Karlsruhe Institute of Technology

Security Analysis of Online Cashless Vending Systems

Bachelor's Thesis of

Janis Streib

at the Department of Informatics, Institute of Telematics
Decentralized Systems and Network Services Research Group

Reviewer:	Prof. Dr. Hannes Hartenstein
Second reviewer:	Prof. Dr. Martina Zitterbart
Advisor:	M.Sc. Jan Grashöfer

01. June 2018 – 31. October 2018

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 31.10.2018

. . . .

(Janis Streib)

Sperrvermerk

Das vorliegende Dokument enthält bislang unveröffentlichte Sicherheitslücken. Veröffentlichungen oder Vervielfältigungen des vorliegenden Dokuments, auch nur auszugsweise, sind ohne ausdrückliche Genehmigung vor dem Ablauf der Frist am 31.04.2019 nicht gestattet. Die Arbeit ist nur den Korrektoren, den Mitgliedern des Prüfungsausschusses und den betroffenen Parteien zugänglich zu machen. Vor Ablauf der Frist darf dieses Dokument nicht ohne ausdrückliche Genehmigung in der Bibliothek der Hochschule ausgelegt werden.

Abstract

Cashless payment is omnipresent at sites like universities or large companies: If one wants to buy goods at vending machines or in the canteen, or pay for local services at the site, the student, guest or employee card is used for payment. The card is revalued with money by the card holder beforehand. These payment systems can be operated cloud based. Therefore, the information security is vital, to protect the user's money. Moreover, the money of the operator, who guarantees the transactions to the other departments and contractors, has to be protected, too.

In this thesis, the cashless payment system used at the campus of the Karlsruhe Institute of Technology will be analyzed in detail with focus on its information security. For the analysis, the risk management process defined in the ISO27005 will be applied: First, the risks will be identified, analyzed and mitigations will be evaluated. Finally it will be discussed, whether the system can be operated in compliance with *BSI Grundschutz* and if the industry standard PCI PA-DSS would have prevented the problems found in context of this work, if applied during implementation of the system.

The analysis process revealed 13 weaknesses. The discussion yielded, that the payment system cannot be operated in compliance with the *BSI Grundschutz*. If the system would have been developed in compliance with PCI PA-DSS, most of the problems would have been prevented. Finally, a brief list of recommendations for this type of system was created.

Zusammenfassung

Bargeldloses Zahlen ist an Universitäten oder in großen Firmen allgegenwärtig: Möchte man an einem solchen Standort etwas an Verkaufsautomaten oder in der Kantine erwerben, oder für lokale Dienstleistungen bezahlen, bezahlt man mit seinem Studenten-, Gäste-, oder Mitarbeiterausweis. Diese Karte wird dabei zuvor von ihren Besitzern mit Geld aufgeladen. Diese Systeme können Cloud-basiert arbeiten, um die finanziellen Transaktionen zu tätigen. Daher ist die Informationssicherheit bei diesen Systemen von großer Bedeutung, um das Geld der Kunden und das Geld des Betreibers, der für die Abwicklung der Zahlung an die jeweiligen Abteilungen oder Dienstleistern die Verantwortung trägt, zu schützen.

In dieser Arbeit wird das System, das unter anderem auf dem Campus des Karlsruher Instituts für Technologie Verwendung findet, genauer auf seine Informationssicherheit hin untersucht. Dabei wird nach dem im ISO27005-Standard definierten Risikomanagementprozess vorgegangen: Zuerst werden Risiken im System identifiziert, diese dann bewertet und schließlich mögliche Gegenmaßnahmen evaluiert. Abschließend wird diskutiert, ob existierende Sicherheitsstandards wie der BSI Grundschutz oder PCI PA-DSS einen Betrieb zulassen, bzw. ob diese die gefunden Probleme verhindert hätten.

Im Rahmen des Analyseprozesses wurden 13 Schwachstellen identifiziert, das Angriffsrisiko analysiert, sowie Strategien zur Risikominimierung entwickelt und evaluiert. In der Diskussion ergibt sich, dass das System nicht unter Beachtung der Vorgaben des BSI Grundschutzes betrieben werden kann. Auch hätte die Beachtung des Standards PCI PA-DSS während der Implementierung einige der im Kontext dieser Arbeit identifizierten Probleme verhindert. Abschließend wird ausgehend von den vorangegangenen Erkenntnissen eine Liste an Sicherheitsempfehlungen für diese Art von System aufgestellt.

Acknowledgments

I would like to thank *Thomas Fluhrer*, *Alen Kecic*, *Patrick Eble* and *Markus Kopf* from the *Studierendenwerk Karlsruhe AöR* for their support for the realization of this thesis, as well as *Andreas Lorenz* and the legal department of the KIT. I would also like to thank *Felix Dörre* for his help with understanding cryptographic methods and algorithms. Additionally, I would like thank the KIT CERT team, which provided the required tool "IDA Pro" and supported the responsible disclosure process.

Finally, I especially want to thank *Jan Grashöfer*, *Prof. Hartenstein* and the DSN team for the support and competent guidance for writing this work.

Table of Contents

List of Figures

List of Tables

1. Introduction

Online cashless vending systems are common on large sites like university campuses or companies. They are mainly used for payment of food or services on site. The user gets a site-specific identification card. In most cases, this is a Near-Field Communication (NFC) card, which is able to store encrypted data. On sites like university campuses, these cards are often also the student identification card. The card holders recharge their cards with money and use them to pay e.g. at site's vending machines or canteens. At university sites, such cards can also be used to pay e.g. in the library, for print and plot services or laundry in student dormitories. The main motivation to use such a system compared to traditional cash systems with coin acceptors and banknote validators is the lower rate of damaged and robbed vending machines and lower costs for handling cash money. Those systems have to be protected and frequently emptied to prevent burglary. Another advantage are fast transactions. Unlike traditional cashless payment solutions like credit or debit cards, strict requirements for operating these systems have not to be fulfilled, which reduces costs.

At the Karlsruhe Institute of Technology (KIT) the system is provided by the *Studierendenwerk Karlsruhe*, the student service of Karlsruhe. At KIT, about 25.000 students[1] are matriculated and therefore also have access to the payment system, as well as students from other universities in Karlsruhe and Pforzheim. According to their annual report [SWKA18], the *Studierendenwerk Karlsruhe* made about 10 million Euro in sales with its culinary offerings. It can be expected, that most of this money was transferred via the cashless micropayment system. Therefore, the information security of the system is vital, to protect the money of the users and the provider of the system.

The vendor of the system used by the *Studierendenwerk* is the British company *Counter Solutions Limited*. They offer solutions including hardware and software to provide a cashless payment system for vending machines, laundry, cash desks, copy and other services. With their current version introduced in August 2018 at the *Studierendenwerk Karlsruhe*, the system is completely cloud-based. This means, that the NFC-Card is only used as authorization and the account balance is stored on a server in the internet. Cards are charged (this process is also called "revaluing") in special revaluing devices called "Kiosk". Devices built into vending machines are called "Vending Reader". Both need to communicate with a server over a network connection.

In this thesis, we analyze the prepaid card payment solution at KIT. We focus on the software of the Counter Solutions Cashless Campus components "Kiosk" (from now on referred as "Revalue Device") and "Vending Reader", as well as small parts of the backend server. We will analyze them, by observing the behavior and network traffic, as well by reverse engineering of the software on those devices.

[1] `https://www.kit.edu/downloads/Statistik_WS2017.pdf`

There are also terminals to retrieve and display the account history and check the NFC card integrity (from now on referred as "Card Terminal") and Revalue Devices based on the same base hardware as the Vending Readers. These two devices will not be analyzed in-depth in this thesis, because they were not available for analysis. We will neither analyze the security of the NFC cards nor the security of peripheral components like vending machines or point of sale computers.

We will structure the analysis along the risk assessment process described in ISO27005, as this is a common industry standard which gets widely applied to identify and evaluate possible threats in the IT infrastructure of a company. To get an overview over the analyzed product, we first describe the system in chapter 3. After defining our method of analysis in chapter 4, we identify possible weaknesses of the product in chapter 5. The risk identification corresponds to the first steps of ISO27005. After the identification process, we analyze these identified weaknesses in chapter 6 to be able to estimate the likelihood and impact. Afterwards, a risk level score by weakness is derived from those parameters. This allows a classification of the weaknesses. In chapter 7 we will develop a list of possible mitigations for the described weaknesses. After the analysis of the product, we will discuss, whether existing security standards are applicable for this kind of system in chapter 8. Finally, we conclude the findings of this work in chapter 9.

2. Related Work

In the following chapter, we will give a short summary of basic cashless vending terminology, an overview of the information security framework used in this work as well as similar research in the specific area of security in cashless micropayment applications.

2.1. Cashless Vending

The cashless vending industry is organized in several associations: One of them is the European Vending & Coffee Service Association (EVA) and another one the National Automatic Merchandising Association (NAMA, USA). EVA's Electronic Payment Specification (EPS) [EVA-EPS] defines basic requirements for the physical security, internal protocols and the user interface for vending machines. According to EPS, we can differentiate between the following electronic (cashless) payment card types:

Debit cards are used to pay directly from a customer's bank account. By using the card number or the card itself, payments are directly "debited to the customer's bank account and credited to the merchant's account." [EVA-EPS, Section 3.1, p. 10] **Credit cards** are not *directly* debited from the customer's bank account, but "charged to the card holder at fixed intervals in total [. . .], or in partial credit amounts." [EVA-EPS, Section 3.2, p. 11]

While the card types above are somehow linked to a bank account, **prepaid cards** are charged beforehand and are not linked to a bank account. We can differentiate between three types of prepaid cards: Open loop cards are accepted by a wide range of merchants, restricted loop cards can be used in a limited range of companies and closed loop cards can be only used in one company. Closed loop cards are according to the EVA [EVA-EPS, Section 3.4.1, p. 12] the most common electronic payment scheme (95%) in vending food and beverage industry.

2.2. Information Security

To be able to analyze a product systematically, security goals have to be defined. This allows to check, if a weakness breaks any of those goals. According to [Eck18, pp. 7-15] the following basic security goals can be defined:

- **Authenticity**
  The authenticity and credibility of a subject or object can be verified, like e.g. the identity of a debit card used at a payment terminal.
- **Integrity**
  Data cannot be altered unauthorized or unnoticed. E.g. a man-in-the-middle cannot *change* the content of a transaction without getting noticed.

- **Confidentiality**
  Information cannot be obtained unauthorized. E.g. a man-in-the-middle is unable to *see* the content of a transaction in cleartext.

- **Availability**
  Authenticated and authorized users cannot be impaired while exercising their rights. E.g. it is not possible to send requests to a server in a way which makes the server unable to respond to genuine requests properly.

- **Non-Repudiation**
  An action by a subject or object cannot be denied afterwards. E.g. a customer cannot deny, that *they* made a transaction, if a transaction succeeded.

- **Anonymization and pseudonymization**
  Anonymization does not allow the reconstruction of personal data or renders it very hard to reconstruct the data. Pseudonymization is the mapping to a pseudonym. Reconstruction requires access to this mapping. E.g. a username, which is only stored as random text without any information of the real name is a anonymized username. If the real name is associated (or can be associated) with the random username *somewhere*, the username is only a pseudonym.

The structure of this thesis is roughly based on the risk assessment process described in [ISO27005]: In the first phase, the product is analyzed for risks (Risk Identification). These risks are thereafter evaluated for the likelihood and impact of an incident, resulting in a qualitative risk level score (Risk Analysis). In the final phase (Risk Evaluation), each risk level score of the risks is compared against acceptance criteria. After prioritizing them, possible mitigations are evaluated and implemented. In this work, we omit the comparison with acceptance criteria, since we do not operate the system and can not evaluate the business impact of the risks we might identify.

2.3. Similar Research

Matteo Pisani perfomed a small analysis of the near-field communication and Bluetooth based payment system, which allows payment using a smartphone app at vending machines by the vendor Argenta [Pis18]: The user account balance is also stored in a SQLite database on the smartphone. The vending machine trusted the account balance, communicated by the smartphone to the vending device. This allowed manipulation of the local database on the smartphone and even the complete emulation of the payment app with arbitrary account balances, to "pay" the goods in the vending machines.

Mathias Dalheimer did research on electric vehicle charging stations with a custom cashless payment system [Dal17], which revealed significant issues on update mechanisms, authentication and user authorization: Automatic code execution and leakage of the complete configuration data to a removable USB storage was possible by just removing some screws on the device. For user authentication, just the plain, consecutively assigned card numbers of the user cards were required and no further authorization by the user was required to enable the vehicle charging. In this case, the payment system is a fixed part of these charging stations and is designed only to be used for charging vehicles. In the system, custom protocols for communication with the power supply and the network were defined. In comparison, the solution used at KIT is designed to be integrated into vending machines, cash desks and other application as generically as possible. Therefore, the system analyzed in this thesis is used in a lot more different types of devices and differs from the implementation at the charging stations by the used protocols, interfaces and operating procedures.

3. System Overview

The system is a closed loop system with prepaid cards. Each member of the site gets a Near-Field Communication (NFC) card issued by the KIT, which is used for identification on the site as well as for payment on vending machines and in cantines. As NFC cards, MIFARE Classic and, from August 2018 on, its successor DESFire by NXP-Semiconductors are used at the KIT since the Mifare Classic system has been proven insecure [Noh+08; KHG08; Sch+]. These cards contain a small amount of storage, divided into sectors. The data in the sectors is stored encrypted with individual keys, which allows different usage of the card by different applications. One of those applications is the cashless payment on the campus, provided by the Studierendenwerk Karlsruhe and Counter Solutions, the vendor of the payment system. In this chapter, we describe the cashless payment system used at KIT and how it is supposed to work.

3.1. Roles and Components

The payment system interacts with several entities of different types. Therefore, we define the key entities of the payment system in this section.

Roles

User The user uses a NFC card to buy goods on the site. Users are for example employees, students, researchers and guests on a university campus.

System Administrator The System Administrator has access to the administration interface of the backend and has the permission to access **Vending Reader**, **Unattended POS (UPOS)**, **Attended POS (APOS)** and **Card Terminal** (physically and remote).

Cashier A Cashier operates the **Attended POS (APOS)**.

Components

Revalue Device The revalue device (also called "Kiosk") reads the card of the **User**, accepts bank notes or electronic cash and revalues the account of the **User**. They are placed at strategically important places on the site, such as in the canteens.

Card Terminal The Card Terminal is operated as stand-alone device and reads the card of the **User**, verifies the card's integrity and displays recent financial transactions.

Point of Sale A Point of Sale (POS) is the location where a sale is completed. We can distinguish between two basic types of POSes:

Attended POS (APOS) An Attended Point of Sale is a **Point of Sale** operated by a person, like non-self-service cash desks in a canteen.

Unattended POS (UPOS) An unattended Point of Sale is a **Point of Sale** with no attendance, like a vending machine, which provides small goods (like snacks, coffee or beverage).

Vending Reader The vending reader reads the card of the **User**, executes transactions and confirms transactions to a **Point of Sale**. It acts as an adapter between the **Point of Sale** and the **Backend Server**.

Backend Server The backend server handles transactions and stores configuration for the **Vending Reader**, **Revalue Device** and **Card Terminal**.

3.2. Processes

Within the payment system, multiple processes are defined. In this section, we describe the four most important processes vending, auditing, maintenance and update.

3.2.1. Vending

For financial transactions, three different transaction types are defined. These types can also be found in the [NAMA-MDB] and [EVA-EPS] standards:

1. **Vend**
   Vending, e.g. when buying something
2. **Negative vend**
   When getting money in return for something, e.g. deposit for bottles or cups
3. **Revalue**
   When "recharging" a account with "real" money using banknotes, debit cards or any other system implemented on the site.

