Formal Methods for the Development of Smart Card Security Applications

Neha Chopra
M. Tech*
BMSCE
Sri Mukatsar Sahib, Punajb
chopra.ecb@gmail.com

Mamta Garg
M. Tech
BMSCE
Sri Mukatsar Sahib, Punajb
mamtagarg104@gmail.com

***Abstract*— Security over the years has been a source of concern to organizations and companies. This has caused quite a significant amount of capital being budgeted for improvements on security systems, simply because it has been discovered that the access control system mechanism is an important part of an organisation. One of the important security systems in building security is door access control. The door access control is a physical security that assures the security of a building by limiting access to the building to specific people and by keeping records of such entries With the emergence of smart cards, industry has become more interested in techniques to establish the correctness and security of the applications developed. Typical smart card applications like door locks, electronic purses and health care information holders contain privacy-sensitive data. This paper does not deal with security aspects like data leakage, but investigates functional behavior and possible abnormalities such as null pointer exceptions. Proper functional behavior and safety properties are a first step towards secure applications. This paper provides a general introduction to the state-of-the-art of formal methods for the development of safety-critical systems. The idea is to combine two program verification approaches: the functional verification at the source code level and the verification of high level properties on a formal model built from the program and its specification. For functional specification we have used PROMELA and for verification SPIN is used in this paper.**

Keywords—formal methods, formal specification, formal verification, Promela, SPIN.

I. INTRODUCTION

Over the years, several security measures have been employed to combat the menace of insecurity of lives and property. This is done by preventing unauthorized entrance into buildings through entrance doors using conventional and electronic locks, discrete access code, and biometric methods such as the finger prints, thumb prints, the iris and facial recognition. In this paper, a prototype door security system is designed to allow a privileged user to access a secure keyless door where valid smart card authentication guarantees an entry. In software engineering, formal methods are mathematically based techniques and tools for the synthesis (i.e. development) and analysis of software systems. Formal methods can be applied at various points through the software development cycle. Formal methods can also be used in reverse engineering to model and analysis existing systems. This paper will focus on formal methods for the specification of functional requirements and design, and for validation/verification which are the most common forms of use of formal methods. The use of formal methods is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical modeling and analysis can contribute to the correctness of the resulting product. However, it should be noted that the use of formal methods does not miraculously guarantee correctness, but can be used to increase the level of correctness. Using formal methods in the specification and design phases implies not only that more flaws are found, but also that they are found already in these earlier phases rather than in the testing or maintenance phases. This is also important factor as the cost of repairing flaws is much higher in the later phases than in the earlier phases, e.g. the investigation reported in. Formal methods are usually only used in the development of safety, business, and mission critical software where the cost of faults is high.

In this paper we have given the system description in section 2. In section 3 we discussed about formal specification. A section 4 deal with study of formal verification and section 5 has the simulation results of the given model. In section 6 we have conclusion of the given study.

II. SYSTEM DESCRIPTION

In this paper the construction of a simple model to represent a building fitted with a card operated access system is given. For this purpose we have defined the processes required to model the door lock, card readers and users. After that we have combined these

processes to create a simulation of a building with a single room accessed through a single door with a card operated lock and a card reader on each side.

Explanation of how the location of each user is represented is given in this model. Simulation takes place in this model using SPIN and shown that when authorized users show their card to a reader, the door opens enabling them to pass through, but the door will not open for unauthorized users. Simulation also illustrates how the locations of users get updated as a result of users passing through doors.

III. FORMAL SPECIFICATION

A specification is a description of a product (either to be build or existing). Specifications are used in many different engineering disciplines including software engineering. In software engineering the products that are specified are software. Associated with the notion of a specification, there is the notion of what it means for a product to satisfy (fulfill/meet/conform to/be compliant with) its specification.

A most common practice in software development is to use informal specifications, i.e. specifications written in natural language or using some diagrams or pseudo code. A complement to informal specification is formal specification. A specification is said to be formal if it is expressed in a formal notation or a formal specification language, i.e. a language that has a precise syntax (e.g. given by a BNF grammar) and for which every sentence in the language has a unique mathematical meaning. The underlying mathematical concepts are often simple, e.g. being based on mathematical logic and set theory.

