What your transportation smartcard can tell about you

Dumping and interpreting Lisboa Viva contents

David Ludovino, 64743 david.ludovino@tecnico.ulisboa.pt
Rafael Santos, 64846 rafael.h.santos@tecnico.ulisboa.pt
Duarte Barbosa, 65893 duarte.barbosa@tecnico.ulisboa.pt

Departamento de Engenharia Informática, DEI
Instituto Superior Técnico, IST
Lisboa, Portugal

January 15, 2014

1 Introduction

The Lisboa Viva is a contact and contactless smartcard used to store tickets and passes that provide access to public transports in the Lisbon Metropolitan Area. It is based on the Calypso platform[1], a set of technical specifications developed since 1990 by a consortium of European transport operators under the lead of Innovatron company.

In the Calypso platform is based on a seven-layer architecture illustrated on Table 1.

Our project initially aimed to investigate the security behind layers 4 and 7. Our plan was to explore the payment system available online at Portal VIVA[1] which allows charging Lisboa Viva cards using a smartcard reader attached to a personal computer (PC). We managed to understand the architecture behind the system, but as it is malfunctioning, it was impossible to collect information about the charging process.

Therefore, we changed the focus of the project and opted to investigate the possible privacy issues created by the use of Lisboa Viva card. To do so we analyzed layers 2, 3 and 5 in order to find what kind of information is stored in the smartcard. Using cardpeek[2] and records of readings made by Portal VIVA we managed to dump the memory contents of the card. Later, after gathering more than 50 card dumps, we were able to interpret the cards' contents, finding that they contain sensitive data such as the holder's birthdate and the last three locations where he has validated his card. Using this knowledge, we programmed an add-on script for cardpeek which translates the information stored inside Lisboa Viva into a human readable form, allowing card holders to know the sensitive information they carry around in their pockets.

On the remaining of this report we start in Section 2 by detailing the architecture behind Portal VIVA, how we were able to analyze it and the relevant information we extracted by doing so. Then in Section 3.3 we explain the process behind interpreting the card contents and detail the kind of information we found. In Section 4 we discuss the privacy issues raised by the information stored in Lisboa Viva cards. Finally on Section 5 we present our conclusions and suggestions to improve privacy in smartcard based public transportation systems.

[1] https://www.portalviva.pt
[2] https://code.google.com/p/cardpeek

2 Analyzing Portal VIVA's card interactions

In essence a Lisboa Viva smartcard is an electronic medium used to store public transport tickets and passes/season tickets (all these will be referred to as contracts). Contracts can be bought using a variety of means: directly at the transport operator, at specialized vending machines, at Multibanco (the Portuguese ATM network) and since recently at your own PC by visiting Portal VIVA and using a smartcard reader.

Portal VIVA and ticket offices of the associated transport operators sell an apparently specialized smartcard reader up for this task, known as Leitor VIVA JÁ. After inspecting the reader we found out that it is just a rebranded SCM 3310v2 from Identive[3] with no hardware or drivers modification. We confirmed this by successfully using Portal VIVA with an original SCM 3310v2. The SCM 3310v2 is a simple with-contact reader compatible with ISO 7816 standard.

With the reader in hands we tried to purchase a contract while carefully listening to the transaction. The aim was to gain knowledge on the workings of Portal VIVA.

2.1 Security flaws on user identity proof

Before being able to buy contracts through Portal VIVA one has register in the system. During the registration process the user has to insert his Citizen Card, the Portuguese e-ID card, in the smartcard reader. This operation is meant to verify the user's identity. However, in practice, the way it is implemented only guarantees possession of the Citizen Card and not that the rightful owner is the one using it.

The Citizen Card is a smartcard which stores a set of private keys that can be used by its holder to authenticate himself electronically, e.g., to prove his identity to a website. The usage of these keys is protected by a PIN

[3] http://www.identive-infrastructure.com

	layer	standards
7	Security Management and Architecture	none
6	Terminal Application Software	none
5	Data Model	Intercode, ITSO, ...
4	Card and SAM security functions	none
3	Card Data Structure	EN 1545
2	Card OS, Files structure and commands	ISO 7816-4
1	Contact and contactless communication interface	ISO 7816-(1-3), ISO 14443-(1-4)

Table 1: Calypso platform layers.

known as authentication PIN. Besides the authentication PIN this e-ID card also has another PIN, the address PIN, which simply allows access to the holders address which is stored inside the card. The address PIN is 0000 by default on all cards.

