Retrieving Dates in Smart Card Dumps is as Hard as Finding a Needle in a Haystack

Thomas Gougeon, Morgan Barbier, Patrick Lacharme, Gildas Avoine, Christophe Rosenberger

Retrieving Dates in Smart Card Dumps is as Hard as Finding a Needle in a Haystack

Thomas Gougeon*, Morgan Barbier*, Patrick Lacharme*, Gildas Avoine†, Christophe Rosenberger*
*Normandie Universite, ENSICAEN-UNICAEN-CNRS, GREYC UMR 6072
†INSA Rennes, IRISA, IUF

***Abstract*—This paper introduces a method to automatically retrieve dates from smart card memory dumps when the card specifications are unknown. It exploits specificities of smart cards, using a multi-dump analysis augmented with contextual information. The experiments performed on more than 180 real smart cards show that our method is highly successful in removing false positives.**

I. Introduction

In our increasingly connected world, smart cards are involved in any everyday activity, and they gather and record plenty of personal data. The need to interpret the raw data of smart card memory without the knowledge of the data structure or the data encoding has never been stronger. This kind of investigation is involved in several scenarios: (i) to establish digital evidences in connection with criminal investigations, (ii) to retrieve information about a missing person, or (iii) to verify whether a system complies with the claims of the manufacturer or the authority.

Smart cards include (but are not limited to) ski passes, electronic passports, public transport cards, payment cards, health insurance cards, and access control cards. Memory dumps of smart cards store various types of information. For example, textual (names or addresses), cryptographic (public keys), dates (travel dates, payment dates), and others (bus line, seat and mirrors position in car ignition keys).

In our context, a dump represents the content of the EEPROM (non volatile memory) of a smart card. These dumps are not retrieved with a bitwise copy but using the API of the smart card's application. In most cases, the access to the data is not protected by any cryptographic mechanism. The objective of this paper is neither to propose a method to extract the content of the EEPROM nor to retrieve the binary code of the installed applications on the smart card. This paper aims to automatically retrieve dates stored in memory dumps. Dates are important data in memory carving because they can be used for traceability or to rebuild the agenda of use of the smart card. Without the technical specifications of the application, the location of the stored information and the encoding functions used by the operator to store the information are unknown. Actually, investigators use ad hoc and manual analyses [1], [2] to retrieve this information. Such a task is hugely time-consuming and it is appropriated dealing with only one application. An automatic method is therefore needed to improve the efficiency and the quality of expert investigations as pointed out by [3].

There are plenty of encoding functions that can be used to encode dates. It can be a *counter* representing the time elapsed since a starting date or a *format*, e.g. `YYYY-MM-DD`. A naive method to retrieve dates stored in smart card dumps consists in applying several decoding functions. Each decoding function needs to be applied on all possible sequences of bits of the dump. Unfortunately, no oracle can efficiently determine whether the decoding of the information is correct. Then, applying a decoding function on a dump generates a lot of false positives. Indeed, a false positive occurs when a bit sequence in a dump is decoded with a function that is different from the one used to encode it. The scientific challenge is to define an automatic method to eliminate the generated false positives.

This paper proposes a three-step method exploiting specificities of smart card memories to retrieve dates. The first step decodes data of memory dumps to generate dates. The second step is a multi-dump analysis eliminating a part of the generated false positives, it exploits the relationship between dumps coming from the same application. The last step is a contextual analysis eliminating dates not respecting contextual information (e.g. information printed on the ticket). A human decision can be considered afterwards to eliminate the few remaining false positives. Applied on real life smart cards, the method retrieves dates in several applications while generating very few false positives. Retrieving holder birth date and creation date in electronic documents (e.g. passport), transaction dates in payment cards, and trip dates in transport cards.

The paper is structured as follows. Section II introduces the problematic with related works and an example of smart cards memory dumps. Then, Section III describes the specificities of smart cards and the proposed method exploiting them to eliminate false positives. Section IV presents the experimental protocol with the decoding functions and the dumps that are used. Finally, experimental results and validation of the method on real life smart cards are presented in Section V.

