MIFARE Classic and NFC Card Vulnerability

Project Repo: https://github.com/rps9/card-vulnerabilities

James Conlon
Ryan Smith
Zhiheng (Alan) Xu
Jerry Zheng

Introduction

Our mitigations are an addendum to the details in the "Making the Best of Mifare Classic" [1] paper, which comes to conclusions similar to ours, that there are ways to secure Mifare Classic cards, but in many situations, it is better to shift to a different technology. The MIFARE classic card is widely used today, as seen in hotels, offices, and public transit, so we are determined to evaluate the effectiveness of the security of these cards. Throughout this paper, we will explore common attack strategies against these cards and propose mitigations that will ideally combat these attacks. Ultimately, we will draw definitive conclusions on our findings and evaluate the pros and cons of the MIFARE Classic cards.

Background

For this project, we investigated vulnerabilities in Near Field Communication (NFC) cards, focusing on Radio Frequency Identification (RFID), with a physical implementation on a MIFARE Classic card. To begin, we aimed to understand what RFID technology entails. Based on our research, RFID is a wireless communication technology that uses electromagnetic fields to identify and track tags attached to objects automatically. [2] Each RFID system typically consists of three key components: a small radio transponder, a radio receiver, and a transmitter. When the RFID tag is within the range of the reader, it is activated by an electromagnetic interrogation signal. Once powered, the tag transmits stored digital information, such as a unique identifier, back to the reader. This exchange of data allows for contactless identification, which is widely used in applications like access control, public transportation, inventory management, and payment systems. However, the convenience of RFID also introduces security concerns, especially in older implementations like the MIFARE Classic, which is known for its cryptographic weaknesses.

The motivation for this project comes from the fact that these cards are commonplace as transit cards here in Boston and in many other cities. They are used in access control for subways, hotels, workplaces, and other examples due to their affordability and ease of use.

Approach

The scope of the project is to attempt to evaluate cloning viability in practice and evaluate ways to increase the security of MIFARE Classic cards. We first evaluated real-life examples of

cloning and the ways an attacker can do this to create an exhaustive threat model. After that, we evaluate the viability of any potential mitigations and determine if the MIFARE Classic cards we have could possibly be secured. Lastly, we developed our proof of concept mitigations and evaluated how they would work in practice.

To demonstrate the problem and test it later, we bought an nfc reader (PN532) along with blank cards that could be written to. [3] We found that we could easily read information from MIFARE Classic cards and write it to the blank card, effectively making a copy of it.

Security Findings of Mifare Classic

It is well established that Mifare Classic cards use a proprietary Crypto-1 stream cipher that has many exploitable weaknesses, broken in 2009. [4,5] The cipher has been broken to the point that attackers can easily recover the secret encryption keys protecting memory sectors of the card, using low-cost equipment like the PN532 we purchased for $8, or an unrooted Android phone with a Mifare Classic App. [6] The key point is that many attacks for key recovery don't even require sniffing traffic between a card and a legitimate reader, meaning cloning can take place with just physical access to a card without needing the official reader to sniff traffic. This makes it easy for cloning to take place, and the weakness of the Crypto-1 algorithm makes it easy to recover encryption keys.

This paper by Wouter Teepe, "Making the Best of Mifare Classic Update" [1], focuses on "State restoration" attacks. An example of state restoration is CharlieCards, which seemingly do not use a server-side database to track the value of the card – instead, the value of the CharlieCard is stored directly on the card. [7] The CharlieCard is stateful (i.e., the state = dollars on card), so a state restoration attack would be loading a CharlieCard with $100, spending it down to $0, then simply restoring the state of the $100 card, which would essentially give you unlimited swipes. The paper concludes that state restoration is not internally preventable with the way the card works, without verifying server-side based on the card's identity. The paper proposes implementing a system of PKI on top of the cards so an attacker can't rollback the card state, as the signature for the restored $100 card would not be accepted server-side. This is the solution we think is likely to be implemented to some degree by the MBTA. This does mean that the CharlieCard is technically not fully stateful, which means you cannot adjust the state (value) of your CharlieCard itself without verifying with the private signature of the MBTA.