Instead of asking for the authentication PIN, Portal VIVA asks for the address PIN. Thus, it is possible for an attacker to impersonate someone by forging a driver that feeds fake information to Portal VIVA instead of reading the card. The only real information the attacker needs is the victim identity number which is easily accessible on a myriad of documents.

After insertion of the address PIN the user account is created and one has immediate access to information about all Lisboa Viva cards owned by the Citizen Card holder and all the contracts which were purchased along the years. We understand that requiring the authentication PIN would severely hinder access to the system as only a few people know it by heart. However we believe that the purchase records are sensitive data which ought to be better protected. A simple solution is to not disclose them until proper authentication is provided.

2.2 Portal VIVA architecture

The first thing we noticed was that the Citizen Card operation uses a Java applet acting as middleware between the browser and the card reader drivers. Later, we observed the same architecture is used to interact with Lisboa Viva cards. Before allowing the purchase of contracts, Portal VIVA asks for the insertion of a Lisboa Viva card in the reader and reads and interprets its contents. In order to figure out the steps involved in this operation we:

- Captured the network communication packets using Wireshark[4] and the USB packets using USBPcap[5].
- Decompiled the Java applet and read the resulting code.
- Read the JavaScript scripts included in the page, which were not minified or obfuscated. We also followed their execution path using Firebug[6].

Using this information we uncovered the architecture of interaction between Portal VIVA and Lisboa Viva cards, depicted on Figure 1. The Portal VIVA HTML page has

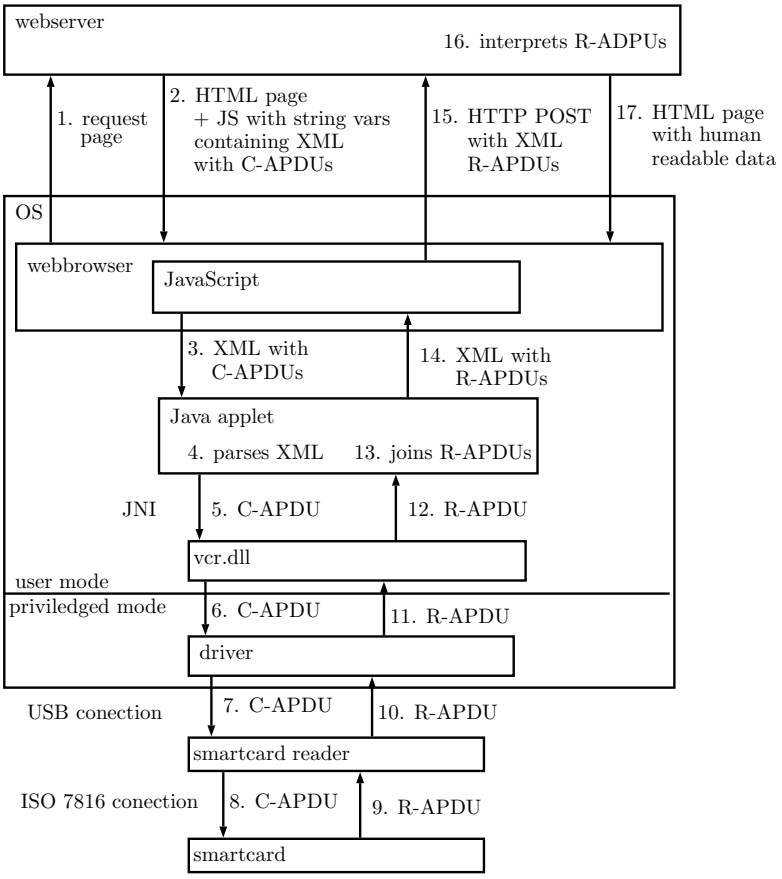

Figure 1: Diagram of the interaction between Portal VIVA and a Lisboa Viva smartcard.