A basic vending machine setup is composed as follows: The vending reader is placed into the vending machine and connected via MDB (Multi-Drop Bus) [NAMA-MDB] to the vending machine controller (VMC) inside the vending machine. Now, the VMC waits for the vending reader to become ready. If ready, the VMC initializes the vending reader. After initialization, the vending machine is ready to serve.

If a customer wants to buy something, they first taps his NFC-Card at the vending reader. The reader does a lookup of the ID at the backend and gets data like the account balance of the user, if the card is valid (known and readable). Then the balance is displayed on the device and is sent to the machine.

Depending on the configuration, the product price may be defined in the vending machine or in the vending reader. The product prices may depend on a user specific price band (e.g. guests may pay more at the canteen than students). As soon as the user selects a product, the information about the selected product is sent to the vending reader, in case

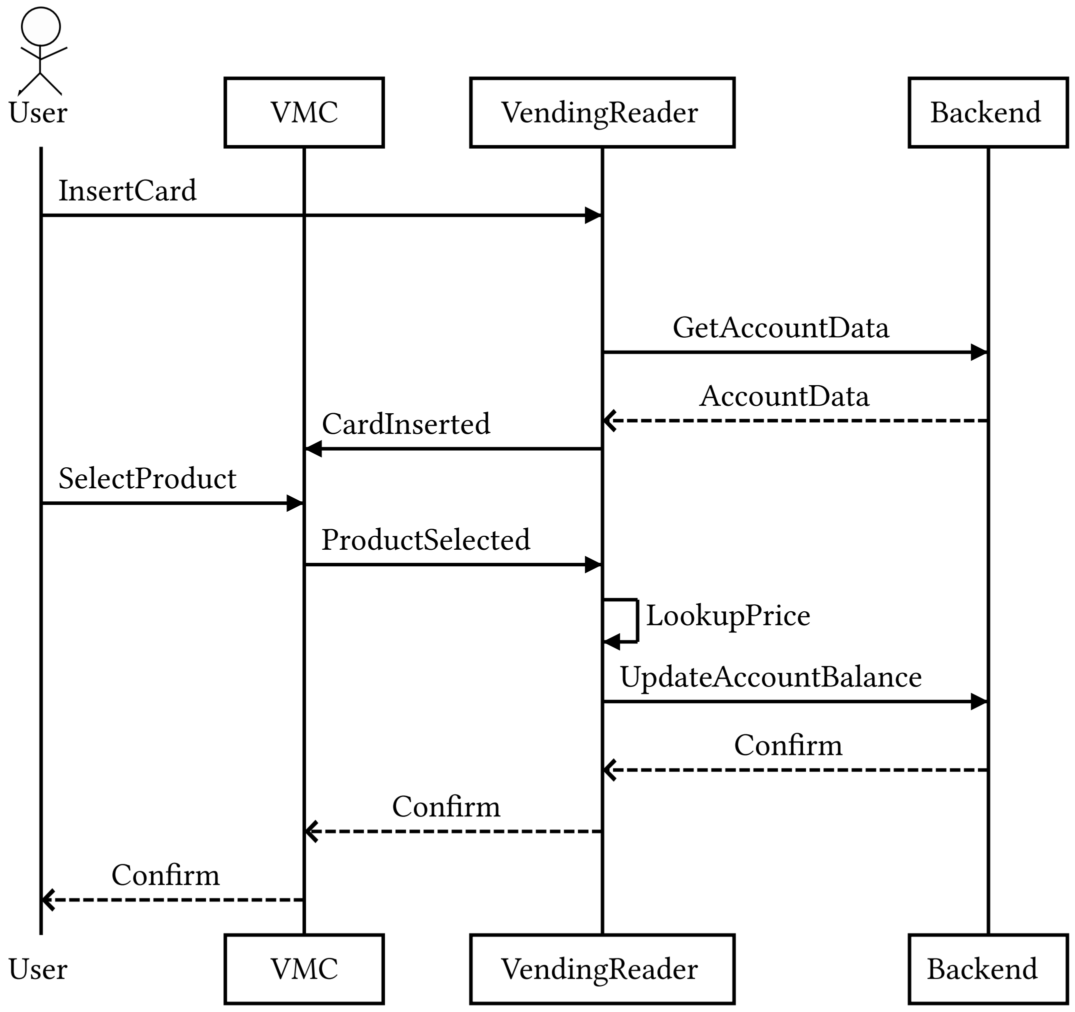


Figure 3.1.: Simplified flow of vending process with UPOS in a vending machine with prices defined by the vending reader

the vending reader defines the price. If the machine defines the price, the price is sent to the vending reader.

After this step, the vending reader updates the account balance and confirms the success of the vending process to the machine, which dispenses the product to the user. A simplified process of a vending machine with prices defined by the vending reader can be found in Figure 3.1. If configured, offline vending is possible: In case of a lost network connection, the vending reader does a lookup in the previously downloaded database of valid card IDs. If the card id is found in this database, no account balance is displayed on the vending reader and a dummy balance of 11,11€ is sent to the machine. To be able to be offline temporarily, the devices retrieve a list of valid card or wallet ids from the server (action 14) and store them into a local database (see section 3.4), as well as the stored offline transactions. Those transactions are sent to the server as soon as the device's connection gets recovered.

There are basically four types of vending setups, including those described above:

1. **Unattended POS (UPOS)** with prices defined by vending reader, e.g. Vending machines, canteen turnstile (see Figure 3.1)

2. **Unattended POS (UPOS)** with prices defined by POS, e.g. Revalue of third party accounts, washing machines (see Figure A.1)

3. **Attended POS (APOS)** connected with vending reader with prices defined by POS, e.g. Cash desks operated by personnel (see Figure A.2)

4. **Attended POS (APOS)** connected with simple reader with prices defined by POS, e.g. Cash desks operated by personnel

The communication with the backend is achieved via the APIX HTTP API (see subsection 3.5.2).

3.2.2. Audit

For audit purposes, both, the backend and the devices log the actions that have happened. The devices maintain a log file, which logs important internal function calls. APIX interactions are stored into a local database. The server maintains an account history which stores entries created by the `LogTransaction` APIX-Action (see subsection 3.5.2). The audit data from the devices can be retrieved with an USB storage device or via the USB slave port of the devices, by copying a log file (`cs_core.log`) and the local database which have to be decrypted using a special database decryption tool.

3.2.3. Maintenance and Update

Updates to the devices' configuration can be deployed by storing the local configuration database (see section 3.4) for each device in the backend. The devices retrieve them using the Heartbeat HTTP API (see subsection 3.5.1).

Updates for the core software are also possible via USB or via the vending readers' USB slave ports. If a USB storage contains an XML (eXtensible Markup Language) file called `cmd.xml`, the sequence of XML tags in the file is interpreted as sequence of commands. This allows an automated update procedure or retrieval of files. A list of commands can be found in Table A.1.

Additionally, a VNC (Virtual Network Computing) server is available on the vending readers. This access is just used for diagnostic purposes because the file upload and download feature of the server is disabled in the read only configuration of the VNC software. On revalue devices, TeamViewer is used. In TeamViewer, file uploads and downloads are enabled to do updates or maintenance.

As the vending readers' operating system is stored on a read only memory, updates of the operating system and software embedded into the system can only be done by reprogramming the hardware e.g. by using JTAG. Just the payment application software is stored on a writable section of the memory.

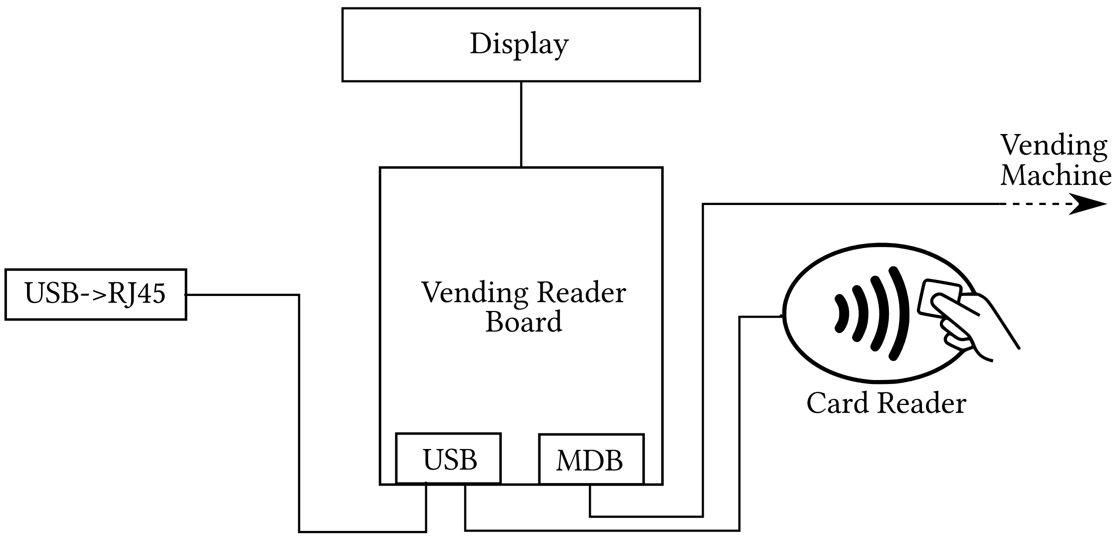


Figure 3.2.: Simplified block diagram of a vending reader

3.3. Device Hardware

At the point of sales, vending readers are connected to the cash desks or to the vending machine. Each vending reader has its own network connection through a RJ-45 wire. To send and receive information to the connected device (e.g. vending machine, cash desk, turnstile) the vending readers are equipped with either a MDB [NAMA-MDB] or RS485 (both serial protocols) interface. The vending reader itself contains a custom ARM microcomputer board with a native MDB interface and 3 USB host ports as well as one USB slave port. Network connectivity is gained via an USB to RJ-45 Ethernet adapter. The card reader unit at the front of the device is connected via USB. A block diagram of a vending reader can be found in Figure 3.2.

The Revalue Devices are powered by a full-size desktop computer with a touch screen and utilize their integrated USB and RJ-45 interfaces to communicate with the server and its peripherals such as bank note acceptors or debit card terminals. There is also an option to build and operate a Revalue Device using the hardware of the Vending Readers. This allows a cost efficient transition from the predecessor system (old casing and peripheral hardware can be kept like in Figure 6.1). This type of setup differs from a Vending Reader only by the *configuration* of the software and the connected peripheral hardware.

3.4. Device Software

On all devices, the software "`cs_core`" is used to provide the service. The software can provide different user interfaces for each use case like small character displays as well as big touchscreens.

The software is configured using three AES encrypted SQLite databases:

- `static.db`
  Used for configuration like the API URLs, price bands or configuration for peripheral devices

- `work.db`
  Used for working data, statistics and audit

- `advertising.db`
  Optional database for advertisements on the display

An XML based configuration file defines fallback and default values in case of a missing or corrupt configuration database and is evaluated at startup.

The Vending Readers are using Microsoft Windows CE6, the Revalue Devices are using Microsoft Windows 7 and the Card Terminals are using Microsoft Windows 10 as their operating system.

3.5. Backend Server Software

The backend server offers API endpoints over HTTPS for the different applications. It is located on a server on the internet, so the complete API interaction is carried out over a public network. In theory it is possible to provide the server locally, since the API URLs are configured in the device configuration, but we have not observed this in the test setup. There are two interfaces: The Heartbeat API and APIX.

3.5.1. Heartbeat API

The heartbeat HTTP API is located at `/HEARTBEAT/HANDLER.ASHX` and structures information with XML. It is used for configuration and monitoring of the software. It is called on startup and in a regular interval while the software is running.

On each call, the following attributes may be sent to the server (depending on configuration and mode of operation), grouped by the XML hierarchy:

- Platform: Processor architecture and serial number (also called "platform ID" in the heartbeat XML and "board number" on the display during startup). On non-ARM based devices (**Revalue Device** and **Card Terminal**), the serial number is equivalent to the MAC address of the primary network interface.

- Kernel version

- Local system time and date, as well as information of the local network configuration (IPv4 address and netmask, gateway)

- Version of `cs_core.exe`, outlet name, device name, device ID, group ID, site ID and hash of the `cs_core.exe`

- Hash of the `static.db` and `advertising.db`

Index	Key
0	Action (see Table 3.2)
1	Application ID
2	Business ID
3	Group ID
4	Site ID
5	Location ID
6	Device ID
7	Message sequence number

Table 3.1.: APIX base structure

Action	Description
1	`GetAccountData`
2	`UpdateAccountDetails` (vending, negative vending)
3	`Revalue`
4	`LogTransaction`
5	Observed but unknown
6	Observed but unknown
7-13	Unobserved and unknown
14	`GetAccounts` (all registered accounts in the system)

Table 3.2.: APIX Actions

The response attributes are similar to the attributes used in the request and may contain base64 encoded binary data of databasees or a new version of `cs_core.exe`. The `cs_core.exe` update feature does not seem to be used or implemented in the current software version.

3.5.2. APIX

The APIX HTTP API is the endpoint for everything related to the user accounts and the payment process. The information is structured by comma separated values (CSV). The first 8 values always have the same key in both, requests and responses (see Table 3.1). The remaining keys are action (index 0 in Table 3.1) specific.

4. Method

In the following sections, we will describe our analysis and risk management approaches as well as techniques used for the phases Risk Identification, Risk Analysis and Mitigation.

4.1. Risk Identification

The risk identification was done by investigating a dedicated test Vending Reader provided by the *Studierendenwerk Karlsruhe* (Figure 4.1), as well as a Revalue Device (like in Figure 7.1) operated against the vendor's staging server, running the payment application software `cs_core` version 5.7.0.4 T12. Most of the tests were done with the vending reader, as the revalue device was not permanently available for testing. There was no documentation or source code provided.

First, a port scan using the tool "nmap" was done to detect open TCP and UDP[1] network ports of the vending reader. Next, the network traffic of the devices was captured and analyzed using a personal computer as man-in-the-middle utilizing the package capture tool "Wireshark" and a self-implemented man-in-the-middle HTTPS proxy written in Java. The custom implementation of the proxy server was required to meet the low cipher specification by the vending reader and the handling of the SSLv2 handshake, because in current standard libraries like OpenSSL, old SSLv2 mechanisms and ciphers are removed or disabled. The proxying was achieved by spoofing the name server requests (DNS spoofing) and network address translation (NAT) to capture hard-coded IP addresses as illustrated in Figure 4.2: A DHCP server runs on the man-in-the-middle computer (MITM), which distributes addresses in the subnet of the MITM and announces itself as gateway and DNS server. This DNS server is configured to resolve all requested names to the IP of the MITM such that all requests should be addressed to the MITM. In case of a hard-coded IP address, foreign addresses are translated to the MITM's address in the firewall of the MITM (NAT). This configuration allows all HTTP requests to be redirected to the proxy server, which is bound to the MITM's address as well.

The payment application binary `cs_core.exe`, which runs on all devices of the system at KIT, was analyzed using the interactive disassembler "IDA Pro". The `cs_core.exe` was obtained from the vending reader via the slave USB port and the software "Microsoft Windows Mobile Device Center", which allows file transfers from Windows CE devices. The binary does not contain debug symbols, but the analysis was simplified due to the very verbose logging of the software. Each important function call is logged, so tracing the strings used for log output helped a lot identifying the used functions and APIs in the disassembled code. Therefore, a lot of the original function names can be recovered

[1]Because UDP is not connection based, ports have to be tested using known protocols. Custom or uncommon protocols may not be detected, if there is now answer to the probing packages.

