A Additional Ethic and Compliance Information

Adhering to ethical standards and despite receiving an exemption from our IRBs, our study strictly followed the designated policies and procedures applicable to human research as specified by our institutions and study protocol. Prior to conducting the interviews, we ensured all participants fully understood their involvement through a clear consent process and upheld their right to withdraw from the study at any point without repercussion. To safeguard participant privacy and confidentiality, all PII was anonymized during the transcription of interviews. Additionally, participants were afforded the freedom to withhold any information they preferred not to disclose during all stages of data collection.

For transcription of interview recordings, we used Otter AI [45], an online service that converts audio to text. We promptly contacted all participants to disclose this information and clarified that Otter AI's privacy policy [46] complies with our IRB protocol. No manual review of transcripts or audio was conducted by Otter AI staff, and no PII was shared with the service. Additionally, according to Otter AI's privacy policy, data used for model training is further de-identified to ensure individual users cannot be identified.

We shared our findings with the MITRE eCTF organizers, who subsequently communicated them to vendors.

B The General Process of eCTF

Figure 4 illustrates the general process of eCTF. The *handoff* phase typically begins 1.5 months after the *design* phase starts. After successfully completing function tests in the *handoff* phase, a team enters the *attack* phase immediately. The competition ends approximately 1.5 months after the *handoff* phase begins.

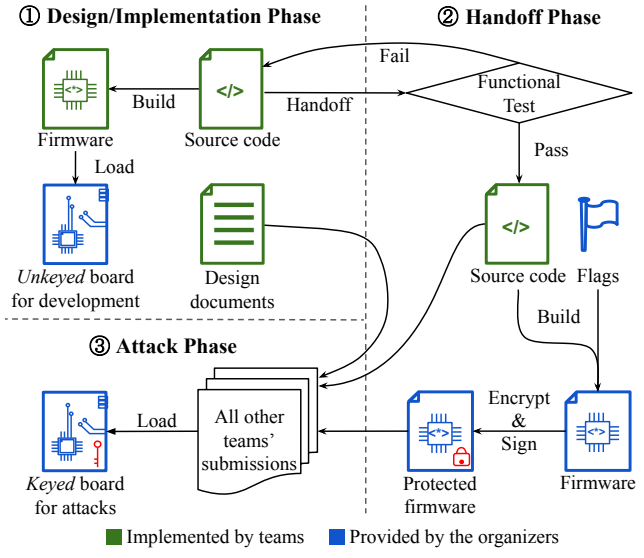

Figure 4: The general process of eCTF.

C Participant Background Survey

Interviewees were linked to their teams during the initial email exchange. The survey was designed to gather demographic and team-related information of the participants.

C.1 Personal background questions

(1) Which year of the MITRE eCTF did you participate?
(2) What was your degree type during the eCTF? For example, high school, undergraduate, Master's, Ph.D., or other.
(3) What was your program of study (major)?
(4) Does your institution offer embedded- or IoT-specific security courses? If so, did you take those courses before the competition? Have you taken them since the competition?
(5) Does your institution offer other security courses? If so, did you take those courses before the competition? Have you taken them since the competition?
(6) Have you previously participated in the eCTF competition? If so, when?
(7) What is your experience with other security competitions?
(8) What was your design or implementation experience of the embedded systems before the eCTF?

C.2 Team-related questions

(1) How many active members were on your team? Please only include members who regularly contributed during the competition.
(2) How many students expressed interest in competing but did not remain active until the end of the competition? If so, what are some of the reasons that they became less involved?
(3) How much was your teams' faculty advisor involved in the eCTF? (give some hints)
(4) Did your institution offer class credit for participation in this competition? If yes, how many credits?
(5) What was your role and contribution in the team?
(6) What were the degree-level and program of study for the other participants on the team?
(7) What was the general level of experience among your team members?

D Interview Questions

Prior to each interview, we ensured that interviewees read and understood the consent information sheet and were fully informed about the study's scope, their rights as interviewees, and our confidentiality measures. At the beginning of each session, verbal consent was recorded to confirm their willingness to participate.

