Hung out to dry: Airing the dirty laundry of stored value washing cards

Alexander Hunt & Aidan Nakache

March 13, 2025

Abstract

This paper details a comprehensive reverse engineering analysis of stored-value laundry cards, prevalent in facilities worldwide. The widespread adoption of localised contactless payment solutions, attributed to their convenience, necessitates understanding their internal operations. This analysis explores the mechanisms behind value storage and modification within these cards. During this investigation, a data structure was identified that presented a significant vulnerability. The implications of this vulnerability raise serious concerns, which extend beyond laundry facilities, potentially impacting the security of similar contactless systems globally.

Introduction

This paper details an investigation into the stored-value memory organisation of Kiosoft systems, typically found within apartment complexes, colleges, multifamily housing, hotels, and residential complexes. Kiosoft systems allow customers to purchase a card from a machine, activate it, repeatedly top up the balance, and use that card to activate washing and drying machines for a fixed price. The research in this paper focuses on WASH Laundry specifically, which is not an insignificant entity within North American self-service laundry providers.

During this project, a total of $47 was legitimately added to the cards to observe changes, with testing expenditures not exceeding $15. All funds spent were contributed legally and not ill gotten. Vulnerability research must be conducted in good faith.

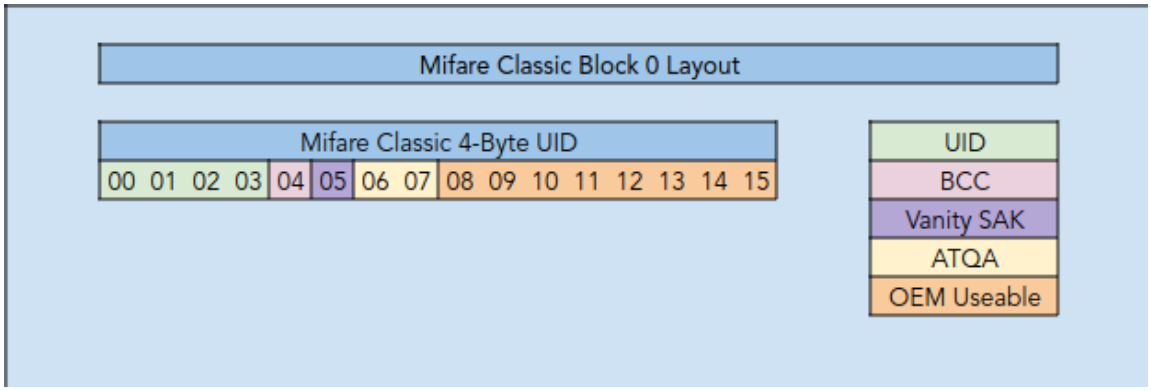
The Proxmark 3 Easy 512kb, running the Iceman Firmware[1], was used to investigate, manipulate and emulate the card belonging to the WASH system for this research project.

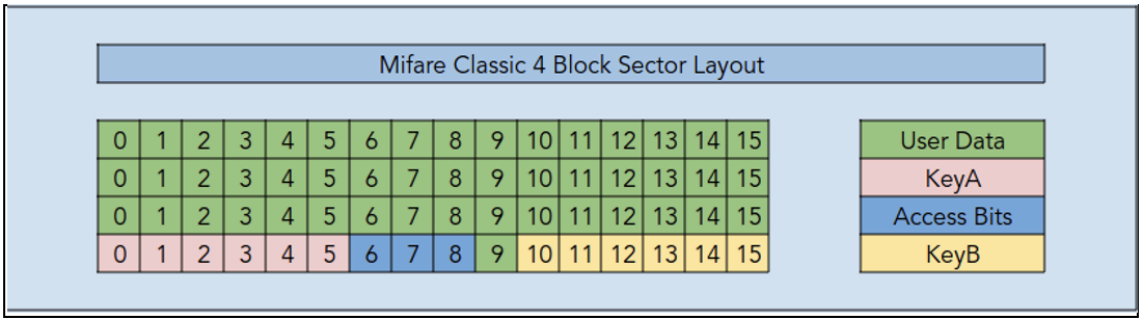
Card Structure

The cards being used by WASH are NXP MF1S50yyX[2] “Mifare Classic 1kb 4-Byte UID”, a 13.56 MHz ISO14443A RFID chipset

Structure

- Memory Layout: The Mifare Classic card features a memory structure divided into sectors. A 1 KB card contains 16 sectors, each with three blocks of data (comprised of 16 bytes each) and a sector trailer block containing the sector access keys and access conditions, which define how those keys can interact with the memory of each block in the sector.
- Access Conditions: Each sector has two keys (A & B) and access conditions (ACLs). The ACLs determine how those keys can interact with the blocks within the sector.
- Block 0: Block 0, within Sector 0, holds the UID (Unique Identifier) of the card as well as representations of the BCC (A XOR of the UID), SAK, and ATQA, as well as bytes available to be programmed by the OEM.