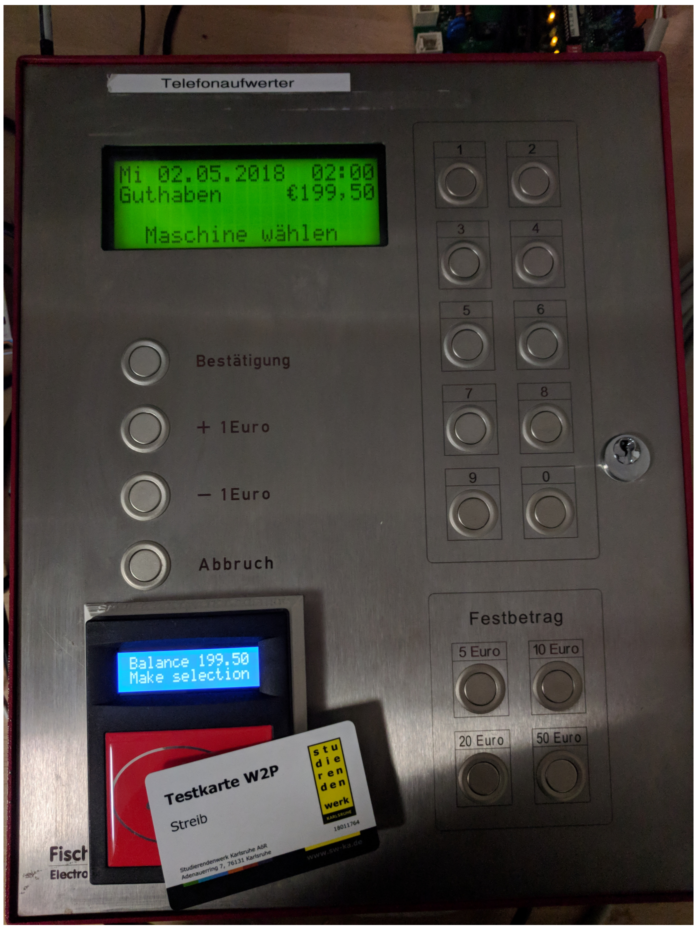


Figure 4.1.: Test Setup: A Vending Reader (lower left) embedded into a washing machine controller

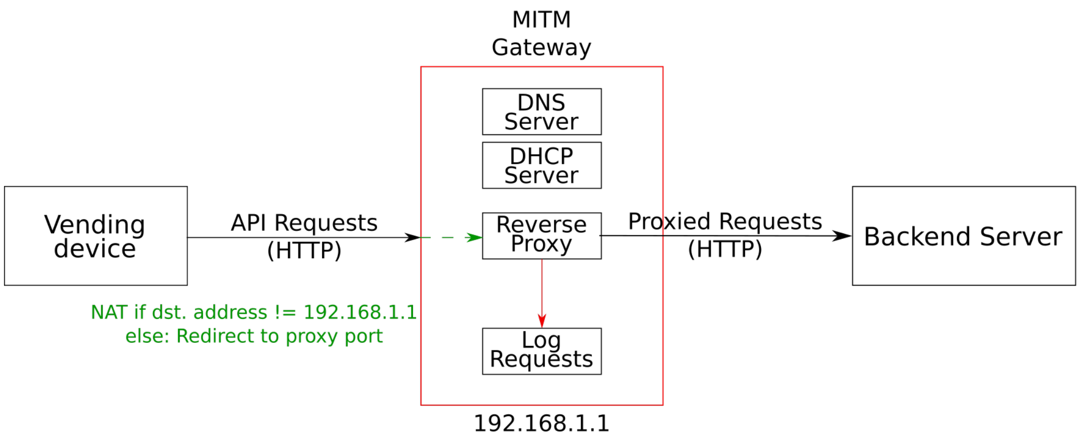
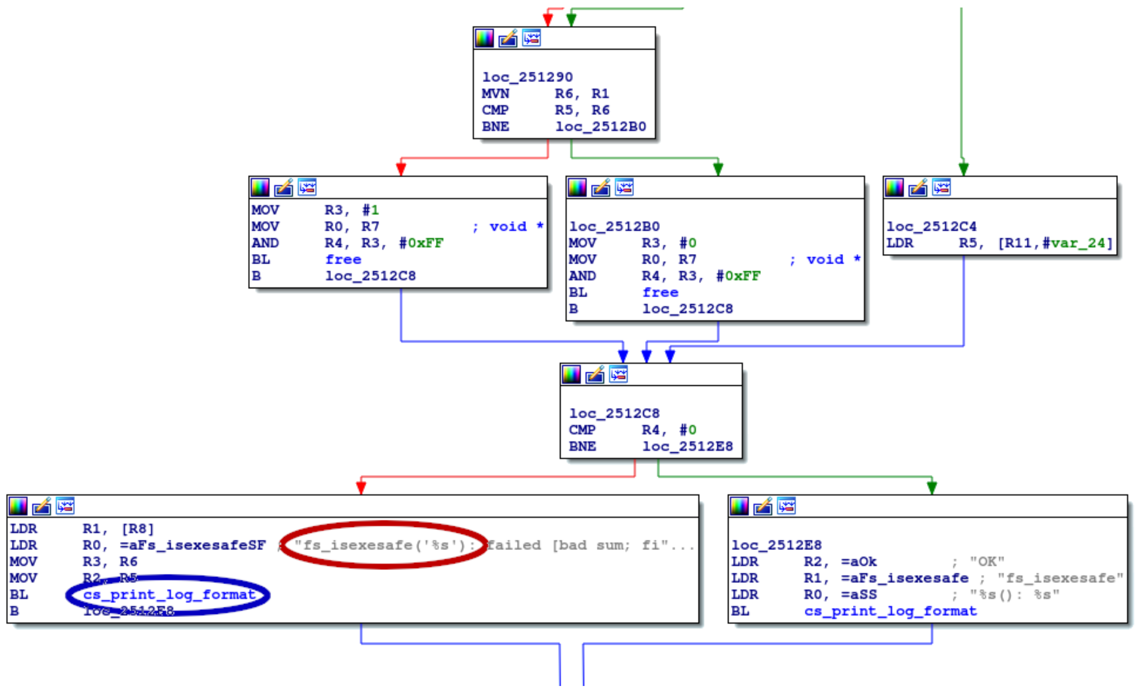


Figure 4.2.: Concept for a man-in-the-middle proxy setup



```
loc_251290
MVN      R6, R1
CMP      R5, R6
BNE      loc_2512B0
```

```
MOV      R3, #1
MOV      R0, R7                 ; void *
AND      R4, R3, #0xFF
BL       free
B        loc_2512C8
```

```
loc_2512B0
MOV      R3, #0
MOV      R0, R7                 ; void *
AND      R4, R3, #0xFF
BL       free
B        loc_2512C8
```

```
loc_2512C4
LDR      R5, [R11,#var_24]
```

```
loc_2512C8
CMP      R4, #0
BNE      loc_2512E8
```

```
LDR      R1, [R8]
LDR      R0, =aFs_isexesafeSF ; "fs_isexesafe('%s'): failed [bad sum; fi"...
MOV      R3, R6
MOV      R2, R5
BL       cs_print_log_format
B        loc_2512E8
```

```
loc_2512E8
LDR      R2, =aOk            ; "OK"
LDR      R1, =aFs_isexesafe ; "fs_isexesafe"
LDR      R0, =aSS            ; "%s(): %s"
BL       cs_print_log_format
```

Figure 4.3.: An excerpt from a IDA call graph of a subroutine. The subroutine can be identified by the log output function calls ("`cs_print_log_format`", marked blue). In this case it is "`fs_isexesafe()`" (marked red).

and mapped to the subroutines. An example of a reconstruction of a subroutine name is shown in Figure 4.3: In the subroutine, the result of the subroutine is written into a log file. Therefore, a string starting with "fs_isexesafe('%s'):" is loaded (marked in red in the figure). Then the subroutine, previously identified in the analysis, which we named "cs_print_log_format" is called (blue marking) to print the previous format string. By the format string used, we can identify the subroutine we are looking at as "fs_isexesafe()".

The analysis of the used database encryption in the software required debugging of x86 libraries used by the database decrypt tool. In theory, it is also possible to debug the encryption mechanism in `cs_core.exe`, but this would require to debug an ARM Windows CE6 process, which is complicated because of unavailable, old or unusable debug tools. Emulating the device was also rendered to be hard because of the expensive software tools required for creating a Windows CE6 image.

The decryption tool is implemented using .NET, which is compiled to bytecode and executed in a runtime environment. Disassembling the tool revealed a very straight forward implementation: The tool loads the shared library `DBCreatorClasses.dll`, which exports utility functions for SQLite interaction. After a file open dialog is shown to let the user pick an encrypted database file, the `DBCreatorClasses.dll` function `Decrypt(string fileName)` is called with the chosen file path as `fileName` parameter. This function then reads the database file and encodes it using Base64. This string is used, to call the `cs_dbpack.dll` shared function `unpack([out] char[] out_bin, unsigned int32 out_sz, char[] in_b64)` with `in_b64` as the Base64 encoded string, the result buffer `out_bin` and the uint32 `out_sz`, storing the size of the resulting data. This function was the main debugging target to understand the implementation of the database encryption. Because of problems when attaching a debugger directly to the .NET process of the decryption tool to analyze the x86 library, we wrapped the required library call `unpack()` into an own, minimalist binary. The library call in the custom binary fails for an unknown reason, but during the process of the call, the database gets decrypted into a temporary file, which was sufficient for debugging the decryption process. The code of this application can be found in Listing A.1. This binary was successfully debugged using IDA's remote debugger feature with IDA's remote debugging server running on a Windows 10 virtual machine.

4.2. Risk Analysis

As described in [ISO27005], we analyzed the risk of an attack by identifying the consequences (impact) and the likelihood of an attack utilizing the identified risks. The impact was estimated by the data obtained or modified. To evaluate the likelihood of an attack, we defined different attacker types with their capabilities as well as the likelihood of an attack by such an attacker. If an attack scenario matches the capabilities of one or multiple attackers, the highest likelihood out of the attackers with the required capabilities to perform the attack is used as scenario likelihood. We evaluated the risk in a qualitative fashion. Therefore, we defined a consequence probability matrix, which assigns all combinations of defined likelihoods and impacts to a risk level.

4.3. Mitigation

The proposed mitigations are derived from industry best practices like the weakness mitigations proposed by the MITRE CWE list[2], OWASP[3] and standards like the *BSI Grundschutz*. The MITRE CWE list is a community driven list of common software weakness types. The weakness types are organized in an abstract class tree such that detailed weakness types are member of more abstract classes of types. The types are containing a detailed description of the weakness, common consequences, a common exploitation likelihood, in which software development phase the weakness occurs (such as "Architecture and Design"), some examples and common mitigations.

The Open Web Application Security Project (OWASP) is a non-profit organisation, which publishes documents and tools, to improve the security of web applications and services. OWASP is most commonly known for its "Top 10" project of most critical vulnerability types out of the over 700 weakness types currently defined in the CWE database. The list is regularly updated and contains the most common and critical CWEs in applications with additional comments and background information.

[2]`https://cwe.mitre.org`
[3]`https://www.owasp.org`

5. Risk Identification

The first step of the risk assessment process is the identification of possible risks and weaknesses. In the following, we documented all weaknesses we found by analyzing the product as in section 4.1 described. The following list may not contain *all* weaknesses of the product because of the complexity of the software. Furthermore, there was no access to the server software. To classify the weakness types of *software issues*, we used the common weakness enumeration list (CWE), which applies to software only, not e.g. the composition of a system.

5.1. Outdated Operating Systems

Affected components	Vending Reader, Revalue Device

According to Microsoft, the extended support of Windows CE 6 (used for vending readers) ended on 2018-04-10 [Mica]. The mainstream support of Windows 7 (Service Pack 1, used for revalue devices) ended in January 2015, while the extended support will continue until the beginning of 2020 [Micb]. This means, that there are no operating system updates and security patches anymore from Microsoft for the vending readers, and only less than two more years of updates for the revalue devices.

5.2. Usage of an Operating System with Desktop Environment

Affected components	Revalue Device, Card Terminal

The **Revalue Device** software is operated in the desktop environment of Windows 7, the **Card Terminal** software in the desktop environment of Windows 10. In case of a crash of the software, users may get access to critical functions of the operating system as well as data stored on the device.

5.3. Remote Management

Affected components	Vending Reader

A port scan using the port scan tool "nmap" reveals, that there is an open Virtual Network Computing (VNC) server port and a RealVNC web service port accessible. The page title of the web service indicates, that the used version of RealVNC is 4.0, which is vulnerable to CVE-2006-2369 [Int06]. The vulnerability allows for an remote unauthenticated attacker to gain desktop access to the target system, by sending a null authentication. If exploited during the startup phase of the vending reader, full access[1] to the Windows CE desktop is possible.

5.4. Missing Server Certificate Verification

Weakness type: [CWE-295]	Improper Certificate Validation
Weakness type: [CWE-300]	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')
CVE-ID	CVE-2018-18292
Affected components	Vending reader, Revalue device

For the given test devices it was possible to intercept all network connections (HTTPS) by using a man in the middle proxy without injecting an authoritative certificate into the target systems. This indicates, that there is no verification of the server's certificate on the client side.

5.5. Usage of Weak Cipher Algorithms

Weakness type: [CWE-327]	Use of a Broken or Risky Cryptographic Algorithm
Affected components	Vending Reader

By capturing the Secure Socket Layer version 2 (SSLv2) protocol handshake of a vending reader to the server, the supported cipher algorithms can be inspected (SSLv2 Client Hello). All of them are considered weak and some of them have practical attacks. A cipher is considered "weak", if the block size or key length is below the minimum recommended size [BSI18a; NIST15] or there exists an attack, which significantly reduces the cryptographic complexity. None of the ciphers are perfect forward secrecy ciphers (PFS-ciphers). A list of the client's supported ciphers and the cipher overlap with the backend server is shown

Algorithm	**Weak?**	**Practical attacks?**
TLS_RSA_WITH_RC4_128_MD5	Yes [FMS01]	Yes [VP15]
TLS_RSA_WITH_RC4_128_SHA	Yes [FMS01]	Yes [VP15]
TLS_RSA_WITH_3DES_EDE_CBC_SHA	Yes (block size)	Yes, requires much data [BL16a]
SSL2_RC4_128_WITH_MD5	Yes [FMS01]	Yes [VP15]
SSL2_DES_192_EDE3_CBC_WITH_MD5	Yes (block size)	Yes [Gil98]
SSL2_RC2_128_CBC_WITH_MD5	Yes [KSW97]	Yes, requires much data [BL16a]
TLS_RSA_WITH_DES_CBC_SHA	Yes (block size)	Yes [Gil98]
SSL2_DES_64_CBC_WITH_MD5	Yes (block size)	Yes [Gil98]
TLS_RSA_EXPORT1024_WITH_RC4_56_SHA	Yes [FMS01]	Yes [VP15]
TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA	Yes (block size)	Yes [Gil98]
TLS_RSA_EXPORT_WITH_RC4_40_MD5	Yes [FMS01]	Yes [VP15]
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	Yes [KSW97]	Yes, requires much data [BL16a]
SSL2_RC4_128_EXPORT40_WITH_MD5	Yes [FMS01]	Yes [VP15]
SSL2_RC2_128_CBC_EXPORT40_WITH_MD5	Yes [KSW97]	Yes, requires much data [BL16a]

Table 5.1.: Supported SSL and TLS Ciphers. Highlighted: Cipher overlap with the backend server

in Table 5.1. If there is a practical attack on 3DES or the SSLv2 handshake[2] to manipulate the cipher negotiation within the given time constraints to use a weak cipher like RC4, it is possible to decrypt the TLS traffic.

5.6. Weak Binary Integrity Check

Weakness type: [CWE-328]	Reversible One-Way Hash
Affected components	Vending Reader

In the software, a function called "`fs_isexesafe()`" is called at startup. It calculates the cyclic redundancy check (CRC) sum of the binary located at `\Intel\cs_core.exe` and

[1]It is possible to see the desktop and to generate mouse and keyboard inputs.

