Abstract tests based on SysML models for EMV Card

Noura OUERDI *, M'hammed ZIANE and Abdelmalek AZIZI
Lab. ACSA, FSO
Mohammed First University
Oujda, Morocco
noura.ouerdi@gmail.com

Mostafa AZIZI[1], Jean-Louis LANET[2]
[1]MATSI Lab, ESTO. [2] SSD – XLIM Lab
[1]Mohammed First University. [2]University of Limoges
[1]Oujda, Morocco. [2]Limoges, France
azizi.mos@gmail.com, Jean-louis.lanet@unilim.fr

Abstract— **The smart cards are increasingly used in several fields with critical data that require security. We cite, as example, the medical field and payment shopping with smart card. Therefore, the hardware and software security of smart cards is one of the key elements of the security of sensitive information handled. Currently, several scientific researchers are interested in studying and enhancing the smart cards security. The study of vulnerabilities is a prerequisite for building security guarantees of this type of devices. Indeed, each vulnerability can easily lead to an attack. In this paper, we generate vulnerability test cases based on models of Europay-MasterCard and Visa (EMV) specifications.**

Keywords- EMV, smart card, SysML, Event-B, test cases

I. INTRODUCTION

Embedded systems perform predefined tasks. They have binding specifications to fill in terms of energy consumption, cost, dependability and safety. Indeed, these systems may be holders of confidential information that should be maintained and protected for their users. In particular, as regards the transmission of payment information through a smart card. In this context, smart cards have become targets for several reasons: First, they are available, easy to obtain [1] and inexpensive, which helps attackers to acquire large number of tests. Second, they are mobiles. So, it is easy for an attacker to submit them to a hostile environment. The third and the important reason is that a successful attack may allow access to confidential and sensitive data of users. For all these reasons, the designers and developers of smart cards are very aware. They pay very attention to the safety of their products, especially, when attackers of security systems are constantly developing new techniques to access information.

Smart cards are devices made to ship potentially critical data and are therefore intended to be inviolable. As example, a case of an attack on a smart card is to determine the value of its cryptographic key by exploiting system vulnerabilities. To avoid such attack, we should ensure that the system is not vulnerable. The verification process is done through a series of tests performed on a real smart card. Our contribution is particularly interested to this axis. Our generated abstract tests contribute to verify not only the functional requirements of the system but also to ensure that there are no vulnerabilities in the EMV protocol. This verification was possible thanks to our proposed methodology of model-based testing.

In this paper, we propose a new approach for testing payment application based on EMV specifications using model based testing integrated into a SysML development process. We chose SysML language because is the language the most suited to the modeling of Embedded Systems including smart cards and it has demonstrated its strength in modeling systems and generating tests based on models.

The paper is organized as follows: the second section presents an overview of Europay-Smartcard-Visa specifications. The third section lists the various tools used to establish our methodology. In the fourth section, we discuss our approach and steps of our methodology.

II. OVERVIEW OF EMV

In this section, we provide a high-level of overview of EMV specifications contained in the volume entitled: The Integrated Circuit Cards (ICCs) compliant with the Europay-Visa-Mastercard (EMV) specifications [2].

A. EMV

EMV is a standard for payments with Integrated Circuit Cards (ICC), i.e credit cards that incorporate a smart chip. In the 1990s, Europay, MasterCard and Visa took the initiative for EMV. Currently, EMV standard is maintained by EMVCo.

The EMV specifications define a set of rules that impose minimum functionalities to implement and how to implement them. Four books present EMV specifications [2]:

- Book1: Application Independent ICC to terminal
- Book2: Security and Key Management
- Book3: Application Specification
- Book4: Cardholder, Attendant, and Acquirer

In our work, we are particularly interested in application specifications defined by the book3 and security and key management presented in the book2, especially, transactions between card and terminal in terms of Application Protocol Data Unit (APDU) commands.

