Antonson J

- Throughout the **design phase**, my contributions included:
  - Contributed to the development of the secure protocol.
  - Assisted in the development of various functional blocks.
  - Conducted multiple code reviews to check for potential security vulnerabilities in the final design.
- During the **attack phase**, my responsibilities included:
  - Actively involved in identifying vulnerabilities.
  - Implemented a MATLAB UART parser to parse the given CSV files (for car3) to give out the data in a human interpret-able format.
  - Created Python scripts to be used by the host machine to exploit teams vulnerable to man-in-the-middle attacks.
  - Assisted in implementing automatic replay scripts for collecting traces, re-flashing firmware to reset the boot-loader, and replaying the captured traces for replay attacks.
  - Assisted in implementing brute force attacks, and exploiting buffer-overflow vulnerabilities

Individual Contributions

Priyanka

- Design Phase
  - Contributed to coming up with the ideas and overcoming challenges faced in the design of the system.
  - Integrating the developed modules such as the unlocking process and the pairing process to ensure that data exchange and other checks are working successfully.
  - Developed functions to read and write the required data from the EEPROM.
  - Contributed to defining the memory mapping considering the data size while generating the EEPROM file.
- Attack Phase
  - Thoroughly reviewing the designs of other teams in order to identify potential vulnerabilities, such as the use of the same keys to unlock other vehicles through car_0, the risk of replay attacks due to weak RNG, and buffer overflow issues.
  - Worked on a few replay attacks & man-in-the-middle attacks by collecting the data, finding the patterns in the data collected, then using it to break the car.

Individual Contributions

Shree Vishnu

Design Phase

- Assisted in the development of the secure protocol
- Reviewed existing literature on generating random numbers in micro-controllers using SRAM startup values
- Removed code execution in SRAM segment to prevent stack smashing
- Implemented random number generation
- Added further protection by zeroing local variables in SRAM after usage

Attack Phase

- Studied basics of RUST language for attacking some of the designs
- Identified buffer-overflow, replay attack, and timing attack vulnerabilities of several rust-based and C-based implementations
- Created a local channel to capture the communication

Individual Contributions

Arun Krishna
Design Phase

- Contributed to the development of the conceptual design of the secure protocol.
- Implemented UART Communication, System tick functionality, and timeout functionality for the UART communication.
- Developed individual modules for unlocking & pairing processes, ensuring stack-protection measures.
- Implemented measures against fault injection: Random delay, Use of internal oscillator, and Brownout protection.
- Assisted in implementing EEPROM Password protection to prevent unauthorized access.
- Conducted multiple code reviews and audits to screen for security vulnerabilities.

Indiviual Contributions

Arun Krishna 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

Thank You!

Heartful thanks to everyone who has been a part of this journey!