[2]Currently, SLOTH [BL16b] requires a lot of computation power and can therefore not considered as "practical".

compares it to a static sum saved in the software. If the check fails, the software refuses to continue the startup process and quits.

The name of the function indicates, that the CRC sum is used as a counter measure against unauthorized modification of the `cs_core` binary. This is not a sufficient method to achieve this goal, because

1. CRC is an error correcting code - it is not designed to be an cryptographic safe method to check the integrity of data, since it is easy to extend a modified binary in that way, that it produces the same CRC sum as the "correct" binary (no collision resistance).

2. The path of the checked binary is static. If someone modifies the binary, this path could also be changed in the modified binary. Alternatively, a correct version of the file have to placed to the hard-coded location while running a modified binary in another location.

3. When modifying the binary, the CRC check can be simply bypassed by patching the instructions that return the result of the check.

5.7. Broken AES Implementation

Weakness type: [CWE-325]	Missing Required Cryptographic Step
CVE-ID	CVE-2018-18295
Affected components	Vending Reader, Revalue Device

To understand the risk, introduced by the Advanced Encryption Standard (AES) implementation in this subsection, we roughly describe the AES process first, before we analyze the actual implementation.

5.7.1. AES Overview

AES as described in [FIPS01] is a symmetric block cipher. Each block of 128 bits (represented as 4×4 byte matrix) is encrypted by combining the block with a round key using exclusive or (XOR, $\oplus$). The operation of applying the corresponding round key to the current intermediate result (state) is called `AddRoundKey`. The round keys are extracted from the encryption key by expanding it using the key expansion algorithm as described later. To gain diffusion, three additional invertible transformations are applied before each `AddRoundKey` operation:

1. `SubBytes`
   Substitute bytes (SubBytes) replaces all bytes of the current state with the corresponding one in a fixed lookup table (*S-box*).

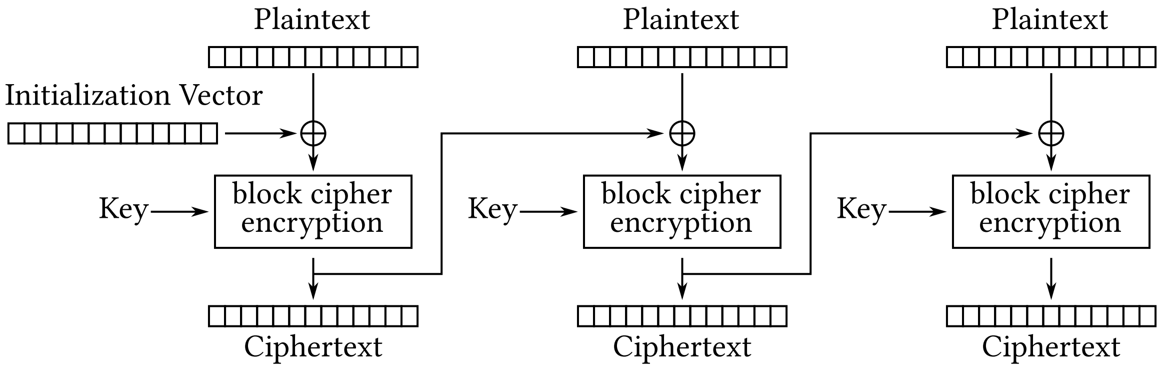


Figure 5.1.: Cipher Block Chaining (CBC) mode of operation [Whi13]

2. `ShiftRows`
   In this operation, each row of the state matrix gets shifted left cyclically by its row index (the first row is shifted zero bytes to the left, the second by one byte, ...).

3. `MixColumns`
   The MixColumns step combines each four bytes columns to another four byte columns so each input byte affects all output bytes. This is achieved using a linear transformation with a fixed matrix.

In the initial round, just `AddRoundKey` is applied onto the plain text block:

$$\text{state}_0 = \text{plaintext\_block} \oplus \text{r\_key}_0 \tag{5.1}$$

After the first round, the diffusion steps followed by `AddRoundKey` are repeated multiple times on each block, where the number of rounds (N) depend on the encryption key length:

$$\begin{aligned} \text{state}_n =& f_{\text{MixColumns}}(f_{\text{ShiftRows}}(f_{\text{SubBytes}}(\text{state}_{n-1}))) \oplus \text{r\_key}_n \\ & \text{with } n \text{ initialized with } N-1 \end{aligned} \tag{5.2}$$

In the final round, only `SubBytes`, `ShiftRows` and `AddRoundKey` are performed:

$$\text{ciphertext} = f_{\text{ShiftRows}}(f_{\text{SubBytes}}(\text{state})) \oplus \text{r\_key}_N \tag{5.3}$$

To be able to encrypt more than one block without leaking information about patterns in the plain text, different block cipher modes of operation are defined. One of them is the *Cipher Block Chaining* mode (CBC): In this mode, the encryption result of the previous block and the plain text of the current block are combined using XOR and encrypted to build the resulting cipher text of each block as illustrated in Figure 5.1. For encryption of the first block, a randomly chosen initialization vector (IV) is used instead of the previous (empty) block result. The IV is therefore also required for decryption. This allows diffusion beyond the margin of a block; blocks with equal plain text are not encrypted to the same cipher text.

The key expansion (also called "Rijandael Key Schedule") utilizes the following four Operations:

- Rotate
  All bytes are rotated cyclically to the left
- Round constant ("`rcon`(i)")
  Exponentiation of 2 to a value i
- S-box
  Byte substitution using a fixed lookup table (*S-box*) as in `SubBytes` above.
- Key schedule round
  The 32 bit input gets rotated by 8 to the left and the S-box is applied.
  Next, the first (leftmost) byte is combined with `rcon`(i) using XOR, where i is the round number of the key schedule iteration.

Depending on the key size and block size, a required key length has to be reached: For a block size of 128 bit and a 256 bit key, the key has to be expanded to a total length of 240 bytes. During each iteration until the required size is reached, i gets incremented by one, initialized with the value 1. The intermediate results are stored into a state variable, which is initialized with the (unexpanded) encryption key.

Each round starts by using the least 4 bytes of the expanded key and applying the key schedule round on it. The result is combined with the 4-byte block n bytes before the new expanded key part. n depends on the key size – for 256 bits $n = 32$. At this point, i gets incremented by one. The next twelve bytes of the expanded key are generated by applying XOR to the last 4 bytes to the four-byte block n bytes before the current expanded key part three times. If the initial key size is 256 bits, the next four bytes are obtained by applying the S-box to the last four bytes and XOR with the four bytes block n bytes before the current expanded key part.

5.7.2. AES Implementation in the Payment System

The AES encryption with CBC is used to encrypt the configuration SQLite databases `work.db` and `static.db`. Due to an implementation error in the key expansion, the array designated for the expanded keys stays `0`, except the last 4 double words ($4 \cdot 32$bit). In each expansion round, the last 4 dwords are overwritten until there is only the last round key at end of the array, while the rest remains zero. This leads to the following reduction of the AES block decryption process:

$$\begin{aligned} f_{\text{transform}}(\text{ciphertext}) &= f^{-1}_{\text{SubBytes}}(f^{-1}_{\text{ShiftRows}}(f^{-1}_{\text{MixColumns}}(\text{ciphertext} \oplus \text{r\_key}_n)))\forall n \in [1, N-1] \\ &\overset{\text{r\_key}_n = 0 \forall n}{=} (f^{-1}_{\text{SubBytes}} \circ f^{-1}_{\text{ShiftRows}} \circ f^{-1}_{\text{MixColumns}})^N(\text{text}) \end{aligned} \tag{5.4}$$

$$\text{cleartext} = f^{-1}_{\text{SubBytes}}(f^{-1}_{\text{ShiftRows}}(f_{\text{transform}}(\text{ciphertext}))) \oplus \text{r\_key}_0 \tag{5.5}$$

Due to the known basic structure of the database file, r_key_0 can be calculated by picking a block with known content (for example `0`-byte blocks in SQLite database files) and its predecessor to resolve the CBC mode of operation:

$$\begin{aligned} \text{r\_key}_0 = & f^{-1}_{\text{SubBytes}}(f^{-1}_{\text{ShiftRows}}(f_{\text{transform}}(\text{known\_ciphertext}))) \\ & \oplus \text{predecessor} \oplus \text{known\_plaintext} \end{aligned} \tag{5.6}$$

After applying this step, the data is decrypted except the first block of each `0x400` bytes block because of the used configuration of the CBC mode of operation: The CBC is reinitialized every `0x400` bytes with a new IV (in total five) which are cycled in the file. This allows incrementally updating the file without re-encrypting the whole file. Without the IVs, the file is not a valid SQLite database, but most of the information is readable in an editor. The IVs can be either acquired from the software, or calculated by inferring the content behind the first block of each `0x400` bytes block, based on the SQLite structure, whereas the first IV can be trivially inferred because of the known SQLite file magic string. Afterwards, the file can be opened in a SQLite browser.

5.8. Hard-Coded Database Encryption Key and Weak Integrity Check

Weakness type: [CWE-321]	Use of Hard-coded Cryptographic Key
Weakness type: [CWE-649]	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
CVE-ID	CVE-2018-18293
Affected components	Vending Reader, Revalue Device

We have discovered, that the same AES key is used for all databases for all devices at least since the introduction of the predecessor of the current system (old configuration databases can also be decrypted using the method described in section 5.7). Further analysis of the software revealed, that the encryption key is hard-coded into the software. The encryption of the database thereby degrades to a form of complex encoding rather than a security measure against unauthorized access of the databases. Additionally, encryption is not a security measure against unauthorized modification. In the software, the plain text just has to be a SQLite database. No further integrity checks are performed.

It is possible to obtain the key and the IVs from a `cs_core` binary or the vendor's database decrypt tool by debugging and analyzing them in an interactive disassembler and debugger or using the method described in section 5.7.

5.9. Automatic Execution of Script from Removable USB Storage

Weakness type: [CWE-284]	Incorrect Access Control
CVE-ID	CVE-2018-18294
Affected components	Vending reader, probably: Revalue Device (not tested due to limited device availability, but code fragments were found in the `cs_corex86.exe` binary)

When plugging a FAT32 formatted USB storage containing a file called "`cmd.xml`" (see subsection 3.2.3) to one of the USB host ports, the file gets executed automatically without any check of authorization or authenticity. Arbitrary code execution and leakage of all data on the device and configuration to the USB storage is possible as well as reconfiguration of the device.

5.10. Unauthenticated Access to Configuration

Weakness type: [CWE-306]	Missing Authentication for Critical Function
Affected components	Vending Reader, Revalue Device, Backend

It is possible to craft a malicious request to the heartbeat API, which retrieves a device's encrypted `static.db` just by using the device's board number ("platform ID"). This ID is shown on the display during startup of the devices as demonstrated in Figure 5.2 and is therefore not a secret. Additionally, the board number is equivalent to the MAC address of the primary network interface for the non-ARM based devices (**Revalue Device** and **Card Terminal**). Thereby it is possible to obtain the board number of all those devices in their broadcast domain via the address resolution protocol (ARP).

5.11. Missing Transaction Authorization by User

Weakness type: [CWE-653]	Insufficient Compartmentalization
Weakness type: [CWE-862]	Missing Authorization
Affected components	Vending Reader, Revalue Device, Backend

When making an APIX request, those requests are authenticated by application ID, business ID, group ID, site ID, location ID and device ID. If a financial transaction is executed, there is no possibility to prove, that the card owner (or at least users' card) explicitly authorized a transaction (non-repudiation). Valid transactions can be created without any knowledge

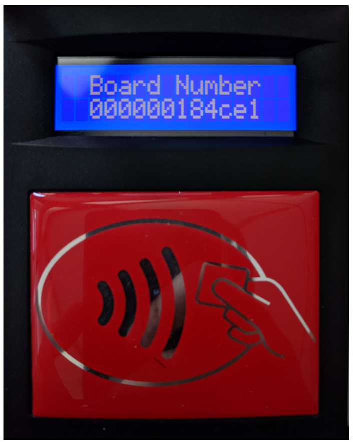


Figure 5.2.: Displayed board number on startup of a vending reader

about an user secret – an attacker only needs to know the IDs mentioned above to authenticate itself as e.g. a vending reader against the server.

5.12. Missing Replay Protection

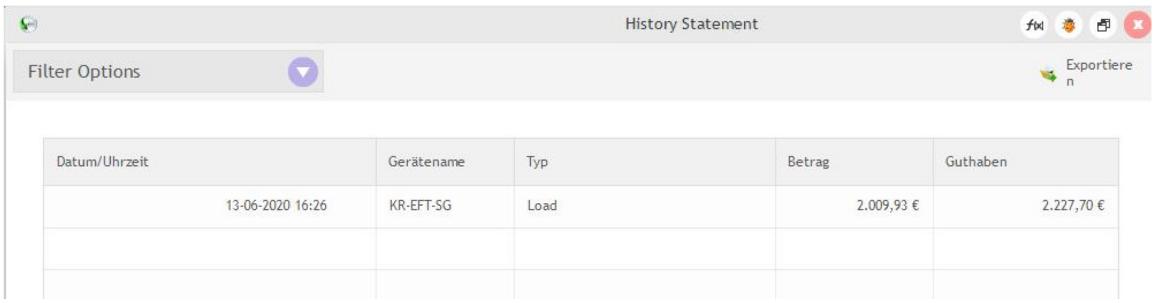
Weakness type: [CWE-294]	Authentication Bypass by Capture-replay
Affected components	Vending Reader, Revalue Device, Backend

Even if the transaction gets authorized by the correct card, the internal protocol still has a lack of a replay protection. If an attacker gains access to a raw financial transaction APIX request, the transaction can be replayed and remains valid – users' previous authorization gets bypassed.

5.13. Lack of Basic Sanity Checks for Transaction History

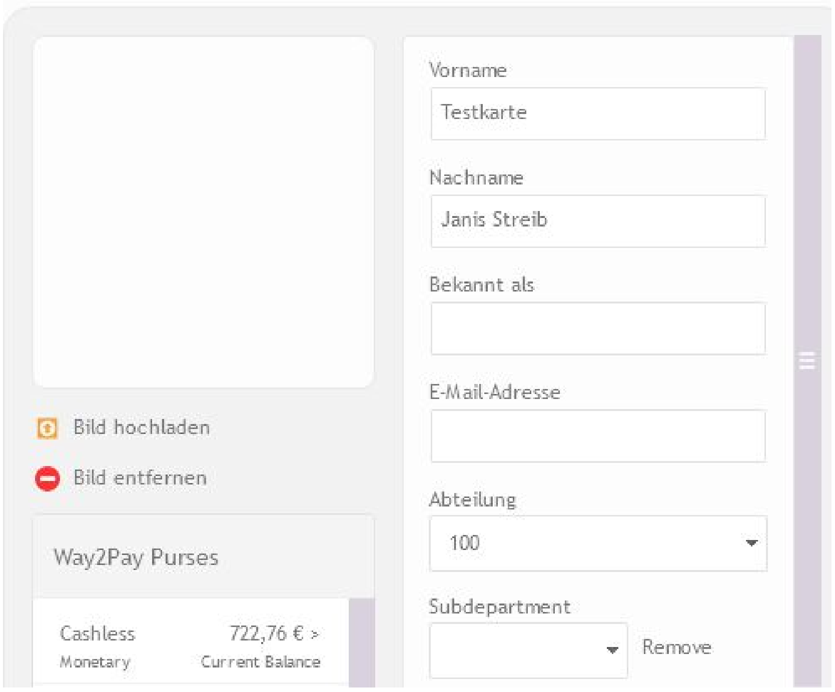
Weakness type: [CWE-223]	Omission of Security-relevant Information
Affected components	Backend

When submitting a financial transaction without sending an audit message, no history log entry is created on the server. Therefore, financial transactions sent without audit log are hard to trace. On the other hand it is possible to create transaction logs without an actual transaction. There are no referential integrity checks between financial transactions and the history log. It is even possible to date back entries or create entries in the future as demonstrated in Figure 5.3.



