Quarkslab's website

SOCIAL

- atom feed
- twitter
- github

CATEGORIES

- Android
- Android, ReverseEngineering
- Challenge
- Cryptography
- Development
- Exploitation
- Fuzzing
- Hardware
- Hardware, ReverseEngineering
- Kernel Debugging
- Life at Quarkslab
- Maths
- Obfuscation
- PenTest
- Program Analysis
- Programming
- ReverseEngineering
- Software
- Vulnerability

TAGS

RFID: Monotonic Counter Anti-Tearing Defeated

Date Tue 18 May 2021 By Philippe Teuwen Christian Herrmann Category Hardware. Tags hardware NFC RFID proxmark3 EEPROM tear-off MIFARE Ultralight EV1 NTAG

Tear-off techniques to the next level.

Introduction

For this second post in collaboration with Iceman, we will briefly present how the generic tear-off tools presented in the previous blog post *RFID: New Proxmark3 Tear-Off Features and New Findings* [2] were used to defeat a secure monotonic counter implementation present in some models of MIFARE Ultralight and NFC cards from NXP.

Bypassing this security feature can have different impact depending on how the vulnerable cards and the specific feature is used in different systems. This blog post will only describe the technical matters of the issue and not analyze the possible impact on various systems. The issue was disclosed to NXP and the vendor has published Application Notes (referenced below) describing possible mitigations.

We won't repeat what was explained in length in the previous blog posts [1] and [2]. The main results we'll use are that each bit gets erased at its own pace depending on stochastic manufacturing process, and that some bits can be weakly programmed, i.e. they are not fully programmed and when being read back, they are sometimes interpreted as `1` and sometimes as `0`.

MIFARE Ultralight EV1 Monotonic Counters

The MIFARE Ultralight EV1 [3] contains three 24-bit monotonic counters with anti-tearing support, which means one can increment a counter by an arbitrary value but never decrement it.

The available commands related to these counters are the following:

- `INCR_CNT` to increment by 0 or a positive value;
- `READ_CNT` to read the current counter value;
- `CHECK_TEARING_EVENT` to read an anti-tearing 8-bit validity flag.

The commands accept a *counter number* as parameter as there are three counters, but to keep things simple we'll assume there is only one counter.

The validity flag is presumably written at the same time as a new counter value is written and it should be equal to `0xBD`. Any other value means that an increment operation was interrupted and, internally, the new counter value could not be fully written. If such event occurs, the counter keeps its previous value.

Under the hood, the counter is saved in two memory slots, similar to the *Protection Words* of the EM4305 we defeated in our previous blogpost. But here, the memory slots are not directly made available and `READ_CNT` implements some logic to return the highest of the slots containing a valid value (i.e. associated to a valid 8-bit flag). The same logic is applied for `INCR_CNT` to decide which slot to read from and in which slot to store the new counter value. On the other side, `CHECK_TEARING_EVENT` returns either the corrupted flag if one is corrupted, or the flag of the slot containing the lowest value.

Thanks to these anti-tearing mechanisms, the counter should never be corrupted or decreased.

Experiments and strategy

We observed that by tearing an `INCR_CNT` by 1 at the limit of getting a valid flag, then reading the counter several times, sometimes a `READ_CNT` returns the previous counter value, while `CHECK_TEARING_EVENT` returns a valid flag.

A possible explanation is that while the flag was properly written, one bit of the new counter was weakly programmed and when it's read as a `0`, the counter value is lower than the old counter value still present in the other slot and yet `READ_CNT` returns the old counter value as being the current one.

Based on this observation, we developed a strategy to reset completely the counter back to zero.

Imagine we set the counter to a power of 2, e.g. `0x000100` and the only bit set is a bit we can program weakly. This counter slot could be read as `0x000100` or `0x000000`. When it's seen as `0x000000`, the other slot with the previous counter value will become the current counter and, as in our first experiment, we only manage to rollback the counter. But if we invalidate the other slot by tearing an `INCR_CNT` sooner to corrupt the flag, then `0x000000` will become the current counter value.

Let's define the two slots as A and B and let's assume that when both slots have the same content and have a valid flag, B gets selected as the current one, i.e. when one reads the counter, slot B value is returned and when one increments the counter, the new value is written in A.

Our strategy is the following:

- Increase the counter to the closest $2^N - 1$ value. If e.g. the counter is equal to `0x00008F`, execute an `INCR_CNT(0x70)` to reach `0x0000FF`;
- Execute an `INCR_CNT(0)` so both slots are equal, e.g. `A:0x0000FF`, `B:0x0000FF`;
- Execute an `INCR_CNT(1)` and tear near the end of the *write* operation. Hope for a counter set to 2^N with a weak bit. e.g. `A:0x000?00`, `B:0x0000FF` with both flags still valid;
- Execute an `INCR_CNT(0)` and tear before the *write* operation. Hope A is read as 2^N so B gets corrupted. e.g. `A:0x000?00` and B flag indicates B is corrupted;
- Execute an `INCR_CNT(0)`. Hope A is read as `0x000000` so A is copied to B, e.g. `A:0x000?00`, `B:0x000000` and both flags are again valid;
- Execute an `INCR_CNT(0)`. Hope A is read as `0x000000` so B is copied to A and they become both stable, e.g. `A:0x000000`, `B:0x000000`;
- If it fails, try again. If the counter increases slightly with attempts, it's not a problem as once the attack is successful we can bump the counter back to $2^N - 1$ and try again;
- If after a while, there is no indication that bit N can be weakly programmed, move to the next bit and bump the counter to $2^{N+1} - 1$.