B. EMV card

All smartcards follow the ISO/IEC 7816 standard [9] while, most of smartcards in banking or credit cards adhere to the EMV Standard [2] called EMV cards. In our work, we treat EMV card that interacts with terminal to ensure payment transaction. The terminal sends APDU command to the card, and the card responds with a response. Thus, the interaction between terminal and card is done through a set of commands/responses sent by terminal and card respectively.

An EMV Transaction uses the following APDU commands:

- SELECT APPLICATION: this command enables the terminal to know the Applet Identifier (AID) in the card with which the terminal wants to interact.
- GET PROCESSING OPTIONS: used to inform the card of the beginning of the transaction, provides the necessary information to the card in order to start the transaction.
- READ RECORD: enables terminal to read some data from the card which are in the form of records.
- GET DATA: the terminal uses this command to find a specific data element from the card.
- INTERNAL AUTHENTICATE: this command is to authenticate the card. The terminal sends the appropriate data as argument. The smartcard had to encrypt or sign these data to prove the secret key.
- VERIFY: concerns the cardholder authentication. It provides the PIN code to verify it.
- GENARATE AC: allows generating Application Cryptogram (AC) by the card. There are three types of cryptogram: TC, ARQC and AAC.

Our approach is based on these APDU commands to model communication between EMV card and terminal. Then, generate vulnerability test cases of the system. The following section describes the tools used to achieve our objective.

III. TOOLS USED TO APPLY OUR APPROACH

A. SysML language

SysML (System Modeling Language) is a general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a large range of systems. It reuses a subset of UML2.0 and defines additional extensions by using UML's profile mechanism [7]. SysML allows to graphically construct an engineering model system.

B. Rodin Plateform, UML-B and ProB Animator plugins

The Rodin is an open source platform and an Eclipse-based IDE for Event-B which provides effective support for refinement and mathematical proof. It contributes to the Eclipse framework and is further extendable with plugins like UML-B.

UML-B [3] is a graphical formal modeling notation which relies on Event-B for its semantics. It is a plug-in for RODIN toolkits and implemented by the Eclipse Modeling Framework (EMF)[5]. UML-B has its own meta-model and provides tool support, including drawing tools based on UML and a translator to generate Event-B models[6]. If the UML-B drawing is saved and has no error, the translator generates automatically the corresponding Event-B model.

ProB is an animator and model checker for the B method. It allows automatic animation of B specifications. In addition to B method, ProB support now even Event-B method. ProB can be integrated in Rodin platform to animate models and easily generate domain specific graphical visualizations.

C. Altova UModel

UModel provides a SysML profile to support SysML stereotypes. The profile can be inserted manually through Project/Include menu option or let UModel profile add the SysML profile automatically when the first SysML diagrm is created in a new project.

D. Event-B formal method

Event-B is derived from B method. It keeps the classical B-method [8] concepts and adds the event concept. Event-B models present well-defined syntax, concepts and semantics and it is possible to test them by proving that transitions (events) made during the software process are correct. In this context, an Automatic generation of vulnerability tests for the Java Card byte code verifier has been performed [11]. To ensure the correctness of a model, Event-B uses proof obligations which are supported by Rodin automated proof tool [4].

IV. RESULT AND OUR NEW APPROACH

As part of our research, we are interested in security and vulnerability tests which aim to ensure that the behavior rejected by the model is also rejected by its implementation. We started by modeling EMV transaction using SysML language. We chose SysML because it is easier to express requirements with SysML diagrams. Indeed, starting with formal models in complex task, so we used less formal models, then, we proceeded to the generation of Event-B model usable to automate tests generation.