II. Problem statement

This section describes works related to our study, an example of memory dumps of a transport card application, and the false positives generated on one dump of this application.

A. Related works

The digital forensics research field can be split in several topics such as the file carving [4], the analysis of volatile memories on computers [5] and smartphones [6], and the analysis of video game consoles [7]. File carving aims to retrieve

specific files (e.g. PDF and ZIP in [8]) in a hard disk drive. A file generally starts with specific bytes indicating its format. The main difficulty of file carving is the file fragmentation on the system. Analyses of volatile memories consist in searching for special strings or signatures, interpreting internal kernel structures, or enumerating and correlating all page frames, in order to retrieve running and terminated processes, open ports, sockets, etc. Nevertheless, these techniques are often too specifics and cannot be applied to our context, as detailed below. A tool called *bulk extractor* [9] retrieves different kinds of information in bulk data such as email addresses, phone numbers, etc. but does not remove the false positives.

Memory carving of smart cards differs from traditional computer forensics because (i) the memory of smart cards is usually poorly structured, with various types of information using different encoding functions, (ii) there are no file metadata indicating the file format like JPEG, (iii) a dump is not a bitwise copy of the EEPROM memory, but a copy obtained through the API of the smart card. Nevertheless, there are also some advantages (i) dumps are generally of few kilobytes only, (ii) it is possible to perform a multi-dump analysis with several dumps of the same application, (iii) contextual information can be exploited, and (iv) an active collect of dumps is possible.

Almost all existing contributions on the memory carving problem for smart cards consider ad-hoc and hand-made analyses. For example, Avoine et al. [10] investigate the tracking of passengers using anonymous ticket in public transport. Another example, Lanet et al. [11] retrieve the area containing the code and the data (e.g., packages and classes information) of a JAVA card applet in the memory dump of a card. Few contributions propose an automatic approach. For example, Van Deursen et al. [2] investigates the memory carving problem for sets of memory dumps, and apply it to public transportation cards. They propose to automatically locate where the information can be stored on the dump. Another work is due to Gougeon et al. [12], which investigates an automatic distinction of cryptographic material in dumps of smart cards in order to eliminate areas of the memory where there is no information to decode. Nevertheless, these works do not provide any automatic interpretation of the data. An exception is the work of Gougeon et al. [13], which investigates an automatic interpretation of textual information in smart card memory dumps looking at n-grams.

To our knowledge, there does not exist any method to automatically retrieve dates in memory dumps of smart cards.

B. Dump examples

	Dump1	Dump2
Line1	01B8B24A02500033011A144300	01B939120250003301 1A144400
Line2	195603219282D2CFF36A688800	19750710928235F9A19512E000
Line3	090EE59204206086600000000	090FD5FC042060866000000000
Line4	090EE57A042060866000000000	090FD5F2042060866000000000
Line5	090EE55A042060866000000000	00000000000000000000000000

Figure 1. Extracts of anonymised transportation card dumps.

To illustrate the considered problem in this paper, this section provides details on dumps of a transport application.

Figure 1 presents two (partial) anonymised dumps of transportation cards (Calypso cards [14]). Each dump contains elementary files that have been retrieved using the card's API. The files are made of *records*, and the information is contained in (non-necessarily contiguous) *fields*, e.g. holder's name, holder's birth date, etc. The list of records is available in the Calypso standard but not their fields and consequently the encoding functions of each fields are not specified. The content is indeed let to the discretion of the public transportation operator. This example shows that three different encoding functions can be used in a single dump to store dates. It also exhibits a relationship between dumps of the same application.

`Line1` contains information about the card and its manufacturing including the serial number, the manufacturer and its country. The manufacture date, which is underlined, can be decoded converting the binary sequence to the decimal representation, corresponding to the number of days elapsed since the 1990-01-01. `Line2` contains information about the holder including the name, the gender and the postal code. It contains also the holder's birth date which is underlined and encoded with BCD (Binary-coded decimal). `Line3`, `Line4`, `Line5` contain information about the 3 last trips done, including the date and time of the trip, the bus line, etc. The trip date, which is underlined on each line, corresponds to the number of days elapsed since the 1997-01-01.