several JavaScript (JS) scripts and a Java applet that is launched when the page is loaded by the browser. The JS comes with predefined string variables containing XML with one or more ISO 7816-4 C-APDUs (Command Application Protocol Data Unit) inside. This XML is sent to the Java applet which parses it and uses JNI (Java Native Interface) to call a DLL, `vcr.dll`, with each C-APDU. The `vcr.dll` is then responsible for calling the smartcard reader driver and retrieving the resulting Response APDU (R-APDU) back to the Java applet. The applet joins the several R-APDUs inside an XML string and sends it to the JS. The JS posts this XML as is to Portal VIVA server which answers with a new HTML page containing an interpretation of the data that went inside the R-APDUs. This new page may also contain new strings with XML in the JS, which are used to repeat the process and retrieve

[4] `https://www.wireshark.org`
[5] `http://desowin.org/usbpcap`
[6] `https://getfirebug.com`

more data from the card.

With this architecture no processing logic is executed in the client PC. The entire software stack on the client side is in place just to direct C-APDUs into the card and retrieve R-ADPUs back into the server.

After this we proceeded to the contracts purchase. Unfortunately we were unable to record any interactions because the credit card payment system of Portal VIVA was malfunctioning. Some days later, after warnings from our part, the purchase system was disabled and is still unavailable as of the date of writing of this report.

3 Dumping and interpreting Lisboa Viva contents

Being unable to analyze the transactions which write contracts into the card, we opted to focus our attention on the byte streams that Portal VIVA retrieves from it, and inquire into what information they contain.

Meanwhile we found cardpeek, an utility which provides a rich Lua[7] API that can be used to easily retrieve data from smartcards and interpret it. Cardpeek already comes with a Lua script, named `calypso.lua`, that is able to dump the contents of Calypso cards.

3.1 Calypso file and data structure

The data inside Calypso cards is organized into an hierarchical structure of files according to ISO 7816-4[2] (check Table 1) and much like any current Operating System. There are two types of files: Dedicated Files (DF), also known as directories of files, and Elementary Files (EF). A DF contains EFs and other DFs. An EF contains only data, which is organized into records. An EF can have one or more records. There are three types of EF files which vary according to the way their records are organized:

- **Linear files** contain a set of records organized in sequence. It is possible to read or write any record directly.
- **Cyclic files** contain a set of records organized in a stack, with the most recent on top. It is possible to read any record. However, writing is only possible by appending a new record to the top of the stack, while discarding the oldest record.
- **Counter files** are like Linear files but besides reading and writing allow incrementing and decrementing a value to one or a few records.

The ticketing information inside a Calypso card is contained under a single DF which has several EFs of different types (check Figure **??**). These allow storing at least:

- application and holder information,
- 4 contracts,
- 4 counters (0 to 16.777.215 each),
- log of at least the 3 last events,
- card manufacturing information and secret keys.

Besides the Calypso DF, a Lisboa Viva card contains two EFs under the root DF, ICC and ID, storing one record each.

3.2 Proprietary APDUs used by Portal VIVA

Reading data from the card is simply a matter navigating through the file structure, selecting a file and reading its contents by sending the C-APDUs defined in ISO 7816-4[8]. The `calypso.lua` script uses these C-APDUs to retrieve the ticketing information inside Calypso cards.

A C-APUD is a byte stream composed of at most seven fields:

CLA, 1 byte, class of instruction

INS, 1 byte, instruction code

P1, 1 byte, parameter 1

P2, 1 byte, parameter 2

Lc, 1-3 bytes, number of bytes present in the data field

Data, Lc bytes, string of bytes with additional data

Le, 1-3 bytes, maximum number of bytes expected in the data field of the response to the command.

By comparing the records retrieved by Portal VIVA and `calypso.lua` we found that Portal VIVA was able get one record from the ID file, while `calypso.lua` had the access denied. We then noticed that before doing so Portal VIVA sent a PIN using the VERIFY C-APDU (check Figure 2).