Our methodology is divided into three main steps detailed in the diagram below

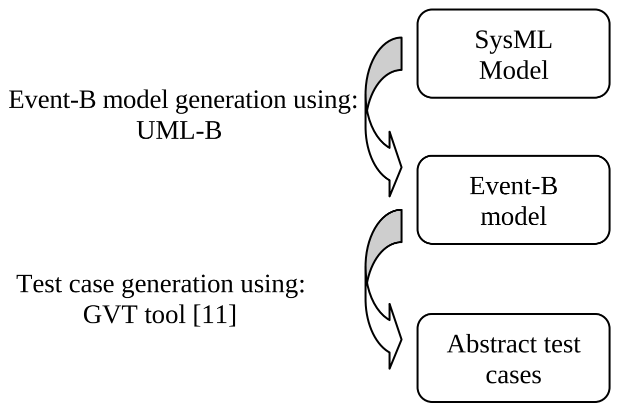
SysML Model
Event-B model generation using: UML-B
Event-B model
Test case generation using: GVT tool [11]
Abstract test cases

Figure 1. Diagram illustrating our approach

A. *Modeling EMV Transaction using SysML language*

We assume that the customization phase of the Cartea already been done. In our work, we treat the use phase of the EMV card represented by EMV transaction. In order to perform a successful EMV Transaction, six steps to follow:

- Application Selection: The first APDU command is sent by the terminal to select the application. This command is SELECT APDU used to select the wanted applet present in the card using its AID (Applet IDentifier).
- Initialization of EMV Transaction using the GET PROCESSING OPTIONS command. It makes the card ready to communicate with the terminal.
- Card Authentication: There two types of card authentication: Static authentication and dynamic one. For static authentication, the terminal signs static data, containing in the card, using a public key to prove the card legitimacy. While dynamic authentication allows terminal to authenticate the card dynamically. So, the card must renew the signature for each transaction. Then, the signature depends on the transaction and remains unique for each one.
  - Cardholder authentication: the cardholder enters his Personal Identification Number (PIN) through the terminal. The terminal sends this PIN code to the card and verify it with VERIFY command APDU.
  - Cryptogram generation: since authentication of both the card and the cardholder are performed, the necessary parameters are initialized. The terminal decides to send GENERATE APPLICATION command. The data field of this command APDU contains Transactions amount, currency and terminal capabilities [10].

Based on EMV specifications [2], we propose our model using SysML language. Indeed, we created Machine state diagram which details the EMV transaction process from initialization to completion of the transaction (see figure 1).