D.1 Questions related to security practices

(1) *The principle of least privilege is a computer security practice that gives tasks least access rights based on their job. The Microcontroller Unit (MCU) used in the competition supports two privilege levels. However, our analysis found that none of the competing teams utilized this privilege separation feature.*
   (a) Did your team know that this MCU offers two privilege levels?
   (b) Why didn't your team use both privilege levels of the MCU?

(c) Do you think using both privilege levels would have improved the security of your design?

(2) *The compiler and linker provide multiple security enforcement options that can be enabled through editing the Makefile or linker script.*
   (a) Do you know the benefits of making the stack non-executable?
   (b) Do you know there are compiler flags to make the stack non-executable, and you can also mark the SRAM region as non-executable in the linker script?
   (c) Do you know how these compiler flags or linker script attributes work under the hood?
   (d) Did you know the memory protection unit (MPU)? How did you know it?

(3) *Stack canaries are a security feature that detects stack buffer overflows by placing a random value before the return address on the stack. When the function returns, the canary value is checked to see if it has been modified.*
   (a) Did you know the stack canary feature before the competition?
   (b) Why didn't your team use the stack canary feature?

(4) *Using the C standard library on embedded systems can be tricky due to limited resources and specific requirements.*
   (a) Before the competition, did you aware that some of the C standard library functions perform differently on the embedded system compared to a general purpose system?

(5) (Some teams) *We found in your team's design, the non-executable stack and* ***relro*** *compiler flags were added in the Makefile.*
   (a) What was the intention in using the flags?
   (b) Were these flags effective at meeting the intentions?

(6) (Some teams) *We found in your team's design, the SRAM attribute in the linker script was changed to read/write only.*
   (a) What was the intention in changing this attribute?
   (b) Were these changes effective at meeting the intentions?

D.2 Questions related to memory wiping

(1) *Sensitive data, such as encryption keys, can be stored in some secure storage at rest, and then be loaded to the main memory for computation. To minimize the time window for sensitive data residing in the main memory, one can clear out the data in the main memory after use. This protection is called memory wiping.*
   (a) Did your team use or attempt to use the memory wiping as a defense technique?
      (i) If so,
         - how did your team implement it?
         - what was your intention in using it?
         - how did you confirm that your use met the intentions?
         - (some teams) we found in your team's design, some of the `memset` function calls were optimized out by the compiler in the resulting assembly. How do you think of this?
      (ii) If not, were you aware that these techniques existed before the competition?
      (iii) Did you know that the compiler may optimize away some code, which in the compiler's view, has no effect on the later code?
      (iv) Did you know that the compiler may treat some of the security-related code as unnecessary and optimize it away?

(2) (Some teams) *We found in your team's design, you implemented your own inline wiping function instead of using those provided by the library.*
   (a) Why did your team choose to do this?
   (b) Do you know your inline function to erase the memory is translated into `memset` in the resulting assembly?
   (c) Did you intend to do this, or you didn't notice what happened during the compilation?

(3) (Some teams) *We found in your team's design, you used the memory wiping functions provided by the crypto libraries.*
   (a) Why did your team choose to use the wiping functions provided by the crypto libraries, instead of using the libc standard functions such as `memset`?
   (b) Did you know that libc's wiping functions could be optimized?

D.3 Questions related to memory-safe programming

(1) *Embedded systems are usually developed in memory unsafe low-level languages such as C, this is also the language that the reference design used. A memory-safe language, such as Rust, aims to significantly reduce memory corruption vulnerabilities in a program while keeping the performance high if used in a proper way.*
   (a) What were the factors you consider when you choose which language to use?
   (b) (Teams used C/C++) *We found your team did not choose to use a memory-safe language, such as Rust.*
      (i) What stopped you from using a memory-safe language?
      (ii) Would you consider using it in the future? Why or why not?
      (iii) Do you believe that utilizing a memory-safe language like Rust can eliminate memory vulnerabilities in embedded systems?
      - *(skip the rest of questions in this section)*
   (c) (Teams used Rust) *We found your team used the Rust programming language in the design.*
      (i) What was your level of experience in using the Rust programming language?
      (ii) What were the challenges when using Rust on this microcontroller-based system?
      (iii) Would you use it again if you still participate in the future? Why and why not?