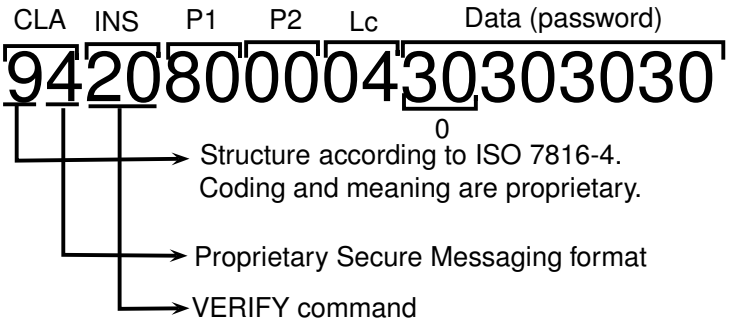

Figure 2: VERIFY C-APDU sent by Portal VIVA.

The most important aspect of this C-APDU are its first 8 bits, `0x94`, belonging to the CLA field. These tell us that although the structure of the APDU follows ISO 7816-4, the coding inside the various fields is proprietary. In order words, Calypso uses undefined behavior in the standard to implement more functionality. For instance, according to the standard, the predicates of a VERIFY commands should always be 00 but we find an 80 in P1. All other

[7] `http://www.lua.org`

[8] `http://www.ttfn.net/techno/smartcards/iso7816_4.html#table11`

C-APDUs sent by Portal VIVA carry the same CLA alongside undefined values in P1 and P2.

We managed to have a notion of which operations are done by those C-APDUs. Some are able encode two instructions that according to the standard had to be sent separately. A single C-APDU is able to instruct the card to both select a given file and then read a record from it. We believe this sort of functionality is used to speed up the ticketing validation transactions. Still, we were unable to decipher the complete logic behind the proprietary values in the predicates.

All C-APDUs sent to the card receive two types of responses (R-APDUs). For instance, as result of a VERIFY instruction or a file selection we simply receive a positive/negative confirmation. However, when we request a read operation over a record we receive a 29 bytes of data. This is the full size of the records used in the card. The data analysis in the next section is over this responses.

3.3 Interpreting the ticketing information

The codification of data elements (dates, time, currency amounts, string encoding, etc.) inside the EF's records is done according to the EN 1545 standard. This standard defines the number of bits used by each data type and how they should be interpreted. However, it does not describe the position of the data types in the card's records and does not mention which data types should be present in a given record. All those decisions are left to the ticketing system designers. Furthermore, the standard is not freely available to the public and we could only find scarce information about it on secondary sources.

After dumping the card's records, the `calypso.lua` script checks the country and network codes of the Calypso card and calls a script specialized to interpret the records of that given card. Cardpeek comes with scripts for Navigo (France), Mobib (Belgium) and RavKav (Israel) and each adopted a slightly different layout for the data. Trying to decode Lisboa Viva records using the same logic applied by these scripts proved incomplete or simply wrong. However as every Calypso implementation seems to keep the same information we used that as a starting point and started by scanning for it.

To help on our analysis, we gathered and annotated more than 50 card dumps. These were taken at carefully crafted occasions, in order to allow us the see the minimum number of bits changing.

3.3.1 ID file with holder's name

When we captured the USB communication between Portal VIVA and the card reader we found the name of the Lisboa Viva holder inside a USB packet originating from the reader. Thus we were sure it was stored in some EF. Though, our initial dumps proved unable to read this data. Only after finding the VERIFY C-APDU used by Portal VIVA (see Section 3.2) were we able to read the record inside ID file and check that it contains the holder's name in ISO-8859-1 (Latin-1) character encoding.

3.3.2 ATR and ICC file with card number

Another information we suspected to be stored in the card was the card number. Like Navigo cards, Lisboa Viva gives an ATR (Answer To Reset) that, besides identifying the card type, also includes this number. A simple conversion from hexadecimal to decimal revealed it is the number printed in the face of the card. Later we found the same value inside the only record of ICC EF. This ICC (Integrated Circuit Card) file is apparently used to store some more data about the card itself but we were unable to decode it.

3.3.3 Event logs

Event logs EF stores 3 records with information about the last 3 validations done with the smartcard, for instance while entering or exiting a transportation network. We had a feeling that these records contained location and time data that was enough to track the holders movements and thus dedicated much effort into decoding them.