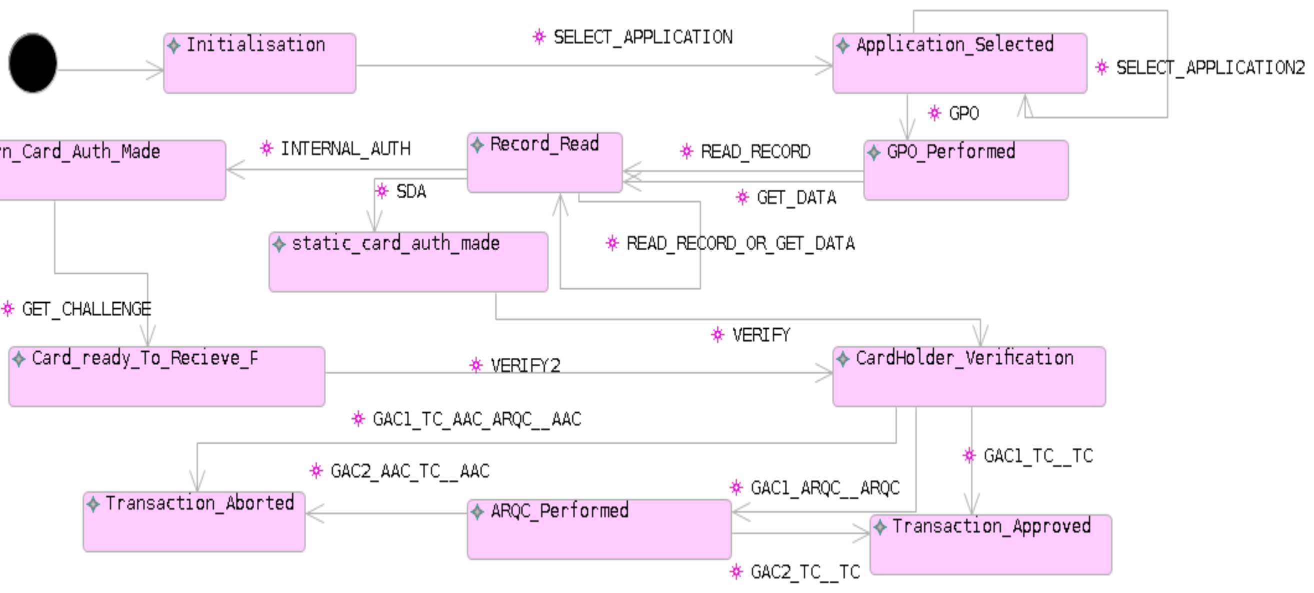

Figure 2. Machine state diagram modeling EMV Transaction

States present the state of the system Terminal-card, and transitions are APDU commands. For the following transitions:

- GAC1_TC_AAC_ARQC__AAC: GAC1 means the first call of GENERATE AC command. It takes as parameter either TC, AAC or ARQC cryptogram and returns AAC cryptogram
- GAC1_ARQC__ARQC: the first call of GENERATE AC command with ARQC cryptogram argument. It returns ARQC cryptogram.
- GAC1_TC__TC: the first call of GENERATE AC command. TC cryptogram as argument and as return value TC.
- GAC2_TC__TC: the second call of GENERATE AC command. TC cryptogram as argument and as return value TC.
- GAC2_AAC_TC__AAC: the second call of GENERATE AC command. It takes as parameter AAC or TC cryptogram and returns AAC cryptogram

To understand the meaning of different types of cryptograms namely TC, ARQC and AAC, the table below clarify the meaning of each cryptogram.

Type	Abbreviation	Meaning
Application Authentication Cryptogram	AAC	Transaction declined
Authorization Request Cryptogram	ARQC	Online authorization requested
Transaction Certificate	TC	Transaction approved

Table 1. GENERATE AC Cryptogram types

B. *Event-B model generation*

Using Rodin interface, we created our Machine state diagram; we specified for each transition (APDU Command) parameters, guards and actions. Then, we generate Event-B model by exploiting the opportunities offered by the UML-B plugin.

The generated Event-B model is as follows:

```
event SELECT_APPLICATION
    where
      @Statemachine1_isin_Initialisation
      Statemachine1 = Initialisation
    then

      @Statemachine1_enterState_Applicati
      on_Selected Statemachine1 ≔
      Application_Selected
  end

  event GPO
    where

      @Statemachine1_isin_Application_Sel
      ected Statemachine1 =
      Application_Selected
    then

      @Statemachine1_enterState_GPO_Perfo
      rmed Statemachine1 ≔ GPO_Performed
      end

  event SELECT_APPLICATION2
    where

@Statemachine1_isin_Application_Selected
Statemachine1 = Application_Selected
  end

  event READ_RECORD
    where

      @Statemachine1_isin_GPO_Performed
      Statemachine1 = GPO_Performed
    then

      @Statemachine1_enterState_Record_Re
      ad Statemachine1 ≔ Record_Read
      @READ_RECORD.Action1 success ≔ TRUE
  End
```

Since the formal models are successfully generated, we verified guards, conditions, parameters and context of models. We moved to animating models with ProB. So we validated our model thanks to model checking provided by ProB plugin.

C. *Test cases generation*

To generate vulnerability tests, we proceeded to the negation of successive constraints of the model. This method is considered by GVT [11]. Below a part of generated abstract tests:

```
<extended_test_suite>
   <test_case>
    <step name="SELECT_APPLICATION">
      <value name="C">Card1</value>
      <value name="CA">ca1</value>
     <modified  name="Statemachine1">
           Application_Selected
      </modified>
      <modified name="success"> TRUE
      </modified>
    </step>
    <step name="GPO">
      <value name="CA">ca2</value>
     <modified name="Statemachine1">
              GPO_Performed
      </modified>
    </step>
    <step name="READ_RECORD">
      <value name="CA">ca1</value>
     <modified name="Statemachine1">
               Record_Read
      </modified>
    </step>
<extended_test_suite>
```