(2) (Some teams) *We found in your team's design, you mixed C and Rust for the implementation.*
   (a) What motivated your Rust/C hybrid design?

(3) *The Rust programming language also provides the unsafe code block which allows developers to violate the memory safety rules. Abusing the unsafe code block could void the memory safety feature of Rust. But sometimes it is necessary to use it for embedded system development.*
   (a) When or where did you feel you have to use unsafe blocks?

(b) How did you approach the use of unsafe blocks in your Rust code? Did you have any security concerns when using them, and did you try to minimize their usage?
(c) What approaches did you use to minimize the number of lines of code in unsafe blocks?
(4) Did you or your team run into any significant roadblocks in using Rust, such as excessively large binaries, that you had to work around? Did you have to make any compromises in the process?

D.4 Questions related to entropy source

(1) *In cryptography, a Random Number Generator (RNG) is essential for creating secure cryptographic keys, generating nonces, and ensuring randomness in various security protocols.*
(a) Did your team use the random number generator in the design?
(i) If so,
- did your team implemented your own RNG, or just used one that provided by the library/SDK? Why?
- how confident were you in your team's RNG implementation?
- did you have any concerns about the robustness of the vendor-provided RNG?
- what entropy sources did your team choose to seed the RNG? Why?
- how confident were you in your entropy design?
- how did your team test the effectiveness of the RNG design?

(ii) If not, why didn't your team consider including randomness in the design?
(b) (2023) Did you know the board had an internal temperature sensor that could serve as an entropy source? How did you know it?
(c) (2024) Did you know the board had a True Random Number Generator (TRNG)? How did you know it?

D.5 Questions related to threat model and embedded-prone attacks

(1) What is the threat model of your team's design?
(2) What are the particular types of attack that you considered to address in your design?
(3) For each of the attacks that was not mentioned in the previous answer, e.g., brute-force, MITM, cold boot attack (to dump the memory), side-channel and fault injection, ask them the following questions:
(a) What is your understanding of this attack?
(b) Did you consider this attack in your threat model? Why not?
(c) How did your design mitigate this attack? How effective was that defense?
(d) How did the embedded nature of the system influence your approach?
(4) In terms of attack surface, do you think the embedded system has a larger or smaller attack surface compared to a desktop system? Why?
(5) Regarding the software vulnerabilities and potential hardware attacks, which one would you give more priority to defend against? Why?
(6) *Questions related to comparing the user-controlled input with secret values.*
(a) When checking PIN value, which method or function did your team use to compare two values?
(b) Why did you choose this method/function?
(c) Did you know how the strcmp/memcmp function works? If so, how? (stop at the first difference)
(d) What issues do you think exist in these functions?
(7) (Some teams) *We found in your team's design, random delays were added between some operations.*
(a) What was the purpose of adding these random delays?
(b) Where do you think are the best locations of adding these delays?

D.6 Questions related to vulnerability discovery

(1) *There are many ways to find vulnerabilities in a system, such as checking the source code, analyzing the disassembly code of the compiled firmware, dynamically debugging the programs during the runtime, fuzzing, and so forth.*
(a) How did your team evaluate the correctness and security of your own design?
(b) How did your team do debugging when implementing your design on the board?
(c) How did your team find vulnerabilities in other teams' design during attack phase?
(d) Did your team use any static or dynamic techniques to detect errors?

D.7 Ending questions

(1) Are there any other notable practical challenges that you encountered during the competition?
(2) Is there anything else you would like to share with us?

E Codebook

The codebook contains theme names, theme descriptions, and associated sub-themes, as detailed in Table 1.

Table 1: Codebook.