- 



Finding the Value

To begin searching for the value in memory, the known current value of the card ($18.75) was used as a starting point. Based on previous experience and logical reasoning, it was hypothesised that if the value were stored, it would likely be represented in a single variable containing the value as a cent amount, as opposed to two separate variables for dollars and cents. Based on this, 1875 was converted to hexadecimal, yielding 0x0753, which was then compared to the memory dump of the card in its current state. This comparison revealed that blocks 4 and 8 each contained two pads of 0x5307, a little-endian representation of the value. Further testing of adding $5 to the card, increasing the value to 2375 (0x0947 = 0x4709), confirmed that blocks 4 and 8 contained the value. With confirmation that the value was stored within the card, progress

could be made in identifying the mechanism controlling value change and labelling other variables that were observed to change between natural value modifications.

Labelling the Variables

Through testing, it was possible to isolate several variables and discern their importance to the system. A comparison of an activated card to a card that had not yet been activated but originated from the same dispenser in the laundry room revealed the information the encoder places into the memory blocks. It was noted that before activation, the cards already contained some data and example mirror functions that applied to various other parts of the memory. Testing showed that none of the pre-encoded data was accessed by the system after the point of enrollment, leading to speculation about its purpose and why it remained after card enrollment.

Variables were labelled as the research progressed to facilitate referencing specific pads and the operations applied to them during various tests.

Block 1: 30 30 00 01 00 00 00 12 98 89 00 00 01 11 EE 45
Block 2: 01 01 4C 55 55 0B 00 00 00 53 07 01 00 00 00 12
Block 3: 45 71 75 69 70 20 68 77 89 00 4C 75 75 31 37 36
Block 4: 53 07 0B 00 AC F8 F4 FF 53 07 0B 00 04 FB 04 FB
Block 5: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 7: 45 71 75 69 70 20 68 77 89 00 4C 75 75 31 37 36
Block 8: 53 07 0B 00 AC F8 F4 FF 53 07 0B 00 04 FB 04 FB
Block 9: 36 42 0F 00 C9 BD F0 FF 36 42 0F 00 09 F6 09 FE

Card Number (printed on card): 0 12 98 89 - (not real CN) Redacted due to being tied to the owner.

Value At Last Topup (VALT): 53 07
Transaction ID: 4C 55 55
Incremental Mirror Byte: 0B
Terminal Byte (XOR): 12
Value Pad (Current Value + Incremental Mirror Byte): 53 07 0B
Middle Bytes: AC F8 F4
Under Value Byte (UVB): 36
Under Middle Byte (UMB): C9
Sector KeyA: 45 71 75 69 70 20 - (not real key value) Redacted due to being specific to site
Sector Access Conditions: 68 77 89
Sector KeyB: 4C 75 75 31 37 36 - (not real key value) Redacted due to being specific to site

Card Number

The seven-digit decimal value printed on the card's reverse side was located within block 1 of the card's memory. Although the card's memory is represented in hexadecimal, the card number is deliberately encoded as a visible decimal value in hexadecimal format, eliminating the need for conversion. This card number is used when purchasing online "top-up vouchers," which can be redeemed at the wall-mounted encoder.

Value at last top-up

The VALT bytes, like the current value bytes, are encoded in little-endian. The VALT bytes store the card's value at the time of its last top-up; for instance, if $20 were added to a card with an existing balance of $5, the VALT bytes would reflect $25. This value remains static until another top-up occurs. The washing machines do not verify a positive discrepancy between the VALT value and the current value to prevent the current value from exceeding the VALT value.

Consequently, the VALT value can remain unaffected when the current value is manually modified, with no negative effect.

Transaction ID

The transaction ID is a 3-byte pad that is updated in relation to transactions performed on the wall-mounted encoder. An invalid transaction ID results in the wall-mounted encoder rejecting the card and preventing fund additions. The logic for calculating transaction IDs has been omitted to prevent the creation of "ghost" cards capable of bypassing the wall-mounted encoder.

Terminal Byte

The terminal byte, located at the end of block 2, is a CheckSum8 XOR of the preceding bytes within that block. This same function is mirrored in the static bytes of block 1.

Value Pad

The value pad consists of three bytes: the first two represent the current associated value in little-endian, and the third is the incremental mirror byte. Four value pads exist within the card's memory.

Middle Bytes

Middle bytes, named for their position within each set of value pads, consist of 3 bytes that are the result of a bitwise negation of the first 3 bytes in the value pad.

Under Bytes

Under bytes, positioned below other relevant variables, change only when the card's value is increased; they remain static when credit is spent on the machines.

- Under value pad decrements by 1.
- Under middle bytes, increment by 1.

Internal ID

Block 13 contains an internal ID that is relevant to the card and the region it is dispensed from, this internal ID is pre-encoded, it is visibly tied to the region as it is ascii in hexadecimal and has a format of (LOC)[ID] for example, a card from florida would have FL123456.