Searching for timestamps was our first idea as, for practical purposes, these are inevitable to verify the freshness of card validations. Following the layout used on Navigo cards[9] we found a possible timestamp in the first 32 bits of the records. Strangely, in order for it to make sense we had to divide the value by 4, indicating that relevant data was included just in the first 30 bits. Also, unlike the common Unix timestamps, these ones did not contain the number of seconds since 00:00 of 01/01/1970. After validating a card at a known time we calculate the epoch base as being 00:00 of 01/01/1997. Later we found that all this was specified in the EN 1545 standard.

As the rest of the data was very cryptic we decided to compare records from multiple cards, on different scenarios and operators. By observing which regions of the record changed and in which scenarios allowed us to start mapping it.

The 4 bits corresponding to the transition type (entry/exit) were easily identifiable by looking at records corresponding to different transitions inside the same Metro station. One further help was that these bits were byte-aligned and we could distinguish them in the hexadecimal representation. Unfortunately for the remaining fields we had to analyze the record in binary.

The operator number was also easy to identify as its position revealed to be consistent between different operators. Then we observed a region of 2 bytes changing between the same values regularly. A detailed analysis was consistent with associating this region with the transport line. Later we found that each operator uses this region in a different way. Metro Transportes do Sul registers the line number once in this region, Metro Lisboa registers it twice, Fertagus leaves it empty as it only has one line and Carris uses it to register the bus number.

Then we searched for the number representing the station. We already had most regions mapped, and knew which information does not change, is zero or changes independently of the station. Therefore it was easy to find a candidate region to store the station, that we suspected

[9] `http://www.piratesmag.com/xxx/pass.html`

to be 8 bits wide. Later we found out that some operators register the station in this position in different ways. For instance, Metro Lisboa only uses the first 6 bits and keeps the last 2 bits to store some additional information.

After observing that a 16-bit region was repeated in some records belonging the same station we suspected and experimentally confirmed that it was the validation device code.

Finally, we read in Calypso documentation[2] that an event is associated with the contract that allowed entrance. We looked at the regions still unmapped, particularity in dumps from cards with multiple active contracts. By doing so we found a 4-bit region where each bit is associated with one of the four coutracts that can be stored in Lisboa Viva. One of these bits is always set to one, indicating that the associated contract was the one to granting access.

We managed to map Figure 3 from our work, classifying groups of bits from the Even log records according to how they are used.

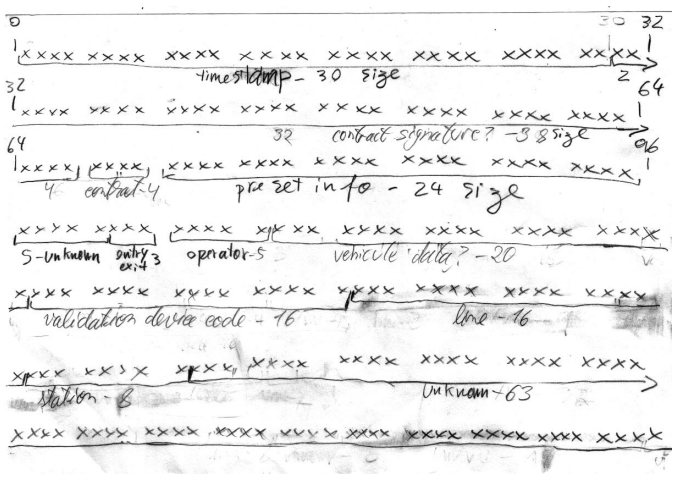

Figure 3: Mapping of the data stored on each Event log record.

Using the information collected from all the dumps we created a mapping between operator, line and station numbers and their real names. We managed to map the entire network of Metro Lisboa and Fertagus. For CP we completed the mapping of Sado and Cascais lines and made a partial mapping of the Sintra-Azambuja line. For Metro Transportes do Sul, Transtejo and Soflusa we also made a partial mapping of their networks. We believe that with additional data and time we would be able to create a complete map of all transport networks that use Lisboa Viva.

3.3.4 Contracts

The Contracts EF has 4 records which are able to store one transportation contract each. In Lisboa Viva each contract may be valid in a single operator or in a set of interconnected operators.