Datum/Uhrzeit	Gerätename	Typ	Betrag	Guthaben
13-06-2020 16:26	KR-EFT-SG	Load	2.009,93 €	2.227,70 €

(a) Malicious history entry in the Way2Pay manager web interface



Vorname	Testkarte
Nachname	Janis Streib
Bekannt als	
E-Mail-Adresse	
Abteilung	100
Subdepartment	

Cashless Monetary 722,76 € > Current Balance

(b) Real account balance in the Way2Pay manager web interface

Figure 5.3.: Excerpts from the management web interface, which shows the missing referential integrity and sanity check

6. Risk Analysis

In this chapter the previously in chapter 5 identified risks are analyzed, by evaluating the the composition of vulnerabilities and the probability of occurrence.

6.1. Method

We will use a qualitative fashion of risk analysis to assess the risk, since we do not have enough statistical data to evaluate an attack quantitatively. Therefore we first need to define possible attacker types with their capabilities and the likelihood of an attack of each type. Then, we define the target data, to evaluate the impact of an attack. Finally, the likelihood and the impact are combined to a risk level in the consequence probability matrix. On all these steps, we assume the worst case scenario.

6.1.1. Attacker Model

To evaluate the probability of an incident, we define different attackers and the likelihood of an attack by the attacker as well as their capabilities.

Curious Student A curious student has advanced technical knowledge and is thereby for example capable to create an analysis setup like a man-in-the-middle proxy. A curious student is not willing to cause any physical, financial or image damage and just wants to gain knowledge for personal interest. This attacker type can be considered to be quite likely on a technical university campus like the KIT.
Likelihood: Likely

Script Kiddie A script kiddie has only basic technical skill and is not able to do own analysis and sophisticated attacks. As the name implies, a script kiddie usually uses already implemented exploits ("scripts") to attack a system so a script kiddie depends on previous work by attackers with higher skill (e.g. by using tools like Metasploit[1]). There is a low to medium criminal motivation, so a financial or image damage is possible. This attacker type is a more likely, preliminary attacker type of the **Criminal with Advanced Technical Skill**.
Likelihood: Possible

Criminal with Advanced Technical Skill A skilled criminal is capable of doing a full analysis of the system (hardware and software) and is motivated to use that knowledge to create a damage for own enrichment. For gaining information, physical force may be applied. According to the *Studierendenwerk Karlsruhe*, the count of damaged devices

[1] `https://www.metasploit.com/`

containing cash money is very low – the count of damaged devices not containing cash money is near zero. Therefore, the Likelihood for this (technically skilled) kind of attacker is unlikely.
Likelihood: Unlikely

Intelligence Agency Intelligence Agencies are capable of very advanced attacks with potentially publicly unknown methods and knowledge of weaknesses as well as a lot of resources. It can be considered to be very rare, that such organisations will attack a target like the payment system analyzed in this work.
Likelihood: Rare

6.1.2. Target Data

In the following we lists data, which may be extracted by an attacker. If any of this data can be obtained by an attack, the impact is at least *medium*. If the data can be altered and injected into an system, or can be used to alter other data, the impact is *high*.

Device Authentication Data Data used for authentication of a device against APIX (application ID, business ID, group ID, site ID, location ID, device ID)

User Account Data Data used for identifying the user (card identifier, internal account ID)

User Account History User transactions over time

Configuration Data Data affecting the behaviour of a device (e.g. `static.db`)

Configuration Secret Secret for device configuration

6.1.3. Consequence Probability Matrix

We define the consequence probability matrix in Table 6.1 to evaluate the resulting risk created by a weakness or vulnerability. Each likelihood is represented by an attacker. To meet the way more unlikely attacker type **Intelligence Agency**, the resulting risk for a rare likelihood is always "Very Low".

Impact / Likelihood	Negligible	Marginal	Critical
Rare	Very Low Risk	Very Low Risk	Very Low Risk
Unlikely	Low Risk	Low Risk	Medium Risk
Possible	Low Risk	Medium Risk	High Risk
Likely	Medium Risk	High Risk	High Risk

Table 6.1.: Consequence probability matrix

6.2. Weaknesses

Weaknesses are implementation or design problems, which do *not directly* lead to attacks. An overview of the analyzed weaknesses can be found in Table 6.2. In the following subsections, we will analyze the risk level score of attacks, indirectly caused by the weaknesses.

6.2.1. Outdated Operating Systems

Outdated operating systems are not receiving (security) updates anymore (End Of Support, EOS). An attacker, capable of obtaining zero day exploits (publicly unknown attacks), may be able to attack the system (e.g. intelligence agencies). Publicly known attacks with existing exploits published after the EOS of an operating system may also lead to an impact. The probability of this attack increases by the time of usage after EOS: The longer the operating system is used after EOS, the more publicly known attacks are available.

Likelihood: Unlikely, **Impact**: Critical, **Risk**: Medium

6.2.2. Usage of an Operating System with Desktop Environment

In case of a software crash, it is possible for the user to interact with the desktop environment of the underlying operating system. An attacker is thereby able to extract data, install malware or cause a denial of service (DOS). An attack already happened after the introduction of the system in August: A **Card Terminal** in a canteen had to be shut down, after students opened a web browser and played a video full screen (see Figure A.3). The software crashed for a currently unknown reason.

Likelihood: Likely, **Impact**: Critical, **Risk**: High

6.2.3. Usage of Weak Cipher Algorithms

Due to the weak cipher algorithms available at the client (see section 5.5), there is only one cipher with no known practical attack yet: 3DES. As it is very unlikely, to capture the required amount of data in the lifetime of a the device (about 780GB [BL16a]) it may only be possible for an *highly* equipped attacker to decrypt captured packages using brute force. A decryption of the traffic allows the extraction of the **Device Authentication Data**, **User Account Data**, **User Account History** and the encrypted **Configuration Data**. Depending of the speed, in which the data gets decrypted, it may also be possible to alter this data during transport.

Likelihood: Rare, **Impact**: Critical, **Risk**: Very Low

6.2.4. Weak Binary Integrity Check

The binary integrity check used can be easily bypassed, is weak and does not work in general in the way it is implemented. This weakness does not create a real attack surface.

Likelihood: Possible, **Impact**: Negligible, **Risk**: Low

6.2.5. Hard-Coded Database Encryption Key and Weak Integrity Check

A hard-coded encryption key for the configuration databases may lead to access to the **Device Authentication Data**, if someone obtains the **Configuration Secret**. This may be possible by exploiting a vulnerability, requesting a sample device, stealing a device, breaking into a device or acquiring a disposed, not properly wiped device. Once obtained, all device configurations worldwide can be decrypted and therefore all **Device Authentication Data**. Furthermore, a **System Administrator** of one site can decrypt databases from other sites.

Likelihood: Likely, **Impact**: Critical, **Risk**: High

6.2.6. Lack of Basic Sanity Checks for Transaction History

The lack of sanity checks for audit logs makes it hard to detect properly crafted attacks as a **System Administrator**.

Likelihood: Unlikely, **Impact**: Marginal, **Risk**: Low

6.3. Vulnerabilities

A vulnerability is a weakness or a group of weaknesses previously identified in chapter 5, which can *directly* be used to perform an attack. An overview of the analyzed vulnerabilities can be found in Table 6.3. In the following subsections, we will analyze the risk of attacks, caused directly by the identified vulnerabilities.

6.3.1. Remote Management

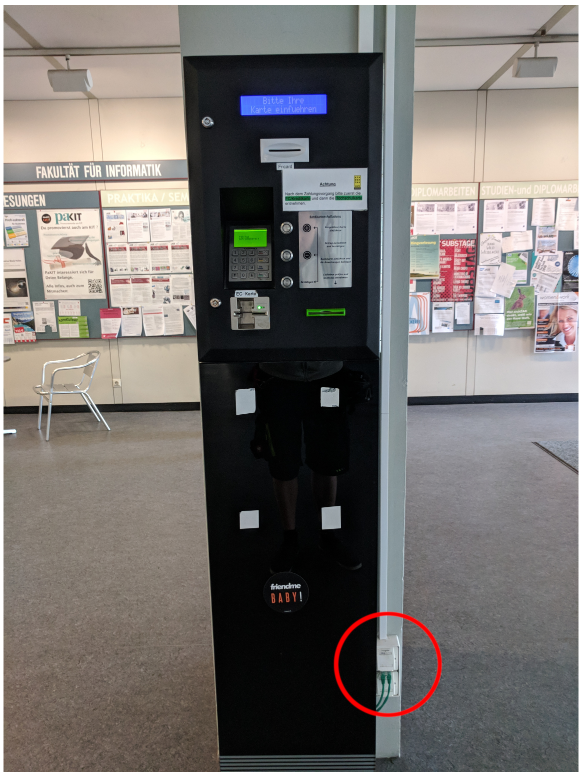
Due to the implementation error in the software used on the devices, a denial of service attack is possible, as well as extraction of the audit log file, which may contain sensitive private user data.

Likelihood: Likely, **Impact**: Critical, **Risk**: High

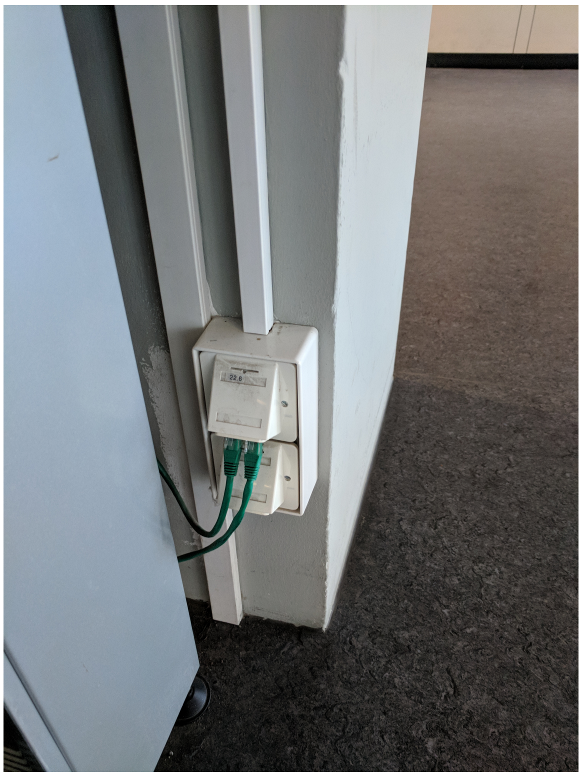
6.3.2. Missing Server Certificate Verification

As man-in-the-middle it is possible to capture each APIX and Heartbeat request as clear text due to the lack of server certificate verification (section 5.4). Thereby, **Device Authentication Data**, **User Account Data** and **Configuration Data** can be obtained. This allows enough data like user card IDs, device secrets and general API behaviour for advanced attacks to be collected. By analyzing the captured requests (e.g. pattern matching with prices) it is not necessary to have an official API documentation to interact with the API.

Furthermore, a man-in-the-middle can also modify data. It is even possible to modify the encrypted **Configuration Data** of the device, which is sent via the Heartbeat API. Because of weak integrity checks (the clear text only has to be a SQLite database, see section 5.8), modifications of the configuration may not be detected.



(a) Old revalue device.



(b) The network connection is freely accessible.

Figure 6.1.: A typical revalue device setup on an university campus.

To be able to act as man-in-the-middle, physical access to the network somewhere between the device and the server is required. A straightforward way to access the network connection of the devices is directly at the device itself. This is possible (like in Figure 6.1) because physical securing the wires is not always feasible on a site like an university campus. In addition to that, a man-in-the-middle is hard to detect remotely, because he can proxy requests using the same IP and MAC-Address as the device. Interception devices like the "Hak5 Packet Squirrel"[2] are small and easy to hide and may intercept the traffic for a very long time without getting revealed.

Likelihood: Likely, **Impact**: Critical, **Risk**: High

6.3.3. Automatic Execution of Script from Removable USB Storage

Automatic execution of a script on a USB-Stick (section 5.9) allows arbitrary code execution (e.g. using the `executefile` command) and placement of malicious files like malware (`copyfile`) to infiltrate the device or the network. Also, extraction of **Device Authentication Data**, **User Account Data**, **User Account History**, **Configuration Data** and the **Configuration Secret** (via `cs_core.exe`) is possible.

[2] `https://www.hak5.org/gear/packet-squirrel/docs`

To gain access to the port, an attacker have to gain access to the back of the vending reader in a machine setup (break into the machine) or mill off the front cover to access the USB cable connecting the card reader to the main device.

Likelihood: Unlikely, **Impact**: Critical, **Risk**: Medium

6.3.4. Broken AES Implementation

The AES implementation error as described in section 5.7 allows the modification of existing **Configuration Data**, extraction of **Device Authentication Data** as well as the creation of new configuration. In combination with subsection 6.3.2 or subsection 6.3.3 a device can be hijacked by changing the APIX URL configured in the database or prices can be updated.

Likelihood: Likely, **Impact**: Critical, **Risk**: High

6.3.5. Unauthenticated Access to Configuration

As in section 5.10 described, it is possible to acquire the device configuration database just by knowing the device serial number (also referred as "Board number" or "Platform ID").

In combination with the broken AES implementation (subsection 6.3.4), **Device Authentication Data** can be acquired remotely. This information allows a remote attacker to do malicious transactions and **User Account History** entries due to the Missing Authorization of the transaction by the User as in section 5.11 described.

Likelihood: Likely, **Impact**: Critical, **Risk**: High

6.3.6. Missing Authorization by User

Just by using the **Device Authentication Data** it is possible to do financial transactions (see section 5.11). This allows a remote attacker to do transactions without user's authorization. An attacker can obtain all cards via APIX action 14 (if available) and then obtain the card info (action 1) containing the current balance. The attacker may now redistribute ("steal") money from the cards and charge it to other accounts. Thereby, the total sum of money in the pool is not changed and the provider of the system does not receive financial damage. Given a large site with > 20.000 users, just "stealing" a value of 0.01€ from just 2.000 evenly distributed users and add 20€ to the attacker's account may not be noticed by the victims. Due to the lack of referential integrity (see section 5.13) this kind of fraud is hard to reveal and to investigate.

In addition to that, users can – technically speaking – deny that each of their transactions was actually done (non-repudiation). It is only possible, to compare device's audit logs with the server's, but both can (mathematically) not be fully trusted.

Likelihood: Unlikely, **Impact**: Critical, **Risk**: Medium

6.3.7. Missing Replay Protection

User authorization for transactions can be bypassed due of the missing replay protection (see section 5.12). If an attacker gains the plain text of a request (e.g. by using section 5.4 or by bypassing the certificate verification or transport encryption using other methods) containing an authorized financial transaction, the transaction can be replayed, because it does not contain any kind of nonce or timestamp, which would be part of an authorization (if even existent).