Note that there is a relationship shared by these three dumps: the binary data of each field are located at the same location in `Dump1` and `Dump2`. In `Dump2`, one can remark that `Line5` is made of zero-bits only, indeed, only 2 trips have been made with the card, therefore, there are no information stored about an eventual third trip.

Dumps of this application have been almost fully decoded without the knowledge of the application specifications, which made the task time-consuming. An automatic method is thus needed to efficiently interpret dumps of different applications.

III. Proposed method

This section presents the specificities of the smart cards and our 3-step method exploiting them, in order to retrieve dates.

A. Smart card specificities

1) Fields relationship: We assume that several dumps belonging to a given application share the same memory structure: they all contain the same fields at the same location. Only the values of these fields may vary. Let D_1 and D_2 two dumps belonging to the same application. Applying a decoding function on a given location may generate a date on D_1 while not generating a date on D_2. Due to the assumption, the generated date on D_1 is necessarily a false positive (because it does not generate a date on D_2). Analysing simultaneously the dumps by performing a multi-dump analysis can eliminate this kind of false positives.

2) Contextual information: A memory dump can store different kinds of dates: about events (e.g. trip in public transport or payment), about subscriptions (e.g. validity of a ski pass), or about the holder (e.g. birth date). Contextual information can be collected to make the investigation easier. In particular, the investigator may obtain the date of a given event or he may know that two dumps come from the same smart card. This contextual information can be exploited to express criteria on the dates generated by each candidate.

B. General sketch

In a dump, each bit can be the most significant bit of a given encoded information. The pair (bit index, encoding function) is called a *candidate*. We therefore propose a method to eliminate all candidates that are actually not used by the operator to store information. Such a candidate generates one false positive per dump of the application. By extension, we also denote such a candidate as a false positive.

This method consists of three steps. The first one consists in applying all decoding functions on dumps of the same application, generating one date value by dump for each candidate. The second phase exploits the relationship between the dumps of an application with a multi-dump analysis, removing some false positives. The third step exploits contextual information, it eliminates the candidates not respecting the criteria expressed by the contextual information. This latter step may be applied several times on an application to retrieve different dates, e.g. birth date and trip date. This method returns the remaining candidates after the analyses together with the corresponding date values that are generated.

C. Decoding phase

In order to retrieve all the dates stored in a dump without the specifications of the application, several decoding functions have to be applied on the dump. In our context of smart cards, encoding functions do not necessarily use entire bytes, and data are not necessarily byte-aligned. As a consequence, each decoding function needs to be applied using all possible binary sequence of contiguous bits of the dump.

Let f be a decoding function such that $f : \{0,1\}^m \rightarrow \mathcal{D} \cup \{0,\ 1,\ -1\}$. Let $\mathcal{D}$ denotes the set of existing dates in the Gregorian calendar. Value 0 (resp. 1) is generated by the input 0^m (resp. 1^m). These outputs may correspond to empty data, e.g. information about the third trip of `Dump3` in Figure 1. Value -1 represents outputs not compliant with the Gregorian calendar, e.g. using the ASCII decoding function the output can contain letters instead of digits.

Let $D_1, \ldots, D_N$ be the dumps of the application $\mathcal{A}$. Each dump D_i is a n-bit dump, that is represented by a binary sequence $(b^i_k)_{1 \leq k \leq n}$. Applying the decoding function f on a binary sequence of the dump D_i with $1 \leq i \leq N$ starting at index j with $1 \leq j \leq n - m + 1$ generates a date $d^f_{i,j}$: $d^f_{i,j} = f\left(\left(b^i_k\right)_{j \leq k < j+m}\right)$.