We started by analyzing a simple contract, for a single operator, and about which we had a good set of information, retrieved from a reading in Portal VIVA. Being a season ticket, that contract had start and end dates. To find them we just interated over all positions of the record with a function that mapped bits into a date according to the EN 1545 date standard (14 bits storing the number of days since 01/01/1997). Eventually we found two positions containing the start date we were looking for and none with the end date. Looking at older contracts we saw that the second position used to contain the end date but nowadays just keeps a copy of the start date.

Thus it seamed plausible that contracts were now storing the validity period, e.g. 31 days, instead of the end date. This assumption turned out to be right and we found 7 bits storing the validity period. Instead of a period measured in days, some contracts store it in months (usually 1). By comparing these different contracts side-by-side we identified a 16-bit wide field which indicates the time unit to be used.

With the dates mapped we run a new scan over the entire record to determine if the operator was stored using the same number as in the Event logs. It was and we found it in the first 7 bits of the contract record.

Finally, by comparing several contracts from the same operator we identified 16 bits which store the product number.

Then we turned our attention to combined contracts, valid in multiple operators. We found that in the operator number these store an high value with no correspondence to any existing operator. These contracts have information in positions to the right that are filled with zeros in the single operator contracts. Thus we directed our search there, finding that it contains 5-bit wide fields with the operators' numbers. In the middle of these fields there is data that we believe is related to the geographical validity of the contract inside each operator's network, though we had no time to further investigate this.

With all this previous information we draw the schematic of Figure 4, classifying groups of bits according to how they are used.

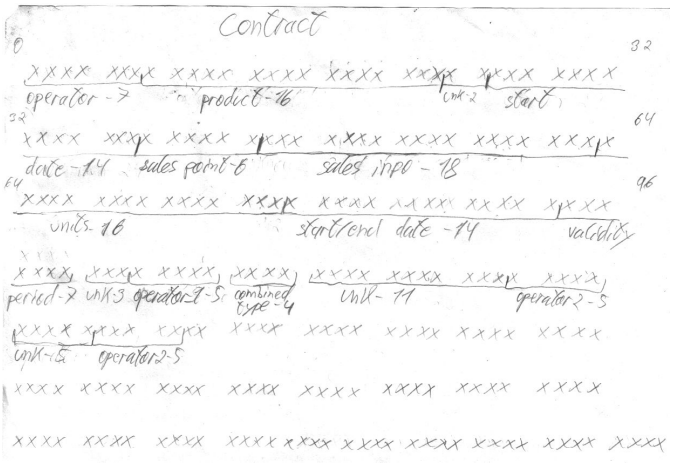

Figure 4: Mapping of the data stored on each Contract record.

As with Event logs, we used the available dumps to create a mapping between product numbers and their real name. With this were are now able to tell the name of the different contracts.

3.3.5 Counters

Each Contract record has an associated Counter EF with one record. The first Countract record is associated with the first Counter EF (202A) and so on. The Counter records are used in Lisboa Viva to store currency amounts, when a contract known as Zapping is present in the Contracts. Zapping allows the usage of Lisboa Viva to store pocket money which is decreased once we validate the card at a public transport. The current balance was easily spotted on the most significant 3 bytes of the counters. Converting these bytes to decimal gives us the amount in Euro cents.

It is plausible that counters are also used to store the number of available single-journey tickets of a given type. Though, we could not confirm this as we found no Lisboa Viva card containing such contracts.

3.3.6 Environment

The Environment EF contains only one record which was processed in a similar way to the records in the Contracts EF. With a scan we confirmed our strong suspicions that this record contained the issuing and expiry date of the card.

Then, while looking at receipts printed by the automatic tickets vending machines, we found that they uniquely identify a card using two distinct numbers: the card number present in ATR and ICC and another number known as Lisboa Viva (LV) card number. With a scan that just converted values to decimal we found the LV card number in the Environment record. The first 3 digits of this number are the number of the operator that issued the Lisboa Viva card.

Finally, based on the Environment record of the Israeli RavKav we found that Lisboa Viva contains the birthdate of its holder. The birthdate is stored in 32 bits. To interpret it we have to group the bits into groups of 4 and convert them to the corresponding hexadecimal characters. The 8 characters obtained are then read like ddmmyyyy.