Likelihood: Unlikely, **Impact**: Critical, **Risk**: Medium

Weakness	**Likelihood**	**Impact**	**Risk Level**
Outdated Operating Systems	Unlikely	Critical	Medium
Usage of an Operating System with Desktop Environment	Likely	Critical	High
Usage of Weak Cipher Algorithms	Rare	Critical	Very Low
Weak Binary Integrity Check	Possible	Negligible	Low
Hard-Coded Database Encryption Key and Weak Integrity Check	Likely	Critical	High
Lack of Basic Sanity Checks for Transaction History	Unlikely	Marginal	Low

Table 6.2.: Overview of the weaknesses

Vulnerability	**Likelihood**	**Impact**	**Risk Level**
Remote Management	Likely	Critical	High
Missing Server Certificate Verification	Likely	Critical	High
Automatic Execution of Script from Removable USB Storage	Unlikely	Critical	Medium
Broken AES Implementation	Likely	Critical	High
Unauthenticated Access to Configuration	Likely	Critical	High
Missing Authorization by User	Unlikely	Critical	Medium
Missing Replay Protection	Unlikely	Critical	Medium

Table 6.3.: Overview of the vulnerabilities

7. Risk Mitigation

There are multiple ways to repair the weaknesses and vulnerabilities described above. In this case, we differentiate our proposed mitigations between techniques, which can be applied outside of the software or the physical devices (external mitigations) and those, which can be applied on the device's software or the device itself (internal mitigations).

7.1. Internal Mitigations

Internal mitigations have to be applied to the codebase of the software or the firmware image. An overview of all weaknesses with internal mitigations can be found in Table 7.1.

7.1.1. Remote Management

An upgrade of the RealVNC server used to a current version should be done, or the VNC server should be completely removed to reduce the attack surface. The server is never used in practice, because it can be only used to inspect a crashed or not starting software (see subsection 3.2.3), so removing the server would not significantly reduce the features of the device. In case the server software is kept on the device, a proper update process should be defined, to address future vulnerabilities.

7.1.2. Missing Server Certificate Verification

The certificate of the backend APIX and Heartbeat server should be verified at all times. Therefore, the root certificate of the certification authority used or the server certificate itself should be placed on the devices to ensure a secure and authentic data exchange between the device an the server.

7.1.3. Usage of Weak Cipher Algorithms

To reduce the risk of future exploits, weak or broken cipher algorithms should be disabled on the client and the server side. On vending readers, this may require the usage of a dedicated SSL/HTTP library instead of the library provided by the operating system's default API.

7.1.4. Weak Binary Integrity Checks

To ensure binary integrity, the binary should be signed and the operating system should check the integrity of the binary. Self-checking can always be bypassed and is therefore non-effective as described in section 5.6.

Additionally, integrity checks should be done with cryptographically safe one-way hashes, not using an error correcting code. As described in [Kat+96], a cryptographic one-way hash function $h(x) = y$ fulfills the following properties:

- **preimage resistance**
  It is hard to find for a given y a value x, such that $h(x) = y$.
- **2nd-preimage resistance**
  It is hard to find a $x' \neq x$ for *a given* x such that $h(x) = h(x')$.
- **collision resistance**
  It is hard to find a $x' \neq x$ for *any* x and x' such that $h(x) = h(x')$.

Any collision resistant function is also 2nd-preimage resistant and any 2nd-preimage resistant function is also preimage resistant. The property "collision resistance" is not fulfilled by CRC, as it is not preimage resistant.

Protection against unauthorized code execution could also be reached by using a trusted platform module (TPM). This is a hardware module, which acts like a smart card: It can execute code and the execution can be verified externally e.g. by the vendor.

7.1.5. Broken AES Implementation

The AES implementation should be tested against the official test vectors (see [NIST01]). This tests should be included into the unit tests as part of the quality assurance. The usage of an external library, which has been proven relatively secure could be another solution.

7.1.6. Hard-Coded Database Encryption Key and Weak Integrity Check

Hard-coded keys should be avoided because if the key of one device is compromised, the key of all devices are compromised. Each device should at least have its individual encryption key. This key needs to be deployed on the devices using a secure method.

Additionally, the configuration databases should have a cryptographic signature (like e.g. in [FIPS-DSS] defined) to ensure authenticity and integrity, since encryption alone does not provide this attribute.

7.1.7. Automatic Execution of Script from Removable USB Storage

External resources (even if physically protected) should not be trusted. Therefore we propose to sign the `cmd.xml` cryptographically (like e.g. in [FIPS-DSS] defined) to ensure, that only authentic scripts are executed. The signature should be issued by a trusted instance (e.g. the vendor), which gets verified before the execution of `cmd.xml`.

7.1.8. Unauthenticated Access to Configuration

The configuration database should not be accessible just by knowing the board ID, since it is not a secret. Access should be authenticated using a proper, individual device secret.

This may be a token, a client certificate or any other authentication mechanism. This applies independently to the database encryption in subsection 7.1.6.

7.1.9. Missing Authorization by User & Missing Replay Protection

Financial transactions should be signed by the user (e.g. by using a secret) or only by the card of the user, to ensure the authenticity of the transaction. This is only possible, if the card system used supports smart card features like signing a message using a challenge-response method. Alternatively, it may be possible to store an encrypted, card specific key to the user's card and sign transactions with this key in the device, but this "shortens" the chain of trust, because the user needs to trust the device, that it handles the user secret correctly. [FIPS-DSS] specifies a set of algorithms and methods to generate and verify digital signatures. Furthermore, this kind of signature should contain a timestamp or a nonce to prevent replay attacks. These methods would prevent an attacker from making transactions without deeper knowledge of each user's transaction secret.

7.1.10. Lack of Basic Sanity Checks for Transaction History

To ensure a sane audit trail, transaction history and the transaction itself should not be handled separately. If the transaction request would contain the timestamp of the transaction, the history can be created implicitly using the information from the transaction request. This method ensures a transaction history with correct referential integrity and each transaction is logged correctly. Additionally, sanity checks for date and amounts should be introduced, which rejects the transaction in case of an invalid transaction, which violates the rules defined by those checks. If the offline feature is used, transactions cannot be revoked afterwards, because the product or service is already payed – so the offline feature increases the risk of invalid transactions.

7.2. External Mitigations

External mitigations can be applied independently from the vendor by the **System Administrator** to the system, to mitigate weaknesses or vulnerabilities. This generally results into additional actions required by the **System Administrator** (higher costs), if the vendor does not supply a fix within time or not at all. Additionally, not all weaknesses can be mitigated externally, as Table 7.1 shows.

7.2.1. Securing the Network connection

To mitigate the missing certificate verification (see section 5.4), we evaluated two external mitigations: Physically securing the connection as well as securing the communication using a encrypted VPN (Virtual Private Networking) tunnel.

Securing Physically The RJ-45 wire can be physically secured as in Figure 7.1 by making it inaccessible for users without applying physical force. This solution is not always

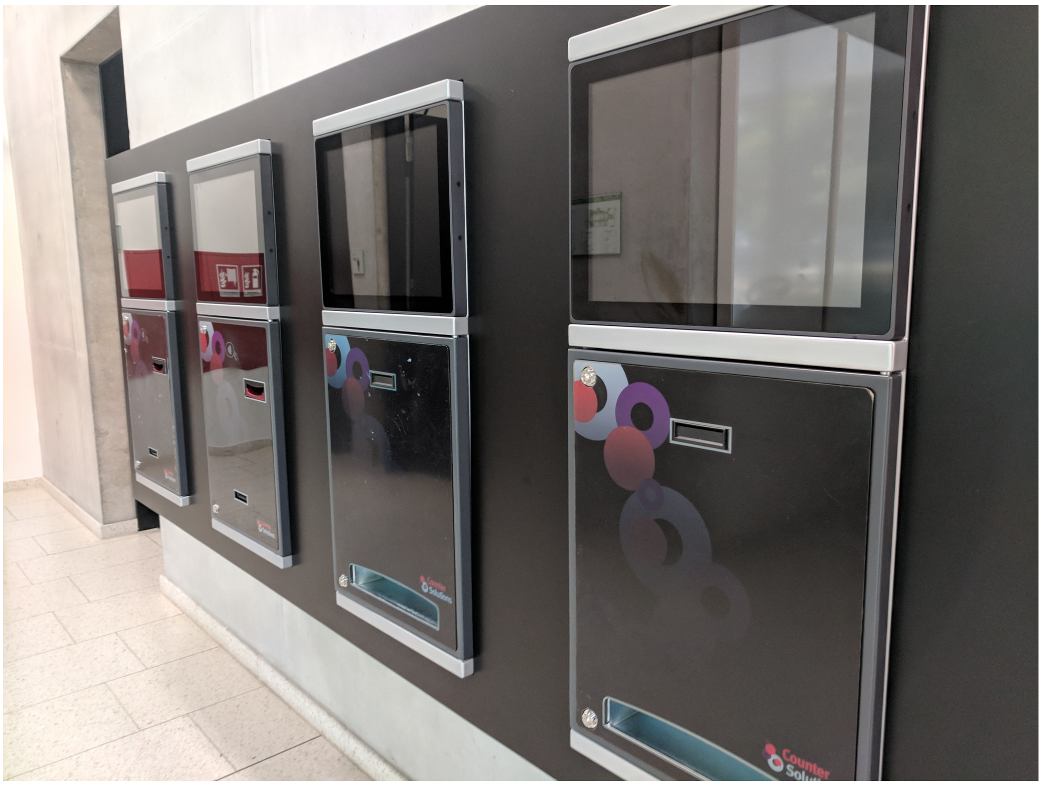


Figure 7.1.: Revalue devices secured by a solid enclosure

applicable or reliable: Installing a solid enclosure surrounding the device and the cable is not always possible because of fire prevention or available space. Alternatively, lockable RJ-45 outlets can be used, but they can be bypassed by cutting the cable and crimping new plugs to the ends. Additionally, these methods do not prevent man-in-the-middle attacks on a higher network level like in a data center or at access switches.

Securing with Encrypted VPN The connection can be secured using an additional device which encrypts the traffic from the vending device by tunneling it through a VPN tunnel terminated by a VPN server.

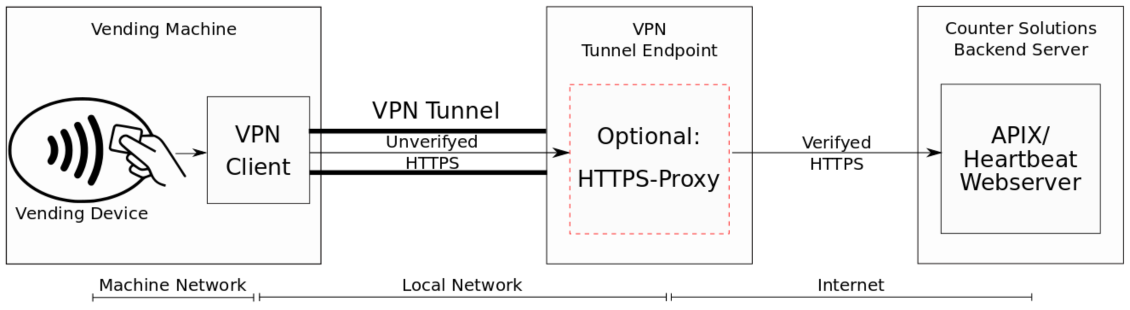


Figure 7.2.: Securing the connection using a VPN tunnel and a proxy server

This requires a dedicated device build into the machine to establish the tunnel, because the operating system image of the vending readers is hard to modify (see subsection 3.2.3). This kind of device should physically fit into a vending machine, should be cost effective and should be able to securely establish a tunnel to a tunnel endpoint as in Figure 7.2 illustrated.

We evaluated two devices to accomplish this:

1. MikroTik Router
   Small Linux based routers with inaccessible, closed operating system and two to five ports in a cost range from about 30€ to 60€

2. Ubiquiti Edge Router X
   Small Linux based router with freely accessible Debian based operating system and 5 ports for about 40€

During evaluation, we discovered a missing certificate verification vulnerability in the MikroTik OpenVPN client custom implementation [JS18], whereas the Ubiquiti Router is based on a Debian distribution, so the OpenVPN client is the original implementation and receives (security) updates by the original project.

Overall this solution is not applicable since it introduces significant costs, may increase the attack surface, increases the maintenance and update overhead, and does not secure the connection end-to-end, as the traffic behind the local network to the backend server is still not secured. The end-to-end security may be provided by installing a transparent HTTPS proxy into the VPN endpoint, which uses the man-in-the-middle attack to verify the server certificates. This may be a temporary solution, if the risk of a local attack on the network connection is not acceptable.

7.2.2. Securing the Desktop Environment

To reduce the risk of an attack caused by the underlying desktop environment (see section 5.2), the capabilities of the desktop environment should be reduced to a minimum, by deactivating, restricting or uninstalling features. Windows 10 provides a single-app kiosk mode[1]. Alternatively, an operating system without requirement for a desktop environment (like a Linux distribution) may be used.

For the **Card Terminal**, the touchscreen can be disabled to reduce the attack surface exposed to an attacker in case of a software crash. This does not significantly reduce the functionality of the terminal, as all displayed information disappears after a few seconds automatically, without the need to touch any button.

[1]`https://docs.microsoft.com/en-us/windows/configuration/kiosk-single-app`

Weakness	**External**	**Internal**
Outdated Operating Systems	✗	✓
Usage of an Operating System with Desktop Environment	✓	✓
Remote Management	✗	✓
Missing Server Certificate Verification	✓	✓
Usage of Weak Cipher Algorithms	✗	✓
Weak Binary Integrity Check	✗	✓
Broken AES Implementation	✗	✓
Hard-Coded Database Encryption Key and Weak Integrity Check	✗	✓
Automatic Execution of Script from Removable USB Storage	✗	✓
Unauthenticated Access to Configuration	✗	✓
Missing Transaction Authorization by User	✗	✓
Missing Replay Protection	✗	✓
Lack of Basic Sanity Checks for Transaction History	✗	✓

Table 7.1.: Overview of weakness mitigations

8. Discussion

The analysis in the chapters 3-7 revealed significant implementation and design issues in the payment system. In this chapter we will discuss, whether the issues could have been avoided, if existing standards and guidelines would have been applied. First we discuss if the system can be *operated* in compliance with the *BSI Grundschutz* in section 8.1. After this we discuss in section 8.2 whether the industry standard PCI PA-DSS is applicable and if it could reduce the risks found in the security analysis, if applied during the implementation of the system. This discussion allows us to conclude whether the evaluated standards are sufficient for closed loop payment systems like the system analyzed in this thesis.

8.1. Comparison with BSI Grundschutz

The *BSI Grundschutz* [BSI18c] provides detailed baselines for information security. Its core is divided into "building blocks". These blocks are containing recommendations and requirements for operating an IT infrastructure such as for the information security management system (ISMS), organisation and personnel (ORP), conception and procedures (CON), operations (OPS), detection and reaction (DER), applications (APP), IT systems (SYS), industrial IT (IND), network and communication (NET) and infrastructure (INF). One of the systems contained in SYS is the generic internet of things (IoT) device (ger.: *Allgemeines IoT-Gerät*, SYS.4.4). SYS.4.4 defines possible threats by and requirements for *operating* such IoT devices. By the definition of SYS.4.4, the payment system used at KIT is a embedded system (SYS.4.3 [BSI18b]), because the vending machines are part of a larger product. SYS.4.3 is currently not an official part of the *BSI Grundschutz* yet, so it cannot be assumed that it gets implemented. Therefore, we focus on SYS.4.4 as an alternative, which is in some points similar to the draft of SYS.4.3. While SYS.4.4 does not provide implementation requirements, the draft of SYS.4.3 does provide some implementation requirements. In the following subsections we will check, if the devices, given the identified weaknesses, can still be *operated* in compliance with the *BSI Grundschutz*. Therefore, also external mitigations proposed in section 7.2 are taken into account.

8.1.1. Remote Management

According to SYS.4.4.A3, devices and its components *must* be checked regularly for updates. In case of security issues, updates *must* be applied as soon as possible.

In case of the used RealVNC software version, the bug is known since 2006. If this requirement would have been fulfilled, the risk identified in section 5.3 would have been avoided. Since the software is part of the image, the update *must* be provided by the vendor. It is not possible to update the RealVNC software as System Administrator.