Now, rather than looking at the set of outputs generated on each dump D_i, we focus on outputs generated by each candidate (f, j) on all the dumps of $\mathcal{A}$, where j is the index on which the decoding function f is applied. It is represented by the set of date values $\{d^f_{i,j}\}$ with $1 \leq i \leq N$. This set of outputs is generated for each decoding function f considered and each possible index $1 \leq j \leq n - m + 1$.

D. Multi-dump analysis

Considering a set of values $\{d^f_{i,j}\}$ with $1 \leq i \leq N$ and $1 \leq j \leq n - m + 1$ obtained by the decoding phase, the multi-dump analysis aims to eliminate false positives.

According to Section III-A1, the relationship between dumps of an application implies the following assertion. Let (f, j) be a candidate that is really used by the operator to store a date in an application, then $\forall D_i \in \mathcal{A},\ d^f_{i,j} \in \mathcal{D} \cup \{0,\ 1\}$. Therefore, a candidate that generates dates fulfilling one of the two following assertions is a false positive and is eliminated.

$$\texttt{Assert1}.\ \exists\ i \in \{1 \ldots N\}\ s.t.\ d^f_{i,j}\ =\ -1$$

$$\texttt{Assert2}.\ \forall\ i \in \{1 \ldots N\},\ d^f_{i,j}\ \in\ \{0,\ 1\}$$

The assertions need to be checked in this order: `Assert1` then `Assert2`. `Assert1` eliminates candidates generating at least one output that is neither a date nor 0, nor 1. `Assert2` eliminates candidates generating only 0 or 1, which are generally empty data. Outputs 0 and 1 are allowed for a candidate because they may correspond to fields expected to store an event that did not occur yet. It is worth noting that, this multi-dump analysis cannot generate a false negative, i.e. it cannot remove a candidate that is used by the operator to store an information.

E. Contextual analysis

This section describes a method to exploit contextual information related to the dumps. Criteria on the dates generated by the candidates are expressed from the contextual information. Candidates generating dates not fulfilling these criteria are eliminated. For a sake of simplicity, the method is split into two filtering criteria. The first criterion eliminates candidates that do not belong to a date range and the second one relies on relationships between candidates.

To retrieve a field (e.g. a birth date), several criteria may be expressed from the contextual information. In order to retrieve several fields of an application, the contextual analysis may be applied several times with different criteria

1) Range filter: Some contextual information related to the use of the card or to the holder might be known: validation during a public transport, payment, purchase a new subscription, etc. This contextual information can be exploited to define a range, or at least define one bound of a range to filter dates. For each kind of contextual information, a bound *DateMin* and a bound *DateMax* can be assigned to each dump of an application: $[D\text{Min}_i, D\text{Max}_i]$. Therefore, all candidates (f, j) generating dates not respecting the following assertion are eliminated. `Assert3`. $\forall\ i \in \{1 \ldots N\},\ D\text{Min}_i\ \leq\ d^f_{i,j}\ \leq\ D\text{Max}_i$.

2) Relationship filter: Another way to exploit contextual information is to define a relationship between events. For example, an investigator may know that a given smart card was used soon after it was purchased. It can also be useful to look for dates related to the holder of the smart card, like the birth date or the manufacture date, when the smart card from which the dump comes from is known.

For each kind of contextual information a relationship can be established between each pair of dumps of the application. This relationship is defined by operators including (but not limited to): $<$, $>$, $\leq$, $\geq$, $=$, $\neq$. Therefore, all the candidates (f, j) that do not comply with the following assertion are eliminated. `Assert4`. $\forall\ i_1, i_2 \in \{1 \dots N\};\ d^f_{i_1,j}\ \texttt{OP}_{i_1,i_2}\ d^f_{i_2,j}$. Where $\texttt{OP}_{i_1,i_2}$ defines the relationship between the dumps D_{i_1} and D_{i_2}, and with $1 \leq i_1 < i_2 \leq N$.