However, the Teepe paper does not come up with a working mitigation for cloning attacks – these are simply attacks where an attacker is copying from a legitimate card to their own cloned card – e.g., cloning someone else's CharlieCard to your own so you can spend their balance down. True cloning prevention is only possible with a way to have hidden blocks on a card. An example is Mifare DESFire, which uses real cryptography (AES) for hidden data blocks. Mifare Classic cards do not have anything functionally resembling that, as we established the Crypto-1 is broken. So, since there is no way to create hidden blocks on Mifare Classic cards,

anyone can clone a card, and there is no way to prevent a full clone once the keys are compromised.[1]

For that reason, we came up with our own set of mitigations to help increase the security of Mifare Classic cards by changing social practices for systems that already use Mifare Classic, and using different schemes to reduce the risk of a cloned card poisoning a system.

Threat Model

The primary attacker we are considering is an attacker who has gained access to someone else's Mifare Classic card. The attacker's goal is to get a functional copy that they can use to get into a building or room. For this attack model, we need to consider ease of attack in two ways: technical knowledge required and physical access to a card that they should not have access to.

The basic understanding that we came to while developing our mitigation strategy is that hidden data is the key to preventing cloning. We already established that the technical knowledge is very minimal for a clone: due to the fact that most cards use a common set of encryption keys such as the library used by the Fipper Zero which has a list of open source keys you can use in a dictionary attack. [9] For that reason, key brute forcing is trivially done, and having knowledge of the card's origin (like knowing the card is a CharlieCard), can reduce the number of keys to brute force down to a list of less than a hundred common ones. Our cards used the default key "ffffffffffff" and if we used it in a production setting without changing the key (which is common), an attacker would be able to clone with minimal effort. We suspect since nfc-mfclassic does not prompt or give a warning when you are writing a card using a default key, that only more savvy administrators would even know a key is involved in Mifare Classic cards. The nfc-mfclassic tool by default just checks a number of default keys while writing – to the point we did not realize during testing that Mifare Classic cards even had an encryption key because it is so invisible to the admin.

As a result, both sized targets for cloning: large metro systems and smaller offices/labs are vulnerable. Large metro systems (e.g. MBTA, London, Moscow, Los Angeles) are vulnerable since they are likely to have the keys online due to their size (we found all 4 of them on GitHub). [8,9,10] And smaller targets like hotels, labs may be vulnerable because their infrastructure is less advanced than a transit authority, and they are likely still using default keys on their Mifare cards.

Mitigations

Since we established that the MIFARE Classic is broken and impossible to prevent a full clone, our project revolved around ways to mitigate that concern. All three of our developed mitigations only work for a specific use case and are not wholly applicable to any usage of the MIFARE Classic. Many of our mitigations require behavioral changes for users as well as social engineering components to keep in mind as part of our threat model. For example, it would be impractical to have passengers tap two CharlieCards to get into the T, so mitigation 3 would not

[1] Private keys for Mifare Classic cards can be readily found on GitHub, including for CharlieCards. [8]

be a good solution for the security concerns of the MBTA. However, the MBTA implements a system similar to the idea in mitigation 1 (counter), which increases the security of the CharlieCards.

Mitigation 1 - Counter

This mitigation is inspired by some of the ideas in the Teepe paper for preventing state restoration attacks. In the Teepe paper, he proposes adding server-side verification to prevent state restoration. This concept can be applied to preventing cloning.

In this mitigation, we are trying to build a connection between the cards and the security system, though this connection is not always on. The database is a CSV file, and it has the UID for the card and the count that it is on. When a card scans into the system, it checks if the UID and the count on the card are the same as the data in the database. If they match, then the card is rewritten to a new value in the count, and the database is rewritten to the same value. If the values do not match any pair in the database, then the system denies them access. Our implementation cycles through the letters A, B, C, and D in alphabetical order. When reading MIFARE cards, the first line in the first block consistently contains the card's UID (Unique Identifier). A standard MIFARE card has 64 blocks in total. To enhance security and reduce the risk of unauthorized decryption or cloning, we store the counter in randomly selected blocks rather than in fixed locations. In a real-world application, each block holds distinct information, and only our database knows the mapping between blocks and their corresponding data. This mitigation is easy to apply and can prevent most cases of cloning.