The strategy must be complemented to adjust automatically the timings and to cover all the possible outcomes, including corrupted flags and weakly programmed flags.

Note that the probability of getting a `0` or a `1` when reading a weak bit can be influenced by controlling the distance between the tag and the reader. Closer to the reader you'll get more often a `1` and further away a `0`.

Results

We have demonstrated the possibility to reset completely a MIFARE Ultralight EV1 counter despite its anti-tearing features and we reported the issue to NXP. The vendor had reproduced our findings and acknowledged it as a vulnerability.

According to NXP, the list of affected products is the following.

In MIFARE Ultralight family:

- MIFARE Ultralight EV1, MF0UL;
- MIFARE Ultralight C, MF0ICU;
- MIFARE Ultralight NANO, MF0UN.

In NTAG 21x family:

- NTAG 210(µ)/212: NT2L1, NT2H10, NT2H12;
- NTAG 213 (TT/F) /215 /216 (F): N2H13, NT2H15, NT2H16.

None of these products are Common Criteria certified.

Other security features likely based on hidden slots might be affected by tearing events and weak bits too, such as OTP bits or Lock bits. But it seems much harder to corrupt them and if the attack succeeds, it will probably clear only a few bits.

Mitigations

An update of *Application Notes* AN11340 [5] and the new AN13089 [6] propose mitigations, such as doubling writes on OTP and Lock bits to update all internal slots and using the upper range of the counters. For example, if an application needs a counter from `0x000000` to `0x000100`, use a counter from `0xfffeff` to `0xffffff`. And when it reaches `0xffffff`, issuing a dummy increment by zero to update the other internal slot will prevent any further attempt to affect the counter as it will only affect slot A while slot B will always have priority when the counter is read.

Adding a MAC (*Message Authentication Code*) may help too but beware of rollback attacks.

The described attack has been realized with the generic tear-off tooling (`hw tearoff` combined with `hf 14a raw`) we added to the Proxmark3 RRG code repository as explained in the previous blogpost [2].

Conclusion

This third blogpost on the EEPROM tearing topic shows that designing a security function while taking care of all the possible weird effects of weak bits is quite challenging. This leaves a number of opportunities for attackers to defeat security features based on EEPROM, even in the presence of anti-tearing countermeasures.

So far, we analyzed only RFID EEPROMs but the same type of issues might probably be found on other non-RFID EEPROMs as well. So, look around you and try to find other interesting targets!

We will present an overview of our tearing techniques, their consequences and their use to defeat various security features at SSTIC 2021 [4] the 3rd of June at 15:00 CET. The talk will be mainly in French but a (long!) article in the proceedings will be published in English at https://www.sstic.org/2021/presentation/eeprom_it_will_all_end_in_tears/. We will cover this attack in greater detail as well.

We want to highlight that we appreciate the cooperation of NXP leading to this coordinated disclosure.

-- @doegox & @herrmann1001

PS: Many thanks to colleagues and friends who proofread this article.

Timeline

- 2020/10/15: <=> we alert NXP PSIRT and receive immediate attention
- 2020/10/28: <= NXP verifies and acknowledges the vulnerability
- 2020/12/17: <=> agreement on coordinated disclosure calendar
- 2021/01/20: <= NXP shares the list of affected products and a draft of the revised Application Notes with mitigations
- 2021/03/05: ... we submit our article at SSTIC
- 2021/05/07: <= NXP publishes both Application Notes [5] and [6]

[1] https://blog.quarkslab.com/eeprom-when-tearing-off-becomes-a-security-issue.html
[2] *(1, 2, 3)* https://blog.quarkslab.com/rfid-new-proxmark3-tear-off-features-and-new-findings.html
[3] https://www.nxp.com/products/rfid-nfc/mifare-hf/mifare-ultralight/mifare-ultralight-ev1:MF0ULX1
[4] https://www.sstic.org/2021/presentation/eeprom_it_will_all_end_in_tears/
[5] *(1, 2)* AN11340 MIFARE Ultralight EV1 Features and Hints - Rev. 3.2 https://www.nxp.com/docs/en/application-note/AN11340.pdf
[6] *(1, 2)* AN13089 NTAG 21x Features and Hints - Rev. 1.0 https://www.nxp.com/docs/en/application-note/AN13089.pdf

Comments