IV. Experimental protocol

A. Decoding functions

1) Decoding functions categories: There exist several ways to encode a date, we separate them into two categories: *counter* and *format*. Decoding functions of type *counter* encode the time elapsed since a given date. The time unit can be for example the number of days or seconds. Decoding functions of type *format* are a representation of a date from the Gregorian calendar where each digit of the representation is stored with the BCD or the ASCII encoding. It is worth noting that, the type *format* can be seen as three *counters*, one counting the number of days elapsed since the beginning of the month, another counting the number of months elapsed since the beginning of the year, and the last one counting the number of years elapsed since A.D.

This separation into two categories is made because these two types of decoding function do not generate the same number of dates when applied on a dump, and by consequence the same number of false positives. Indeed, the type *format*, by using ASCII and BCD in order to represent the date, has an important number of inputs that do not generate digits and by consequence do not engender a date. The length m of the input of a decoding function of type *format* `YYYY-MM-DD` using ASCII is $m = 64$. As a consequence, $|\{0,1\}^m| >> |\mathcal{D}|$. For example, looking at ASCII using 8 bits, only 10 outputs among the 256 possible outputs generate a digit. Moreover, ASCII and BCD encoding schemes generate series of digits that are not compliant with the Gregorian calendar, e.g. 1995-01-37. On the other hand, all inputs of functions of type *counter* generate dates compliant with the Gregorian calendar.

For the experiments, we redefine the decoding function by allowing only dates of the Gregorian calendar posterior to $1900 - 01 - 01$ and anterior to $2050 - 01 - 01$. This lower (resp. upper) bound is chosen because only very few people are born before (resp. after) and none event nor validity dates have taken place before (resp. after). Outputs that do not respect these bounds are assigned to the value -1.

2) Selected decoding functions: We have selected 25 decoding functions to be applied on the previous dumps to validate the proposed method. These decoding functions are separated into two categories: *counter* and *format*.

Counter functions that have been used are functions using 14 and 15 bits counting days from 1990-01-01, 13, 14, and 15 bits counting days from 1997-01-01 and 31 and 32 bits counting seconds from 1970-01-01. Starting dates 1990-01-01 and 1997-01-01 are considered due to their presence in transport cards, and 1970-01-01 is the POSIX time.

Format functions that have been used represent the following formats : DDMMYY, DDMMYYYY, MMDDYY, MMDDYYYY, YYMMDD, YYYYMMDD. There exist several ways to encode each digit representing the day, year and month. ASCII, BCD, and BCD-8 are considered for the experiments. BCD-8 is BCD padded to 8 bits using 4 bits set to zero, because in order to keep data byte-aligned.

B. Dumps

1) Database: To validate the proposed method and evaluate its efficiency, we used a database of real dumps. We have 346 dumps obtained from about 180 smart cards from about 50 different applications. All these dumps have been retrieved using the API of each device. These dumps are classified as follows:

- 86 Calypso dumps from 12 devices from 5 different cities,
- 53 transport tickets from 10 cities in France, Italy, Finland, China, Canada, Norway, Belgium.
- 130 dumps from 40 ski passes from 17 ski resorts,
- 8 dumps of French and Belgian electronic passports,
- 28 payment cards from Mastercard and Visa,
- 41 dumps from 19 other various applications: Vitale (French health dossier), Belgian eID, Moneo, etc.