Additionally, SYS.4.3.A2 requires for the developer of the product, that *only* required services should be activated. Not required services, like the RealVNC server seems to be (it is only available for diagnostic purposes), *must* be disabled. Furthermore, access to all application interfaces *must* be protected with authentication and debugging interfaces *should* be disabled (SYS.4.3.A6).

8.1.2. Software Integrity and Operating System

According to SYS.4.3.A8, the operating system used on devices *should* be adapted for the secure use in the embedded application. The operating system *should* be capable of the required security mechanisms. As the usage of a operating system with full capabilities of the underlying desktop environment available (see section 5.2) and the weak binary integrity check (see section 5.6) demonstrate, this requirement is not fulfilled. Additionally, the usage of a trusted platform module (TPM) is recommended, to prevent unauthorized code execution.

8.1.3. Missing Server Certificate Verification

The requirement SYS.4.4.A11 defines, that secure protocols like SSL/TLS *should* be used. However, SYS.4.4.A11 does not require proper verification of the communication explicitly. Therefore, the risk identified in section 5.4 would not haven been covered by SYS.4.4 explicitly.

If we interpret "secure" in a way, that verification is included, the vending system does not use a secure connection. In this case, SYS.4.4.A11 *requires* the usage of a VPN tunnel, to secure the communication. Therefore, the device can be operated in compliance with the *BSI Grundschutz* regarding the missing certificate verification, if a VPN is used like in section 7.2.1 proposed.

8.1.4. Usage of Weak Cipher Algorithms

SYS.4.4.A11 defines the usage of SSL/TLS, but does not define minimum requirements for those protocols. If we apply general BSI recommendations for key sizes [BSI18a] during implementation, most of the weak ciphers would have been disabled.

8.2. Comparison with PCI PA-DSS

The PCI Security Standards Council is an organisation founded by American Express, Discover, JCB, Mastercard and VISA and has participations from banks, hardware and software developers and POS vendors. The "Payment Card Industry (PCI) – Payment Application Data Security Standard (PA-DSS)" [PA-DSS] is a payment industry security guideline for payment application *vendors and developers*. It defines several requirements for payment systems with focus on credit card cashless payment. In the following subsections we will check, if the payment industry's software security standard covers the issues found in this work.

8.2.1. Cryptographic Weaknesses

PCI PA-DSS requirement 11 "Encrypt sensitive traffic over public networks" [PA-DSS, p. 68] defines "if the payment application sends, or facilitates sending, cardholder data over public networks, the payment application *must* support use of strong cryptography and security protocols to safeguard sensitive cardholder data during transmission over open, public networks" [PA-DSS, p. 68]. "Public networks" include connections through the internet. The protocol *should at least* only accept trusted keys and certificates. This is not fulfilled in the application, as the missing certificate verification in section 5.4 shows.

Requirement 11 also defines, that "SSL and early TLS are not considered strong cryptography. Payment applications *must not* use, or support the use of, SSL or early TLS. Applications that use or support TLS *must not* allow fallback to SSL" [PA-DSS, p. 68]. Again, this requirement is clearly not fulfilled, as section 5.4 shows.

Requirement 5 "Develop secure payment applications", section 5.2 explicitly requires to "develop all payment applications to prevent common coding vulnerabilities in software-development processes" [PA-DSS, p. 46] and explicitly mentions the OWASP Top 10, SANS CWE Top 25 and others. The AES implementation error as described in section 5.7 could have been prevented, if the quality assurance and review processes of security relevant code would have been implemented along PCI PA-DSS.

8.2.2. Management Weaknesses

PCI PA-DSS requirement 7 states to "test payment applications to address vulnerabilities and maintain payment application updates" [PA-DSS, p. 59]. This is also required for "any underlying software or systems that are provided with or required by the payment application" [PA-DSS, p. 59]. The vendor *must* "identify new security vulnerabilities using reputable sources for obtaining security vulnerability information" [PA-DSS, p. 59]. Such sources to identify security vulnerabilities may be the MITRE CVE Database. Additionally, the vendor *must* "assign a risk ranking to all identified vulnerabilities" [PA-DSS, p. 60], "test payment applications and updates for the presence of vulnerabilities prior to release" [PA-DSS, p. 60] and "*must* establish a process for timely development and deployment of security patches and upgrades" [PA-DSS, p. 60]. The bug in the RealVNC implementation as described in section 5.3 is in the MITRE CVE database since 2006 [Int06]. If the product would have been maintained in compliance to PA-DSS, the RealVNC server would have been updated as soon as possible.

The USB-Feature of the devices is mainly used for updates. "Patches and updates are delivered to customers in a secure manner with a known chain of trust" [PA-DSS, p. 61]. The unverified data on the USB-Stick with arbitrary code execution (see section 5.9) breaks the chain of trust. Additionally, "Patches and updates are delivered to customers in a manner that maintains the integrity of the patch and update code" [PA-DSS, p. 61]. This is not fulfilled. The `cmd.xml` is executed without any verification of the script or the content.

Configuration deployment (see section 5.10) is not directly covered by PCI PA-DSS, but it requires generally secure authentication for administration of those systems (requirement 3). This may *indirectly* imply also securely authenticated channels for automated deployment processes.

8.2.3. Weaknesses in Financial Transaction Handling

The missing authorization by the user for financial transactions as described in section 5.11 is not covered by PCI PA-DSS, since the guideline is originated in the credit card payment industry. In this industry branch, it is by design impossible to do a transaction without the authorization of the user.

Transaction audit logs are not directly covered by PCI PA-DSS, too, as this part is covered by the banks in open loop/credit card systems.

8.3. Summary

Most of the requirements defined by the *BSI Grundschutz* SYS.4.4 cannot be fulfilled by the operator of the system and require updates by the vendor.

PCI PA-DSS does provide a strong guideline for the implementation of a payment application. Nevertheless it is designed to apply to the existing credit card system and does therefore not cover design baselines for a "new" payment system. Whereas it defines requirements for secure maintenance and update processes in detail, it lacks requirements for secure, automated configuration rollout and management for large setups with multiple instances of an application, which becomes increasingly important on large, centrally managed appliances.

Implementing a strictly PCI PA-DSS compliant payment application is more expensive than implementing a system, which does not necessarily need to be PCI PA-DSS compliant, due to the high requirements. For a system like that one analyzed in this work, a cost effective trade-off between a strict PCI compliance and sufficient security has to be found. The following aspects rendered to be vital in the system used at KIT:

1. Security of the network connection
   Check encryption and verification of cryptographic signatures, as well as verification of the server identity at client side and client identity at server side

2. Security of management interfaces and configuration deployment
   Ensure that management interfaces are *only* accessible by the System Administrator, and that they are sufficiently secured

3. Security of secrets
   Ensure that there are no secrets exposed to the User or other unauthorized roles

4. Definition of update process
   Ensure that there is an update process defined and that updates are sufficiently secured (transport as well as verification of the updates)

5. Definition of risk management process
   Ensure, that the vendor provides a process to handle possible security issues

9. Conclusion & Future Work

In this thesis, we analyzed the cashless payment solution used at KIT by observing the behavior of the software and the network traffic, as well as reverse engineering the payment application software "`cs_core`" to identify security weaknesses.

The analysis methods used were effective to analyze the system. To detect open network ports, a port scan was done. A man-in-the-middle attack was used to observe the interaction of the devices with the backend server's API, which allowed us to understand the mechanisms and usage of the API. The software was then extracted from the Vending Readers used at vending machines, as well as from Revalue Devices, used to recharge the user accounts. By disassembling the software, it was possible to investigate more functions like update procedures and integrity checks. To understand the local database encryption as well, we debugged a part of an utility program by wrapping the required library call into an custom binary. The analysis revealed 13 weaknesses in the cashless payment system.

The operation of the system poses a significant risk. After identifying weaknesses, we analyzed them to evaluate the risk of attacks, composed using the weaknesses. We therefore created an attacker model to estimate the likelihood of an attack. The impact was evaluated by defining the target data: If the target data can be modified, the impact is higher as if the data can be obtained, and negligible if the data can neither be obtained nor modified. Those values where then combined to a risk score in a qualitative fashion using a consequence probability matrix. Six of the identified weaknesses were ranked with a high risk level during the risk analysis.

It is the vendor's responsibility, to mitigate the risks. We proposed mitigations for the weaknesses found. To mitigate the weaknesses identified, we differentiated between two mitigation types: Internal mitigations, which have to be applied to the firmware image or the code of the software by the vendor, and external mitigations, which can be applied by the system administrator independently form the vendor. External mitigations are causing additional costs for the operator of the system. We proposed internal mitigations for all the weaknesses identified. For two of the weaknesses, we also proposed external mitigations. While the evaluation of network devices, to build a secure VPN tunnel as external mitigation, we discovered a vulnerability in the firmware of one of those network devices.

The payment system in the current state cannot be operated BSI Grundschutz compliant. The discussion showed, that the system with the weaknesses discovered, cannot be operated in compliance with the *BSI Grundschutz*, as most of the requirements cannot be fulfilled by the system administrator. For example, the *BSI Grundschutz* requires that updates for security issues *must* be applied. However, a security vulnerability known and fixed since 2006 is still present on the system and the affected component cannot be updated by the system administrator.

PCI PA-DSS could have helped, to prevent some of the problems. Comparison of the weaknesses with *PCI PA-DSS* requirements revealed, that most of the implementation errors would have been prevented, if the software was developed to be compliant with this standard. But *PCI PA-DSS* also lacks of definitions for scenarios with a lot of distributed components involved: There are no process definitions for secure configuration management and mechanisms for mass deployment.

The payment system still has potential for future analysis. There was no access to the backend server software and no source code of any component was provided. Additionally, only the Vending Readers were available for analysis all the time, whereas the Revalue Device was only available for short time periods. The Card Terminals were not available at all for analysis. By the findings of this work it can be expected to find more weaknesses in the components, which were not available during this work.

Competing systems should be analyzed. A future work might be the analysis of online cashless vending systems by other vendors, as there may be similar flaws in other systems. For future analysis, we created a brief list of requirements for this type of cashless payment application. This list also provides a brief guidance for the implementation of such systems.

Application specific standards and guidelines are needed. To provide a cost effective trade off between the expensive implementation along strict security standards and a high risk of operation, the definition of best practice guidelines regarding the secure implementation of cashless online payment systems may be created. The guidelines should include secure deployment and maintenance processes for a large number of instances. This may help to increase the general information security of those systems to reduce risks, while preserving the advantages of this type of payment process.

Bibliography

[BL16a] Karthikeyan Bhargavan and Gaëtan Leurent. “On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: ACM, 2016, pp. 456–467. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978423. URL: http://doi.acm.org/10.1145/2976749.2978423.

[BL16b] Karthikeyan Bhargavan and Gaëtan Leurent. “Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH”. In: *Network and Distributed System Security Symposium–NDSS 2016*. 2016.

[BSI18a] Bundesamt für Sicherheit in der Informationstechnik. *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. May 2018. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile (visited on 2018-07-19).

[BSI18b] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grundschutz-Kompendium – SYS.4.3 Eingebettete Systeme*. Guideline (Final Draft). May 2018. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/IT-Grundschutz-Modernisierung/BS_Eingebettete_Systeme.pdf?__blob=publicationFile%5C&v=2 (visited on 2018-10-26).

[BSI18c] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grundschutz-Kompendium, Edition 2018*. Guideline. Feb. 2018. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Kompendium/IT_Grundschutz_Kompendium_Edition2018.pdf?__blob=publicationFile%5C&v=7 (visited on 2018-08-31).

[CWE-223] *Common Weakness Enumeration (CWE) ID 223: Omission of Security-relevant Information*. URL: https://cwe.mitre.org/data/definitions/223 (visited on 2018-08-08).

[CWE-284] *Common Weakness Enumeration (CWE) ID 284: Incorrect Access Control*. URL: https://cwe.mitre.org/data/definitions/284 (visited on 2018-08-08).

[CWE-294] *Common Weakness Enumeration (CWE) ID 294: Authentication Bypass by Capture-replay*. URL: https://cwe.mitre.org/data/definitions/294 (visited on 2018-08-08).

[CWE-295] *Common Weakness Enumeration (CWE) ID 295: Improper Certificate Validation.* URL: `https://cwe.mitre.org/data/definitions/295` (visited on 2018-08-08).

[CWE-300] *Common Weakness Enumeration (CWE) ID 300: Channel Accessible by Non-Endpoint ('Man-in-the-Middle').* URL: `https://cwe.mitre.org/data/definitions/300` (visited on 2018-08-08).

[CWE-306] *Common Weakness Enumeration (CWE) ID 306: Missing Authentication for Critical Function.* URL: `https://cwe.mitre.org/data/definitions/306` (visited on 2018-08-08).

[CWE-321] *Common Weakness Enumeration (CWE) ID 321: Use of Hard-coded Cryptographic Key.* URL: `https://cwe.mitre.org/data/definitions/321` (visited on 2018-08-08).

[CWE-325] *Common Weakness Enumeration (CWE) ID 325: Missing Required Cryptographic Step.* URL: `https://cwe.mitre.org/data/definitions/325` (visited on 2018-08-08).

[CWE-327] *Common Weakness Enumeration (CWE) ID 327: Use of a Broken or Risky Cryptographic Algorithm.* URL: `https://cwe.mitre.org/data/definitions/327` (visited on 2018-08-08).

[CWE-328] *Common Weakness Enumeration (CWE) ID 328: Reversible One-Way Hash.* URL: `https://cwe.mitre.org/data/definitions/328` (visited on 2018-08-08).

[CWE-649] *Common Weakness Enumeration (CWE) ID 649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking.* URL: `https://cwe.mitre.org/data/definitions/649` (visited on 2018-08-08).

[CWE-653] *Common Weakness Enumeration (CWE) ID 653: Insufficient Compartmentalization.* URL: `https://cwe.mitre.org/data/definitions/653` (visited on 2018-08-08).

[CWE-862] *Common Weakness Enumeration (CWE) ID 862: Missing Authorization.* URL: `https://cwe.mitre.org/data/definitions/862` (visited on 2018-08-08).

[Dal17] Mathias Dalheimer. *Schwarzladen: Die Schwachstellen öffentlicher Stromtankstellen.* 2017. URL: `https://gonium.net/schwarzladen.html` (visited on 2018-08-27).

[Eck18] Claudia Eckert. *IT-Sicherheit : Konzepte - Verfahren - Protokolle.* Berlin, 2018. URL: `https://doi.org/10.1515/9783110563900`.

[EVA-EPS] European Vending & Coffee Service Association. *Electronic Payment Specification For Unattended Point Of Sale (UPOS).* Specification. Nov. 2013.

[FIPS-DSS] *FIPS PUB 186-4, Digital Signature Standard (DSS).* U.S.Department of Commerce/National Institute of Standards and Technology. July 2013.

[FIPS01] *FIPS PUB 197, Advanced Encryption Standard (AES).* U.S.Department of Commerce/National Institute of Standards and Technology. 2001.

[FMS01] Scott Fluhrer, Itsik Mantin, and Adi Shamir. "Weaknesses in the Key Scheduling Algorithm of RC4". In: *Selected Areas in Cryptography.* Ed. by Serge Vaudenay and Amr M. Youssef. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–24. ISBN: 978-3-540-45537-0.

[Gil98] John Gilmore. *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design.* 1998.

[Int06] IntelliAdmin LLC i.a. *CVE-2006-2369.* The MITRE Corporation, 2006. URL: `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2369` (visited on 2018-07-16).

[ISO27005] *ISO 27005: Information technology – Security techniques – Information security risk management.* Norm. 2018.

[JS18] Janis Streib. *On Mikrotik OpenVPN Security (CVE-2018-10066).* 2018. URL: `https://janis-streib.de/2018/04/11/mikrotik-openvpn-security/` (visited on 2018-08-19).