Testing value increments and decrements

Top-up Procedure

During a $5 top-up using the encoder, numerous variables within the card are modified, including:

- Middle Bytes (Byte #1: +0x0C, Byte #2: -0x02, and Byte #3: -0x01)
- Value Pads (Byte #3: +0x01) (top-up counter)
- Value At Last Topup
- Incremental Mirror Byte (+0x01) (byte #3 of value pad aka top-up counter)
- Under Value Byte (-0x01)
- Under Middle Byte (+0x01)

Based on this information, it was observed that the middle bytes changed consistently with every $5 increment. Consequently, those bytes could be modified by the same constants to continually add $5 to the card without payment, and the machines would accept it. However, the encoding logic was not yet fully understood, preventing the specification of exact credit amounts to be added.

Operation of Dryers and Washing Machines

During the use of the washing machines and dryers, only minor changes were observed compared to credit additions. Specifically, only the value and middle bytes were altered. All other variables remained unchanged.

Encoding Pattern Discovery

An attempt to increment the value by modifying only the value bytes to a sum higher than the currently encoded credit proved unsuccessful when tested against a washing machine. This indicated that other variables in the surrounding pads required alignment with the value pad to be accepted by the machines. Utilising a collection of dumped samples from each card used on the machines, a relationship between the value bytes and the middle bytes was discovered. Each byte in the value pad corresponds to a byte in the middle bytes pad through a linear relationship.

The relationship follows this pattern: the first byte in the value pad is subtracted from FF (255) to determine the corresponding byte in the middle bytes at the same position. This discovery was made through the application of the slope formula:

$$\frac{y_2 - y_1}{x_2 - x_1} = m$$

Where:
y_2 = First middle byte after adding $5

y_1 = First middle byte before adding money

x_2= First value byte after adding $5

x_1 = First value byte before adding money

From this calculation, the slope was determined to be -1. Using a data point and this slope, the full linear relationship was derived:

$$y = -x + 255$$

Applying this formula to the actual value bytes, each middle byte was calculated as:

$$Middle\ Byte = FF(255) - Value\ Byte$$

This pattern holds for all three bytes in the middle bytes pad. As a result of this discovery, it became possible to successfully modify the balance on the card, not only in $5 increments but in any amount down to the cent. When tested on a washing machine, the modified balance was recognised and accepted.

Function Mirrors

It was observed that within the card, both before and after activation, static byte pads mirrored essential card functions, such as stored value examples and middle byte relationships. This redundancy was noted, as these bytes served no apparent purpose; altering or removing them entirely did not impact the card's functionality or generate errors from the encoder or the machines.

It became increasingly apparent that the card presented significant vulnerabilities. Notably, these mirrors were discovered after the logical functions applied to other bytes had been determined. Had the relationships between the static bytes been examined more closely earlier on, a near-complete list of balance manipulation functions would have been revealed.

Why a Valid Card Cannot Be Created from nothing

Even with a blank MIFARE Classic card and a card writer, the creation of a fully functional counterfeit card for the WASH system is not feasible. This is because each officially dispensed card contains both an externally printed number and a card number within block 13, which is tied to the facility's geographic location. Neither of these IDs can be fabricated, as their validity is verified during initial enrollment. Furthermore, the sector keys are exclusive to each facility. Therefore, a WASH deployed card cannot be created on

demand; a legitimate base card is required to be modified/or duplicated to produce a working card.

Challenges in System Upgrades

Upgrading this system to enhance security would necessitate substantial investment in hardware, software, and labour, requiring a complete overhaul for all facilities using this encoding. Even without financial constraints, migrating to a more secure system would involve a complex user migration. The absence of card association with specific individuals or precise addresses, along with the need to transfer existing balances securely, presents a significant challenge. Ensuring a seamless user experience across diverse demographics, while maintaining simplicity, further complicates the upgrade.

Financial considerations are not something to be waved away, and the impact of this exploit is relatively minor to the overall financial performance of companies employing this setup. RFID hacking remains a specialised field, and the financial losses incurred by individual exploitation are negligible compared to the cost of a comprehensive system upgrade.

Conclusion

Research into the Kiosoft stored-value laundry card system reveals a notably insecure implementation that facilitates manipulation through the provision of key function examples. By analysing the memory structure, incremental changes, and relationships

between critical variables across a comprehensive dataset, it was demonstrated that stored balances could be modified with precision and without detection. This highlights the system's lack of robust integrity checks beyond initial card enrollment while in use. While the system's exploitability is evident, its real-world impact is limited by the niche nature of RFID hacking and the economic impracticality of a system-wide upgrade. This serves as another example where "security by obscurity" has been used to justify the omission of security features, which would have significantly impeded, if not prevented, this research.

References

Iceman Firmware - RfidResearchGroup “proxmark3” GitHub project
https://github.com/RfidResearchGroup/proxmark3

MF1S50yyX Mifare classic - https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf

AN10833 Mifare identification type procedure -
https://www.nxp.com/docs/en/application-note/AN10833.pdf