V. Experimental validation

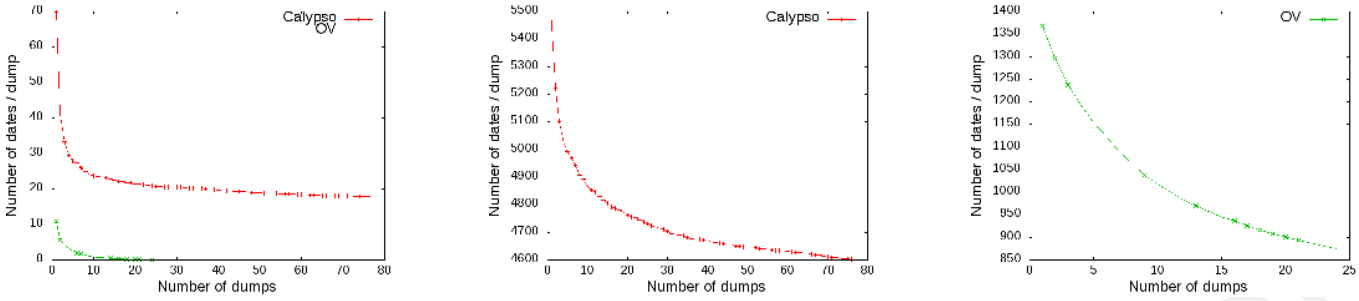
A. Decoding phase

This section provides results on applying the decoding phase on two real-life applications: Calypso and OV. OV dumps come from the public transport system in the Netherlands. A Calypso dump contains about $5,000$ bits, and analysing such a dump leads to $5,555$ candidates on average (70 *format* and $5,485$ *counter*). An OV dump contains 512 bits and provides $1,379$ candidates on average (11 *format* and $1,368$ *counters*).

From the above examples, one can notice that the decoding phase generates thousands of candidates. Most of them are false positives, which makes the output of the decoding phase unmanageable for a human. Having only few false positives (e.g. less than 10 on average), is an interesting result that lightens the work of the human investigator.

B. Multi-dump analysis

This section provides results on the multi-dump analysis on real-life applications. The presented results are the evolution of the number of dates depending on the number of considered dumps. Graphics on the Figure 2 show these results for dates of types *counter* and *format* on OV and Calypso dumps.



(a) Format dates. (b) Counter dates (Calypso). (c) Counter dates (OV).

Figure 2. Number of dates per dump after the multi-dump analysis.

1) Calypso: Performing the multi-dump analysis with 10 Calypso dumps leads to $4,893$ candidates (24 *format* and $4,869$ *counters*). It represents a reduction of 12% (65% for *format* and 11.24% for *counter*) compared to the $5,555$ candidates obtained with only one dump. The impact of adding dumps when the number of considered dumps is already higher than 10 only slightly reduces the number of remaining dates per dump. This reduction is 17% with 75 dumps. Actually, *format* dates that are still remaining represent data that are identical in all the dumps.

2) OV: Performing the multi-dump analysis with 24 OV dumps leads to 875 candidates (all *counters*) representing a reduction of 37%. As for Calypso dumps, the number of candidates is more reduced adding one dump when the number of dumps is low. The reduction rate of the OV application using 20 dumps is higher (about 35%) compared to the reduction rate of the Calypso one (about 10%). It is explained by the fact that data in the OV application differ much more between each dump than for the Calypso application.

3) Conclusion: The multi-dump analysis is not sufficient to retrieve dates in real-life application due to the high number of remaining false positives: several thousands for Calypso and several hundreds for OV. Nevertheless, this analysis can be applied without any contextual information and the false positives are still reduced by 17% for Calypso and 37% for OV. Further analyses exploiting contextual information are needed to reduce the number of false positives.

C. Contextual analysis

This section provides results on applying the contextual analysis on real-life devices, only the `Assert3` is used.

1) General statistics: Figure 3a shows the number of candidates obtained after applying the filter range with different ranges using one dump in several applications. The lower bound of the range is picked randomly in the range of date from $2000-01-01$ to $2015-01-01$ and the upper bound is fixed by adding a number of days to the lower bound. On average, allowing a range of one year leads to more than 100 candidates on a Calypso dump, and to around 30 candidates on OV and ski pass dumps. Allowing a range of one day only leads to 0.23 candidate for a Calypso dump, 0.11 for an OV dump, and 0.07 for a ski pass dump. Thus, the contextual analysis applied on one dump, even with a large range, strongly reduces the number of false positives to some decades for OV dumps and to one hundred on Calypso.

Figures 3b and 3c show the number of dates remaining per dump using different ranges for the filter of the contextual analysis, depending on the number of dumps used for the multi-dump, respectively for Calypso and OV applications.

