WildCerts

University of New Hampshire

Joshua Calzadillas
Advised by: Professor Qiaoyan Yu, PhD

Design Overview

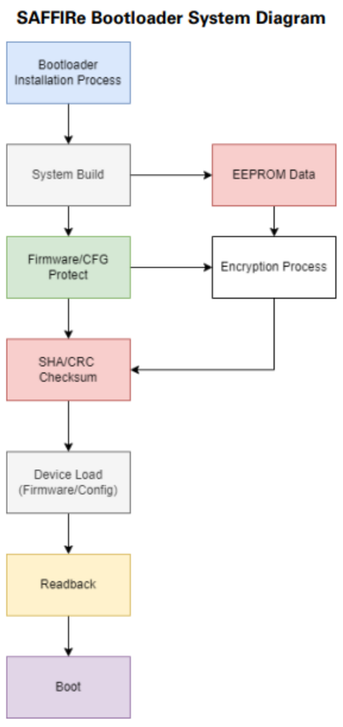
This section could include the following:

1. The major design setup was integrating an encryption setup to encode the values of one firmware to another based on the booting process.
2. The software features included instruction-based encryption (polymorphic encryption), CRC32 checksums, Blowfish encryption, AES encryption, and a memory management system.
3. However, the instruction-based encryption was removed due to time constraints.

Offensive Highlight

This section should include:

1. Most of the attack-phase attacks were not done due to the fact that we didn't meet the attack phase during this eCTF.
2. That being said, the team has learned an intangible amount of experience and information about securing an embedded system and it's contents form all sorts of different attacks.
3. This CTF also allowed the team to learn how to use debuggers more effectively, how to reverse engineer an executable or a piece of firmware using Ghidra and strings.
4. We also enjoyed the SCA and the RE segments of the CTF as well and the Boston Cybernetics Institute training that we received earlier in the competition.
5. Overall, this CTF was one of the team's favorite CTF's that we have ever participated in.
6. Thank you for sponsoring this!



Defensive Highlight

This section should include:

1. The countermeasures that were implemented in my design were CRC32 checksums and a separate (custom) CRC32 based hashing algorithm which took the inputs of several CRC32 values and outputted a separate hash values for that set of values.
2. The other defense features were encryption and memory management. The encryption was used to make sure that the contents of the device were somewhat secure (not easily visible to be seen without some hard work). The memory management allowed the software to detect when certain parts of memory were being accessed or tampered with without the software giving consent. Which made the system even more secure by making it harder for the user to just substitute values as they please.
3. The system works however, there wasn't enough time for the team to got the attack phase due to time constraints with other classes as well. However, there are enough resources used to apply this to other projects as well. (The team had to scrap a lot of work and try to ad-hoc our way through to get into the attack phase.)

4. How you could build upon this feature in the future to create an even more secure design Probably should put these 4 things in the template so that it's clear how we want them to be identified
    1. The first thing I would do is implement the instruction-based encryption system to make the system even more secure.
    2. I would take out a lot of the fluff-based hashes and only do a few CRC32 checksum (only what is applicable) to save on memory resources.
    3. I would also have several boot codes at which the bootloader would know when and how to boot (mostly hardware implemented codes) that can be managed by the device.
    4. I would also add noise to some of the functions to make it harder to do SCA due to random noise constraints being applied to remove some of the ease of accessibility and guess work for a side channel analysis to occur.

References

1. Professor Yu
2. Joshua Calzadillas