Theme Name	Theme Description	Sub-theme
Knowledge of security mechanisms	Statements that describe the awareness and understanding of general security knowledge.	* Participants were not aware of privilege separation. * Those that knew about privilege separation were unaware that it is offered by Cortex-M. * Other participants were not confident that privilege separation would improve the security of the system. * They believed that since there was no OS, privilege separation was not needed. * Participants were unaware of memory wiping as a defense technique. * Some were aware that the compiler optimizes unnecessary code, but none were aware that memory settings would be optimized. * Participants were unaware that the stack canary, or unaware of how canary works on a microcontroller system. * They thought the stack canary needed to be coded manually.
Knowledge of tools/features	Statements that describe the understanding embedded system development tools/features.	* Participants were not aware that the NX compiler flag did not make memory non-executable on the microcontroller. * Participants were unaware that they could use the MPU to set regions of memory as non-executable. * Participants were not aware that some C standard library functions performed differently on the embedded system compared to a general-purpose system (e.g., time(), printf()).
Attack and defense	Statements that describe the awareness of embedded-prone attacks and how participants defense them.	* Participants were not always aware of embedded-prone attacks that they must implement defenses for. * Even if participants were aware of embedded-prone attacks, they are not always aware of how to defend against them. * Even if participants were aware of embedded-prone attacks and they tried to implement a defense, their defense was conceptually flawed.
Embedded memory	Statements that describe challenges of using embedded system memory.	* Participants mentioned that memory bugs are more severe because everything (Flash, SRAM, peripherals) is accessible in a single memory space. * Participants mentioned that storing sensitive data is more difficult on microcontroller systems. * Participants mentioned that the microcontroller systems' lack of dynamic memory management and OS poses security challenges and may downgrade the security design. * Participants mentioned that the limited flash and SRAM space in statically linked microcontroller systems makes it harder to use third-party libraries. * Participants mentioned that testing and debugging are harder on a microcontroller than on microprocessor systems.
Embedded Rust support	Statements that describe participants' challenges of using Rust for embedded system development.	* Participants thought Rust was useful because it eliminates virtually all memory safety bugs. * Participants mentioned having to remove the Rust standard library because there is no operating system support on the embedded system (e.g., memory allocation). * Participants mentioned some libraries not running because the Rust standard library wouldn't work on the embedded system. * Participants mentioned having no direct vendor support for getting Rust to compile to the board. * Participants mentioned encountering excessive stack memory consumption because of how the Rust compiler constructs objects. Because embedded systems are resource-constrained, using excessive stack memory will eventually cause issues. * Participants mentioned having to write a HAL, perhaps using unsafe blocks, in order to let Rust access board hardware.
Experience with Rust	Statements that describe participants' understanding of the Rust language.	* Participants thought Rust was useful because it eliminates virtually all memory safety bugs. * Participants who didn't use Rust cited lack of experience/programming knowledge as a reason. * Despite their lack of knowledge about the language, participants believed that Rust eliminates memory bugs.
Security-related compiler flags	Statements that describe the knowledge of security-related compiler flags.	* Participants did not notice any warnings from the compiler when security-related flags have no effect. * Participants tried to use NX or tried to modify the linker script, but none of them knew that these were not effective. * Some participants were aware of security concepts like NX but were not sure how to apply them to their design. * Participants did not know that because the ELF header is removed when the firmware is flashed to the board, any bits that are set in the header are always discarded.
Lack of entropy source	Statements describe the awareness and knowledge of entropy source of embedded systems and how participants get random numbers.	* Participants were aware that an entropy source is needed to seed the random number generator for cryptographic operations. * Participants may be unaware that general-purpose systems get their entropy from system time, cursor location, sources of noise, etc. * Participants were not aware of the existence of the CPU temperature sensor, which was a good source of entropy. * Participants who were aware of the CPU temperature sensor all used it as the entropy source. * Participants mentioned the use of mixed entropy sources. Build time entropy supplied by host, run time entropy from systick timer, cycle count, uninitialized SRAM. * Participants were aware that the board vendor provided an API for a true random number generator. * Participants mentioned only using one secure of entropy, such as build-time entropy, or no entropy. Some teams used rand() and time(), which doesn't work on microcontroller. * Participants had challenges in testing their RNG design on the microcontroller. * Participants were unaware that the same library, e.g., the libc standard library time() function, might work differently in a microprocessor system and a microcontroller, and no warning was given to the user.