2) Calypso: The method is very efficient using up to $5-10$ dumps for all ranges, dividing by 3 the number of candidates (compared to using only 1 dump). For example, using a range of one year, it remains only 40 candidates using 5 dumps whereas it remains 112 candidates with 1 dump. This amount of false positives becomes manageable by an investigator, but some further analyses exploiting the `Assert4` of the contextual analysis can be performed to reduce them.

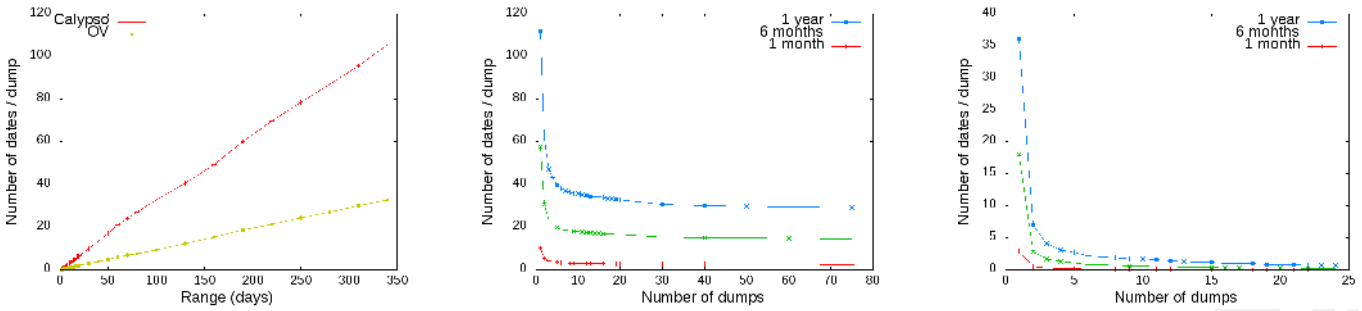
3) OV: Using only 2 dumps with the contextual information leads to a number of dates per dump that is very low. This amount of false positives is already manageable for an investigator. Indeed, with only 2 dumps, there are less than 1 date per dump with a range of 1 month, and only 7 dates per dump with a range of 1 year. To obtain, on average, less than 1 date per dump with a range of year, 20 dumps are necessary, that is also a credible scenario.

D. Real scenarios

This section provides results of applying the method on real scenarios. Obtained results are directly exploitable for a human investigator, leading to very few false positives.

1) Calypso: The holder birth date: Seeking the birth date of the holder in the dumps and applying the full analysis, 189 candidates are still remaining. To encode a birth date, it is more likely to have a *format* encoding function than a *counter* since 1970, 1990, or 1997 because people born before the starting date cannot be encoded. There are only 2 candidates that are decoded with a *format* decoding function. These 2 candidates are actually used by the operator to store the holder birth date. Thus, the analyses generated no false positives.

2) Calypso: The trip date: Seeking the trip date in Calypso dumps and applying the multi-dump analysis and `Assert3` of the contextual analysis leads to some false positives. Allowing only a range of one day (i.e. the exact dates of the trips), applying the method with 75 dumps returns the trip dates



(a) Only the contextual analysis. (b) Calypso dumps. (c) OV dumps.

Figure 3. Results after multi-dump and contextual analyses.

without any false positive. With only 10 dumps, there are 4 dates per dump on average that are returned, corresponding to the three dates trips stored in the dump, and one false positive. Allowing a range of 1 month and using all the dumps, less than 4 candidates are returned on average, representing the three dates of trips. Thus, there is on average less than one false positive per dump. Using less than 10 dumps generates on average 3 false positives per dump. Allowing a range of 1 year, generates around 50 false positives per dump, but exploiting the `Assert4`, it leads to only 12 false positives.

3) OV: The purchase date: Seeking the purchase date in 24 OV dumps and applying all the analyses with a range of one year leads to 1 date per dump, representing the purchase date. Thus, there is no false positive generated. Reducing the number of dumps to 2 generates less than 10 false positives per dump. Allowing a range of 1 year without the exploitation of the `Assert4`, leads to one false positive per dump.