This test case presents the test of the following commands APDU serie:

SELECT APPLICATION → GET PROCESSING OPTIONS → READ RECORD

This APDU command presents a functional test. So, the EMV card should return positive response to this test. For the vulnerability test, we can mention the following vulnerability test which presents the negation of functional requirements:

SELECT APPLICATION → GAC1_TC__TC

According to EMV specifications, the APDU command GENERATE AC (GAC1_TC_TC) should be sent after the card authentication and the cardholder verification. Therefore, sending cryptograms with the GENERATE AC command APDU Immediately after the SELECT APPLICATION without going through the previous steps is a vulnerability test. This vulnerability test is shown below:

```
</test_case>
<step name="SELECT_APPLICATION">
      <value name="C">Card1</value>
      <value name="CA">ca1</value>
      <modified name="Statemachine1">
      Application_Selected
      </modified>
       <modified   name="success"> TRUE
      </modified>
</step>
<step name="evt_GAC1_TC__TC_9_6_EUT">
      <value name="CA">ca2</value>
      <value name="R"> APDU_Response1
      </value>
      <modified name="Statemachine1">
      Transaction_Aborted </modified>
      <modified name="eut"> TRUE
      </modified>
  </step>
 </test_case>
```

The step `evt_GAC1_TC__TC_9_6_EUT` presents the GENERATE AC command. It takes as input and output the TC cryptogram.

V. CONCLUSION

The SysML model proposed as state machine diagram is easy to understand for anyone familiar with EMV standard and clearly shows the steps of real EMV transaction between terminal and card. Usually, terminal-card specifications are spread over the four books [2]. So, we proposed simple models that gather the specifications in term of EMV transaction between Terminal-Card. Then, we generated formal models with well-defined concepts and semantics using Event-B method. The formal models are animated with ProB to ensure their correctness. Next, we generated abstract test cases based on constraints negation using Vulnerability Test Verifier. These tests will allow us to test that our system is functional and at the same time it is not vulnerable.

Our goal in the future steps is to generate concrete test cases based on our abstract test cased already generated. We will execute these tests on real EMV card and seeing the response returned by the card. We will simulate the terminal as a simple application that interacts with the card.

REFERENCES

[1] IMS Research, The World Market for Smart Cards and Smart Card Ics, janvier 2010.

[2] EMVCo. Book 1-2-3-4 - Application independent ICC to Terminal Interface requirements, 4.3 edition, November 2011.

[3] Snook, Colin and Butler, Michael(2008) UML-B: A plug-in for the Event-B tool set. In Abstract State Machine, B and Z, First International Conference ABZ 2008.

[4] Butler, M., and Hallerstede, S. (2007). The Rodin Formal Modelling Tool. In Proceedings of the BCS-FACS Christmas 2007 Workshop – Formal Methods In Industry, London, United Kingdom, BCS.

[5] Tossaporn Joochim. Thesis of Bringing Requirements Engineering to Formal Methods: Timing diagrams for Event-B and KAOS. February, 2010

[6] Abrial, J.-R. (1996). The B-book : Assigning Programs to Meanings, Cambridge University Press.

[7] Vanderperren, Y., and Dehaene, W. (2005). UML 2 and SysML: an Approach to Deal with Complexity in SoC/NoC Design. In Proceedings of the Conference on Design, Automation and Test in Europe (DATE05), Munich, Germany, IEEE Computer Society.

[8] Schneider, S. (2001). The B-method : An introduction, Palgrave Macmillan.

[9] ISO/IEC, ISO/IEC 7816: Identi_cation cards Integrated circuit cards.

[10] Julien Lancia, “fuzzing framework for smartcards: application to EMV protocols. SSTIC 2011.

[11] Savary, A., Frappier, M. and Lanet, J., “Automatic Generation of Vulnerability Tests for the Java Card Byte Code verifier”, Network and Information Systems Security (SAR-SSI) conference, 2011. DOI:10.1109/SAR-SSI.2011.5931379. may 2011.