MITRE eCTF 2017

UConn Firmware Dogs Write-ups

Brian Marquis, Patrick Dunham, Luke Malinowski, Cameron Morris, James Steel, and Chenglu Jin
Advisor: Dr. John Chandy
4/18/17

Contents

1 Attack - Non Flag

1.1 Summary

The AVR architecture only allows the application flash section to be written from the bootloader flash section. However, if the application flash section is able to read the bootloader section due to a certain lock bit configuration, this attack can be used to modify the bootloader code in memory. The basic concept is to call a function from the bootloader section of memory to write to a location in flash from the application flash section. This can be used to modify the bootloader code in memory.

1.2 Requirements

- Must be done after malicious firmware has been installed
- The lock bits must be set appropriately: BLB11 and BLB12 must be set.
- Affected Teams: *RPISEC*, *Sprite*, *Northeastern is a Trade School*, *pgm_read_flag*

1.3 Description

This attack exploits a vulnerability in the AVR architecture in that, if the attacker knows the bootloader code, he/she can use a jump or call instruction to that location in memory to execute SPM instructions as though it were the bootloader. Depending on the compiler optimization, the page writing code may be able to return to the original call address in the application flash section. Otherwise, a Timer interrupt can be used to jump back to the application flash section.

To do this, an attacker must first be able to upload arbitrary malicious firmware. This has been demonstrated to be possible on many of the systems in this competition by using a clock glitching attack to skip the key initialization. The malicious firmware will need to read out the bootloader section of flash memory so that the attacker can find the instructions of interest. Once this is done, the attacker now knows the location of the SPM instruction in the bootloader section. Next, that firmware image can call this address to write to any location in memory, even the bootloader.

1.4 Impact

- Allows attackers to change bootloader:
  - Add backdoors
  - Disable security checks
  - Change the key
- The attack is inconspicuous; the system may not appear to be compromised.
- Attempts at reconfiguring can be spoofed. That is, the bootloader can make the user think it was reconfigured, but actually do nothing.

1.5 Scale

- Malicious firmware routine should be able to find the bootloader addresses of interest on all AVR bootloaders

1.6 Feasibility

- It has been proven that arbitrary firmware can be installed using clock glitching with relative ease.
- Has been used in Fignition, an open source project.

2 Defense

2.1 Threats

Threats Considered:

- Clock Glitching
- Brown-out attacks
- Timing Side Channel Attacks
- Power Side Channel Attacks
- Integrity/Authentication
- Decapping

When designing the bootloader, clock glitching and brown-out attackers were thought to be the most likely in this competition. Clock glitching can be catastrophic as it can allow an attacker to skip instructions at their discretion. Brown-out attacks are threatening, but less dangerous than clock glitching because skips are less consistent and do not work on all types of instructions. Side-channel attacks, while effective, did not seem particularly viable in this competition due to the limited time span and precision equipment required to do so. However, such attacks could be used to break the bootloader's encryption. Finally, integrity and authentication were considered when designing the bootloader to prevent attackers from installing unauthorized firmware. Various countermeasures were employed in the bootloader design to prevent these five attacks.

2.2 Implemented Defense

- Internal RC Oscillator
  - Completely neutralizes clock glitching attack
  - Removes signal to help synchronize Brown-out attacks or side channel attacks
- Random Clock Switching
  - Switching the clock between $1 - 8MHz$ at random times
  - Makes brown-out attacks and side channel attacks harder to synchronize
- Random Delays
  - Makes brown-out attacks and side channel attacks harder to synchronize
  - Adds "dummy" instructions to thwart side channel analysis
- Brown-out Detection Fuse
  - Prevents slow brown-out attacks
- Attempt at Constant Time Functions
  - the quickRand() function required 0x35 instructions to execute, regardless of the random number generated. This makes timing attacks to discover the random number nearly impossible, forcing the user to rely on power side channel attacks.

- Even power EEPROM reads
  - the Hamming Distance of each byte read is the same, making power side channel attacks more difficult
- CBC-MAC Verification
  - Entire firmware CBC-MAC must be verified before uploading
  - Prevents most unauthorized firmware installation
- Side-Channel Resistant Cryptography Code
  - Multiple S-boxes and dummy instructions to improve side channel resistance
  - Random operation shuffling to improve side channel resistance: The number of required power traces would be 96 times more than what is needed for attacking an unprotected one, because there are 5 dummy operations for each byte operation.
  - All of the registers are initialized with random numbers before loading a secret, so Hamming Distance model would never work.
  - All the linear operations in AES are protected by additive masking. So the only way to attack this AES implementation by side channel would be to attack the output of the SBOX using Hamming Weight model.

2.3 Costs

- The use of the internal clock and clock switching incur a cost of upload time due to the clock speed being $1MHz$ or $8MHz$.
- Initial difficulty in tuning the internal clock

2.4 Integration

- Many of the countermeasures were simply fuse settings which did not require much in the way of integration
- The same randomness was used for clock switching and random delays
- The reduced clock speed required the use of a custom random function using a Linear Feedback Shift Register to meet upload time requirements

2.5 Residual Vulnerability

- Had to use the LFSR for light-weight randomness
  - Vulnerable because maximal state
  - An attacker can potentially follow the pattern to know when the clock switches or random delays would happen to help with brown-out attacks or side channel attacks
  - Could defend against this by using clock jitter
    - Uses randomness from manufacturing process
    - Differences, called jitter, between the Watch Dog Timer and the internal RC oscillator can be used to generate randomness
    - Cost: increased code size
    - Impact: Should make clock switching and delays very random