Mitigation 2 - Clock in / Clock out

For the clock-in / clock-out mitigation, we followed the example of a company that uses NFC cards to allow employees access to the building. To make sure the building was secure, we developed a set of rules which go as follows:

- The reader only accepts unique cards assigned to each employee.
- Scanning in counts as clocking in; scanning out counts as clocking out.
- A card already scanned in cannot be used to scan in again until it has been scanned out.
- Scanning into the building outside of regular work hours is blocked.
- If the employee is on PTO or sick leave, the card will not work.
- If an attacker scans in before the employee arrives, the system alerts that a non-employee has scanned in.
- If an employee forgets to scan out the previous day and can't get back in, the system detects the long clock-in period and treats it as user error, not an attack.
- When the employee scans out, the system writes unique data to the card that it expects at the next scan.

To illustrate how this works, we can go through an example:

Let's say that an attacker is trying to break into the building to steal sensitive data, and they have cloned an employee's card. The only way they could get in undetected is if the employee whose card they cloned was no-call-no-show.

To implement this, we just wrote a simple Python script that has a database of verified employee cards stored in a CSV file that keeps track of whether they are clocked in or out. When they clock in, it checks a few things, like if they are already clocked in, if it is outside of regular work hours, if they employee put in for PTO or a sick day, and if they have been clocked in for an extended period of time (in the case that they forgot to sign out). This makes it so there is a much smaller window in which the attacker could get into the building. It is hard to measure exactly how much the attack surface would be reduced by, but it would be fair to say that if an attacker has a cloned card, they will be caught upwards of at minimum 98% of the time. This number is obtained from the Bureau of Labor Statistics, which lists the absence rate for work around 3.6% on the high end, and we can overestimate the amount of no-call-no-shows to be about 2% of the time. The only way this system lets an attacker through is if the employee whose card is cloned is a no-call-no-show, which is where the 98% number comes from. So without our system in place, if an attacker cloned a card, they would be able to get into the building undetected 100% of the time, and now, with our system, we can estimate that they would only get in 2% of the time at the very max, more than likely it is even less than that.

Mitigation 3 - Dual Tags

This mitigation is inspired by common Two-Factor Authentication schemes like the one used by BU with Duo. The key idea is that the dual-tag system must take advantage of two different form factors of Mifare Classic cards – one that attaches to a keychain, along with the traditional format of a card that goes in the wallet. This is different from the previous two mitigations that rely on actively managing a card state, it instead makes it more difficult for an attacker to clone. The secondary tag "anchor" goes on the keychain. The writing process involves first randomly generating a key K. The payload P is the primary identifier (Person's name, employee ID#, etc.), and is encrypted with AES like done in class. We use AES-ECB in the code, but AES-CBC would be a better choice in a real implementation to prevent patterning in P. Then the card "primary" can be written as C=AES-encrypt(P,K) which is written to the primary card, and the secondary gets the key written to it.

Advantages:

- Less likely to be skimmed by "proximity" where an attacker sniffs a card through a wallet or purse, because having access to any one key is useless.
- Using AES on the primary makes plaintext state restoration attacks (Like those theoretically possible with CharlieCards) practically impossible as an attacker cannot derive any meaning from the Cipher on the primary tag.

Disadvantages:

- Gaining access to both tags still allows cloning
- Difficult to implement: more complicated tag reading algorithms and behavior changes are needed to present 2 tags.
- High friction for users, slow
- Not useful for high throughput systems like mass transit, intended for smaller access points (computer labs, office buildings).

Evaluation and Results

All of our mitigations have caveats. For the first mitigation—the use of a counter—there remains a limited chance that an attacker could successfully clone and use the card. Since MIFARE cards cannot connect to the Internet, real-time detection of cloning is not possible. However, this approach still prevents the majority of cloning attempts. The only viable attack scenario would be if an attacker clones the card and uses it before the legitimate owner does, ensuring that the owner does not use the card for a period afterwards. Otherwise, they won't succeed on this.

Timing results are another concern. We implement the "nfc-mfclassic" toolset from *libnfc* which is a C based library for NFC tools, we use a hardware PN532 module for reading and writing. [11,12,13] We wrote a primary file "nfc_functions.py" that all our mitigations use so the timing results are consistent across mitigations. It is very slow, it takes an average of 1.15s to dump a card's data (reading). For writing, it is also very slow as the entire file is sent back to the card. It takes 1.76 seconds for a write. Basic read and write operations are implemented by all our mitigations and the full timing results are aggregated in the table below. Raw data is in the timing_results folder in the project repo.