[Kat+96] Jonathan Katz et al. *Handbook of applied cryptography.* CRC press, 1996.

[KHG08] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. "A Practical Attack on the MIFARE Classic". In: *Smart Card Research and Advanced Applications.* Ed. by Gilles Grimaud and François-Xavier Standaert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 267–282. ISBN: 978-3-540-85893-5.

[KSW97] John Kelsey, Bruce Schneier, and David Wagner. "Related-key cryptanalysis of 3-way, biham-des, cast, des-x, newdes, rc2, and tea". In: *International Conference on Information and Communications Security.* Springer. 1997, pp. 233–246.

[Mica] Microsoft Corporation. *Windows CE Product Lifecycle.* URL: `https://support.microsoft.com/en-us/lifecycle/search/1143` (visited on 2018-07-17).

[Micb] Microsoft Corporation. *Windows lifecycle fact sheet.* URL: `https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet` (visited on 2018-07-17).

[NAMA-MDB] National Automatic Merchandising Association (NAMA). *Multi-Drop Bus / Internal Communication Protocol (MDP/ICP).* Specification. Feb. 2011.

[NIST01] *NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation: Methods and Techniques.* U.S.Department of Commerce/National Institute of Standards and Technology. 2001.

[NIST15] National Institute of Standads and Technology. *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Nov. 2015. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf (visited on 2018-07-19).

[Noh+08] Karsten Nohl et al. "Reverse-Engineering a Cryptographic RFID Tag." In: *USENIX security symposium*. Vol. 28. 2008.

[PA-DSS] PCI Security Standards Council. *Payment Card Industry (PCI) – Payment Application Data Security Standard*. Guideline. May 2016. URL: https://www.pcisecuritystandards.org/documents/PA-DSS_v3-2.pdf (visited on 2018-08-31).

[Pis18] Matteo Pisani. *How I hacked modern Vending Machines*. Oct. 2018. URL: https://hackernoon.com/how-i-hacked-modern-vending-machines-43f4ae8decec (visited on 2018-10-12).

[Sch+] Ronny Wichers Schreur et al. *Security Flaw in MIFARE Classic*.

[SWKA18] Studierendenwerk Karlsruhe AöR. *Geschäftsbericht 2017*. 2018. URL: http://www.sw-ka.de/media/?file=5038_geschaeftsbericht_2017_web.pdf&download (visited on 2018-08-19).

[VP15] Mathy Vanhoef and Frank Piessens. "All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS." In: *USENIX Security Symposium*. 2015, pp. 97–112.

[Whi13] WhiteTimberwolf. *Encryption using the Cipher Block Chaining (CBC) mode*. June 2013. URL: https://commons.wikimedia.org/wiki/File:CBC_encryption.svg (visited on 2018-08-08).

A. Appendix

A.1. Vending Setups

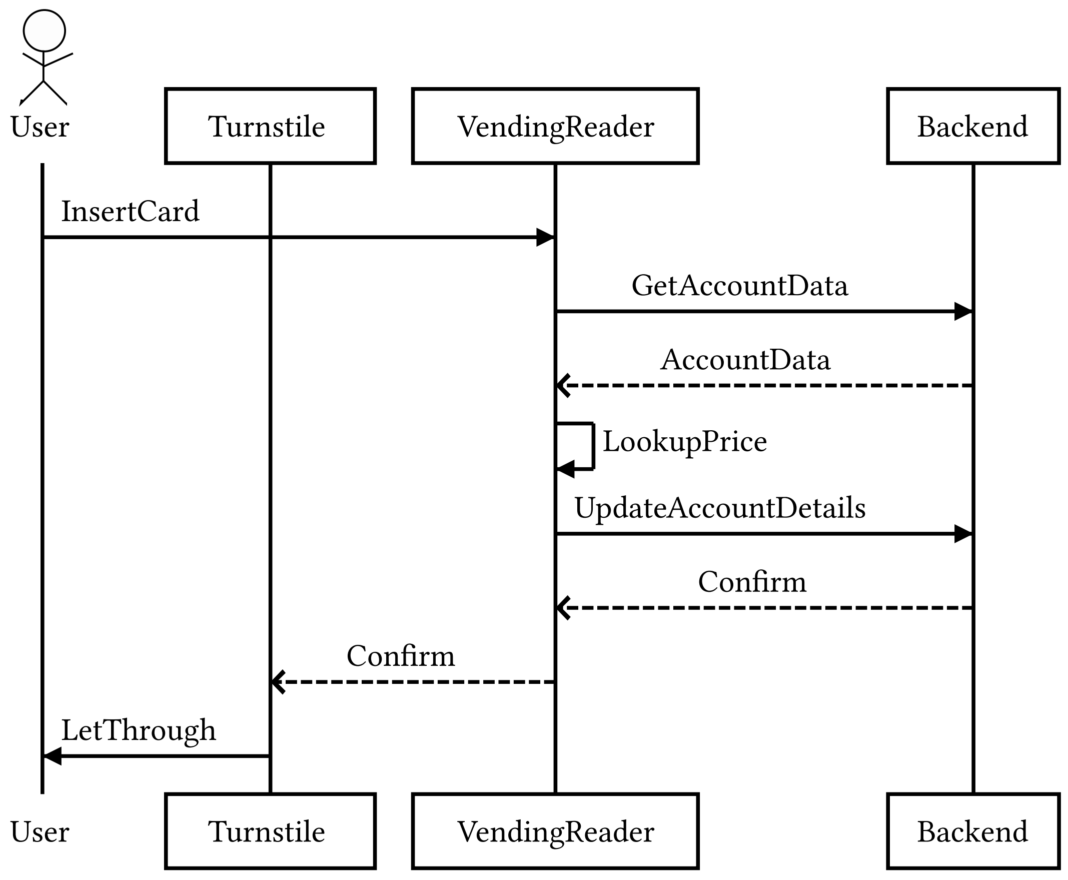


Figure A.1.: Simplified flow of vending process with UPOS and turnstile with prices defined by the POS

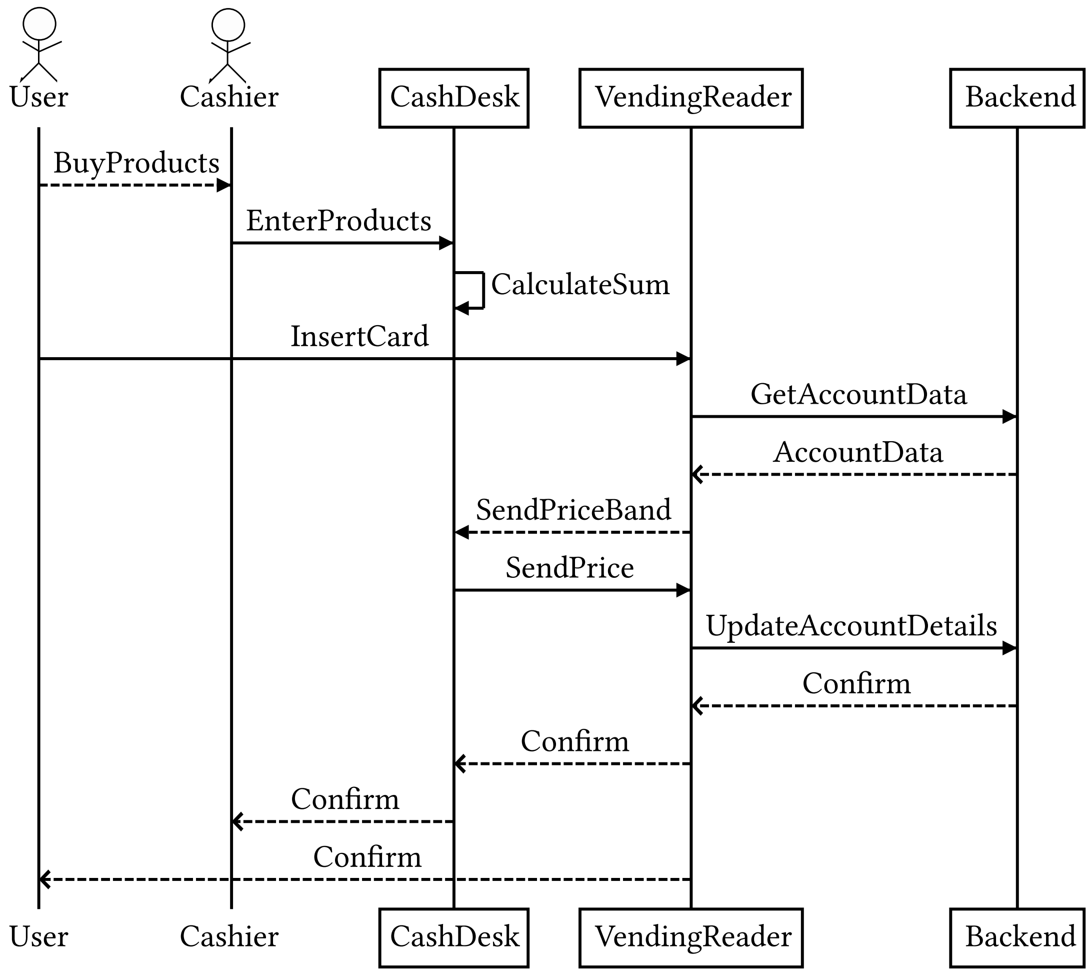


Figure A.2.: Simplified flow of vending process with POS operated by a cashier with prices defined by the POS

A.2. Commands for `cmd.xml`

Command	Description	Example
`savedisplay`	Save the current state of the display	`<savedisplay></savedisplay>`
`restoredisplay`	Restore the saved display state	`<restoredisplay> </restoredisplay>`
`display`	Display a `text` on `row`	`<display row="2" text="Please wait"> </display>`
`executefile`	Execute a file `fn`. Path substitutions `$stick` and `$rdr` are supported.	`<executefile fn="$stick\updater.exe"> </executefile>`

Command	**Description**	**Example**
`copyfile`	Copy a file from a source `src` to a destination `dst`. Path substitutions `$stick` and `$rdr` are supported.	`<copyfile src="$stick\cs_core.exe" dst="$rdr\cs_core.new"> </copyfile>`
`delay`	Delay for `ms` milliseconds by calling the architecture's native `sleep` function.	`<delay ms="1000"></delay>`
`wait`	Delay for `ms` milliseconds by busy waiting.	`<wait ms="1000"></wait>`
`dismount`	Unmount the device containing the executed `cmd.xml`.	`<dismount></dismount>`
`reboot`	Restart `cs_core`. If `skip_onerror` is set, it does only restart, if no step before failed. If `if_update` is set, it will only restart, if a update was applied.	`<reboot skip_onerror="true"> </reboot>`
`hardreboot`	Reboot the hardware.	`<hardreboot></hardreboot>`
`deletefile`	Delete the file `fn`. Path substitutions `$stick` and `$rdr` are supported.	`<deletefile fn="$stick\cs_core.exe"> </deletefile>`
`copyaudit`	Copy the audit log to `dst`. Path substitutions `$stick` and `$rdr` are supported.	`<copyaudit fn="$stick\audit.xml"> </copyaudit>`
`legicupdate`	Load new firmware file `fn` for card readers made by the vendor LEGIC. Path substitutions `$stick` and `$rdr` are supported.	`<legicupdate fn="$stick\legic.bin"> </legicupdate>`
`purgelogs`	Purge logs.	`<purgelogs></purgelogs>`
`unprotectaudit`	Not fully known. Needs to be executed before `purgelogs`	`<unprotectaudit> </unprotectaudit>`
`xmlputstr`	Put string `value` in XML file `fn` for location specified in the XPath expression `xpath`. Path substitutions `$stick` and `$rdr` for the `fn` attribute are supported.	`<xmlputstr value="bar" fn="$rdr\sample.xml" xpath="/element/@attribute"> </xmlputstr>`
`engineer`	Unknown. Requires argument `password`.	
`heartbeat`	Trigger an heartbeat (see subsection 3.5.1).	`<heartbeat></heartbeat>`

Command	Description	Example
updateprogram	Update program dst with program src. If cmp is set, the update will only executed if the contents of src and dst are not equal. Path substitutions $stick and $rdr for the fn attribute are supported.	<updateprogram src="$stick\cs_core.exe" cmp="true" dst="$rdr\cs_core.exe"> </updateprogram>
inserttotals	Unknown	<inserttotals> </inserttotals>
updatekernel	Not fully known. Update the base system with the image file in img using library lib, which is loaded during the process.	

Table A.1.: Commands usable in cmd.xml. Documentation was obtained by analyzing the cs_core and may therefore not be complete.

A.3. Debugging of the x86 Library cs_dbpack.dll

```
#include <windows.h>
#include <stdio.h>
#include <string.h>

void main(int argc, char* argv[]){
    HINSTANCE dllHandle = LoadLibrary("cs_dbpack.dll");
    // Get function pointer of the "unpack" function
    int (*unpack)(char *, int , char *) = GetProcAddress(dllHandle, "unpack");
    // Read a previously base64 encoded, encrypted work.db
    FILE *f = fopen("work.b64", "rb");
    fseek(f, 0, SEEK_END);
    long fsize = ftell(f);
    fseek(f, 0, SEEK_SET);  //same as rewind(f);

    char *workdb = malloc(fsize + 1);
    fread(workdb, fsize, 1, f);
    fclose(f);

    // 0-terminate string
    workdb[fsize] = 0;
    // Result
    char data[fsize];
    /* This call fails, but during the process (before the failure), the database gets
        decrypted into a temp-file.
      To analyze the decryption process, this is sufficient. */
     unpack(data, fsize, workdb);
}
```

Listing A.1: Wrapper code for the unpack function of cs_dbpack.dll

A.4. Proof of the Attacker Model "Curious Student"

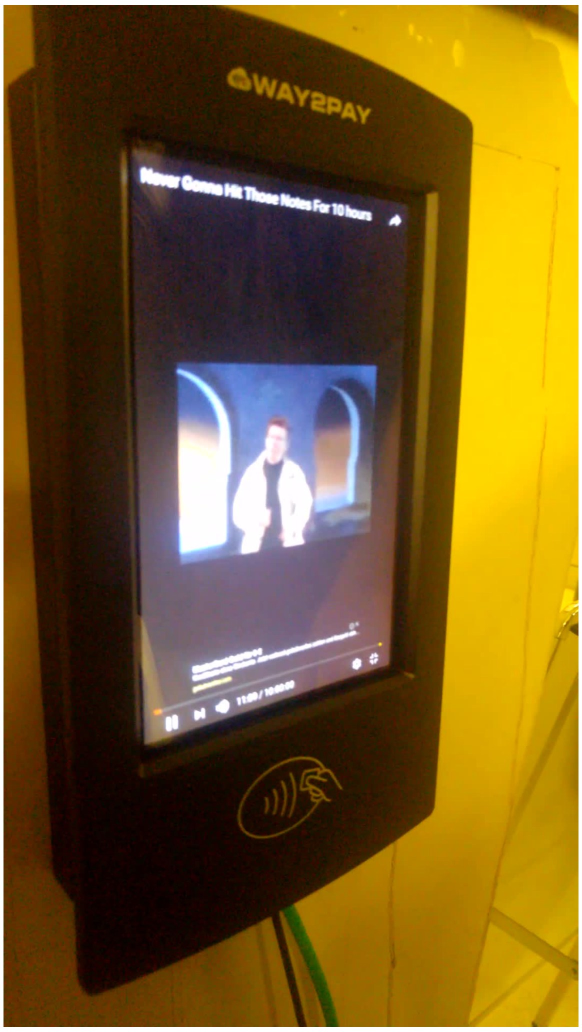


Figure A.3.: A **Card Terminal** playing a video (with audio output!) in a browser, 2018-10-11. Photo by anonymous source.