4) Results overview: Calypso cards can contain dates about the manufacture, the holder birth date, and the last trips done with the card. The number of records varies from 1 to 5 depending on the operator. Transport tickets of various cities are generally single-use tickets and can store dates of validity, purchase, and usage. Ski passes usually store the date about the validity and the date of the last ski lift used. French health insurance cards store the holder birth date and can store the children's holder birth date and dates related to medical events (e.g. a long-term illness). Biometric passports store dates about the holder birth date, the creation, and the validity period. Payment cards (EMV) store dates about the last transactions done with the card varying from 1 to more than 100 depending on the operator and also the validity end date of the card.

VI. Conclusion

In this paper, we propose a method to retrieve dates in memory dumps of smart cards. The proposed 3-step method exploits two specificities of smart cards. First, a decoding phase consists in applying several decoding functions on the dump to generate candidates. Then, the multi-dump analysis exploits the similarities between dumps of the same application. Finally, the contextual analysis uses information that can be collected to select the most interesting candidates. This method strongly reduces the number of false positives generated by the decoding phase, leading to no false positive looking at trip dates of transport cards or looking at birth dates in Calypso cards. This method allows to retrieve dates in several kind of applications like transport cards, ski passes, French health insurance card, etc. It can be adapted to retrieve other kinds of information like credit card number, age of the holder, etc. Differences with retrieving dates rely on the contextual information exploited and the decoding functions to be applied on the data.

References

[1] G. Avoine, T. Martin, and J.-P. Szikora, "MOBIB extractor," Software, Jan. 2009.

[2] T. Van Deursen, S. Mauw, and S. Radomirovic, "mCarve: Carving attributed dump sets," in *USENIX Security Symposium*. San Francisco, California, USA: USENIX Association Berkeley, August 2011, pp. 107–121, Tool available at http://satoss.uni.lu/software/ccarve/,.

[3] J. I. James and P. Gladyshev, "Challenges with automation in digital forensic investigations," *arXiv preprint arXiv:1303.4498*, 2013.

[4] R. Poisel and S. Tjoa, "A comprehensive literature review of file carving," in *ARES*. Regensburg, Germany: IEEE, September 2013, pp. 475–484.

[5] M. Burdach, "Physical memory forensics," 2006, https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Burdach.pdf.

[6] P. Wächter and M. Gruhn, "Practicability study of android volatile memory forensic research," in *Information Forensics and Security (WIFS), 2015 IEEE International Workshop on*. IEEE, 2015, pp. 1–6.

[7] S. Khanji, R. Jabir, F. Iqbal, and A. Marrington, "Forensic analysis of xbox one and playstation 4 gaming consoles," in *Information Forensics and Security (WIFS), 2016 IEEE International Workshop on*. IEEE, 2016, pp. 1–6.

[8] M. I. Cohen, "Advanced carving techniques," *Digital Investigation*, vol. 4, no. 3, pp. 119–128, 2007.

[9] D. Corpora, "Bulkextractor," 2015, http://www.forensicswiki.org/wiki/Bulk_extractor.

[10] G. Avoine, L. Calderoni, J. Delvaux, D. Maio, and P. Palmieri, "Passengers information in public transport and privacy: Can anonymous tickets prevent tracking?" *International Journal of Information Management*, vol. 34, no. 5, pp. 682–688, 2014.

[11] J.-L. Lanet, G. Bouffard, R. Lamrani, R. Chakra, A. Mestiri, M. Monsif, and A. Fandi, "Memory forensics of a java card dump," in *Smart Card Research and Advanced Applications*. Paris, France: Springer, 2014, pp. 3–17.

[12] T. Gougeon, M. Barbier, P. Lacharme, G. Avoine, and C. Rosenberger, "Memory carving in embedded devices: separate the wheat from the chaff," in *ACNS*, vol. 9696. Guildford, UK: Springer, June 2016, pp. 592–608.

[13] ——, "Memory carving can finally unveil your embedded personal data," in *ARES*, Reggio Calabria, August 2017.

[14] C. CNA, "Calypso," 2017, http://www.calypsostandard.net/.