3.3.7 Other EFs

The remaining EFs contained by the card, Special events and Contract list, were always found filled with zeros and thus we suspect they are never used in Lisboa Viva.

3.4 Lisboa Viva interpreter script for cardpeek

Using all the information detailed in Section 3.3 we programmed an add-on script for cardpeek. Our script runs after `calypso.lua` and translates the data dumped from the card into a human readable form, as shown in Figure 5. This allows card holders to know the sensitive information they carry around in their pockets.

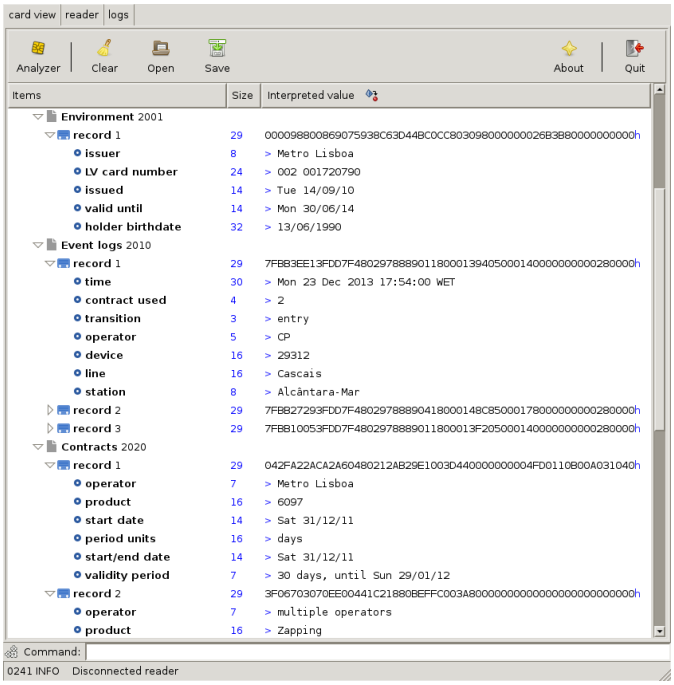

Figure 5: Screeshot of the cardpeek script displaying the contents of a Lisboa Viva card.

4 Security and privacy issues

From the privacy point being able to read the holder's name is not a big issue since this info is available on the face of the card. Other information, such as the last trip information could have more serious implications. By obtaining a card is possible to know the time, departing station and in some operators the arriving station of a trip. This information could ultimately be used to spy on a person, and allow an attacker to discover where someone lives and work and whether the person deviates from this route. While the information stored on the card is limited, and the kind of data revealed by spying on the card similar to what one would learn by looking at a paper ticket for example, a person concerned by its privacy should treat Lisboa Viva with care. Lending the card or losing it can possibly be used to reveal a lot of personal information.

smartphones -¿ NFC -¿ read peoples names. Same info available using contactless interface.

5 Conclusion

Our analysis allowed us to have a clear insight of the possible security issues with Lisboa Viva card. Although we did not manage to use the PC interface to charge a card, we believe from what we discovered, that we would not be able to easily attack it. The system avoids doing any processing outside the server or the card chip and since Calypso disclosures they use triple-DES encryption for transactions it should not be possible for an attacker to tamper with the card.

Portal VIVA show no purchase records until proper authentication.

Our experiences with Lisboa Viva card allowed us to

better understand the security mechanisms used by the card. Although the first part of the project failed we still managed to discover the Lisboa Viva system architecture a learn a lot about the interaction with the card. Our security analysis to the card dumps was more successful. We managed to decipher the information stored on the card about the user's trips. We believe that most of this info is necessary and the requirements of cost and transaction speed drove the Calypso project to adopt this approach. Still, we hope that future systems will take privacy in consideration and encrypt this kind of information or simply don't store it in the card itself.

References

[1] Calypso Networks Association. Calypso handbook. http://calypsotechnology.net/index.php/documents/specifications/public-documents/79-100324-calypso-handbook-v11, 2010.

[2] Calypso Networks Association. Calypso functional specification. http://calypsotechnology.net/index.php/documents/specifications/public-documents/78-010608-functional-card-application-v14, 2010.