	Baseline	Mitigation 1	Mitigation 2	Mitigation 3
Swipe in/read (R) Writing card (W)	**R: 1.15s** W: 1.76s	**5.75s**	**1.15s**	**R: 6.6s+** (depending on speed/skill of user) W: 12.3s

Our implementation of nfc-mfclassic to read and write uses a strategy of reading by downloading the entire file and reuploading the entire file (1024 bytes @ 106kbit/s) which has a theoretical speed of ~0.2 seconds at max speed to dump the entire card. [2] Our result for this is 1.15 seconds. We are much slower than a theoretical max, but our result can be made faster by only operating on a single block rather than doing operations on the whole card. E.g. only modifying the counter from A→B rather than reloading the entire card's data. We were not able to find a working way to do this with nfc-mfclassic with our PN532 board. As a result, this would cause throughput issues in a high traffic environment (think Kenmore station where taps go from ~350ms to ~5s. Thus it is absolutely necessary to reduce timing by finding a nfc library that is faster or able to write to specific blocks rather than requiring the entire file to be written.

Something mentioned in the project proposal that we were not able to implement is adding a security layer to NFC directly using nfcpy. While nfcpy does include in its documentation references to the ability to simulate NFC traffic with a udp driver module, we found that this is not currently possible with the source repo without extensive modifications to the source code. [14] It seems to us that the reference in the documentation is not fully implemented in the current main branch, and the nfcpy project itself is abandoned. The

documentation says to use “send_rsp_recv_cmd” which is not implemented in the UDP simulator in the source. For that reason, we decided to pivot our project to evaluating NFC cards with real hardware using a PN532 and MIFARE Classic cards, which have known vulnerabilities.

Conclusion

Based on our research and testing, it is evident that fully securing the MIFARE Classic card is impossible. Despite our proposed mitigation strategies, attackers can still exploit known vulnerabilities using relatively accessible tools and techniques.

Given these weaknesses, the most effective long-term solution is to phase out MIFARE Classic cards entirely and transition to more secure alternatives such as MIFARE DESFire or other modern smart card technologies that offer robust encryption and security protocols. While this may come with increased costs and implementation overhead, it significantly reduces the attack surface.

That said, the MIFARE Classic card continues to be useful due to its low cost, simplicity, and high reliability in operational environments. For organizations or applications where budget constraints take precedence over strict security requirements, the MIFARE Classic can still serve as a functional solution, provided its limitations are accounted for.

References

[1] https://www.cs.ru.nl/~wouter/papers/2008-thebest-updated.pdf
[2] https://www.researchgate.net/publication/221631846_Dismantling_mifare_classic
[3] https://www.elechouse.com/product/pn532-nfc-rfid-module-v4/
[4] https://www.sciencedirect.com/science/article/pii/S1363412710000348
[5] https://helix.stormhub.org/papers/Fredriksson%20B.,%20A%20case%20study%20in%20smartcard%20security%20-%20Analysing%20Mifare%20Classic.pdf
[6] https://play.google.com/store/apps/details?id=de.syss.MifareClassicTool&hl=en_US&pli=1
[7] https://medium.com/@bobbyrsec/operation-charlie-hacking-the-mbta-charliecard-from-2008-to-present-24ea9f0aaa38
[8] https://github.com/zhovner/proxmark3-1/blob/master/client/default_keys.dic
[9] https://gist.github.com/noproto/63f5dea3f77cae4393a4aa90fc8ef427
[10] https://github.com/RfidResearchGroup/proxmark3/issues/2019
[11] https://www.mankier.com/1/nfc-mfclassic
[12] https://github.com/nfc-tools/libnfc
[13] https://www.aliexpress.us/item/3256805076433294.html
[14] https://nfcpy.readthedocs.io/en/latest/modules/clf.html#module-nfc.clf.udp

Additional References used in code development and Mitigation strategy

https://www.blackhillsinfosec.com/rfid-proximity-cloning-attacks/
https://security.stackexchange.com/questions/234637/hardening-nfc-tags-for-authentication
https://security.stackexchange.com/questions/63483/how-do-nfc-tags-prevent-copying
https://electronics.stackexchange.com/questions/103741/is-it-possible-to-provide-security-in-passive-rfid-tags
https://forum.arduino.cc/t/prevent-nfc-tag-cloning/276786
https://seritag.com/learn/using-nfc/nfc-tag-authentication-explained