Formal specifications are, just as informal specifications, used in the analysis and design phases of the software development cycle to record requirements and design decisions, respectively. They can be used as contracts or communication media between customer and developers, and between developers. Besides being used as a base for design and implementation, formal specifications can also be used as a base for generating test cases, for simulation and for formal analysis of the described products in order to predict their behavior before they are implemented.

We have used a client server kind of architecture to represent the problem. Here the card reader process will be client and there will be a common server. A global channel will be used by the card readers to communicate with the server. The server will respond to each card reader on a private exclusive channel which will be passed by the card reader to the server at the time of request. The overall design abstraction is given below in the code for server process is given as:

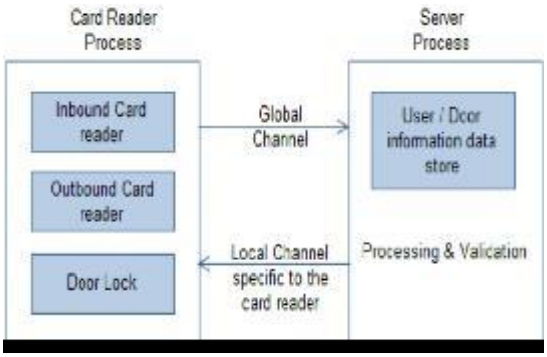

Fig. 1 Design abstraction of doorlock system

```
#define NO_USERS                    2

/* Please specify NO_DOORS one higher than
required as we want to avoid using the 0th

element of array */
#define NO_DOORS                    2
/* Please specify NO_USRCAT one higher than

required as we want to avoid*/
/* using the zeroth element of array */
#define NO_USRCAT                   2

/*1 - Cat Allowed Users */ mtype = { request, deny,

grant};

typedef Array { byte element[3]; };

typedef doors {
bit inbound[NO_USRCAT]; bit
outbound[NO_USRCAT];
};
chan global = [0] of { mtype, chan, bool, byte, byte };
proctype Server()
{
chan readerChannel; Array users[NO_USERS];
doors doorStatus[NO_DOORS]; byte i, readerID,
usrID;
bool inwards = false; bool allow = false;

d_step{
initializeData()
}
end:    Do
::      global?request(readerChannel,      inwards      ,
readerID,usrID) ->atomic{
d_step{
```

```
allow = false; i = 0;
do
:: (i < NO_USERS ) ->
}
}
```

The code for cardreader in Promela is given as:

```
proctype cardReader(byte readerID; byte usrID; bool
inwards)
{
chan me = [0] of {mtype}; bool doorLock = true;
/* int time = 0; */ end: do
::global!request(me, inwards, readerID, usrID) ->do
::me?deny ->break
::me?grant->doorLock = false; wait: doorLock = true;
break;
}
```

It is a simple process which takes initialization parameters. Ideally there should not be any parameters in this case. But we have used them to set input conditions so that we can initialize different card readers with different parameters. This enables the successful verification of the model with zero unreachable states. The process passes a channel defined as me to the server when communicating to it through the global channel. Later on this channel is used by Server to communicate with the card reader. We have modeled the door lock with a bool parameter doorLock which defines the status of door at a given moment. If doorLock = true this means that the door is locked and vice versa. The two card reader associated with a process are shown through the bool inbound parameter. If inbound = true this means that the request to be send is from the inbound card reader, otherwise the request is from outbound card reader. We could have also used a separate process to model door lock and could have updated its status through a dedicated channel attached to the card reader.

IV. FORMAL VERIFICATION

A classical approach of formal verification consists in building a model of the system in a formal framework, for instance a theorem proven language, and target properties are proved to be satisfied by this model. This approach can be found for instance in industrial domains, when formal methods are used to increase the security level of products. A model of a given sensitive system is usually built from the system requirements specification. Security policies can then be translated into security properties and proved in the same formal framework. In this approach, the implementation is generally developed in parallel with the verification process. Therefore the main problem is to justify the link between the verified model and the implementation. This correctness of the model with respect to the source code is mandatory to claim that the code verifies the target properties. A usual way to strengthen this link is to refine the high level model in lower level models, until a low level model whose link with the code is as straight forward as possible, in terms of data structures and functions. The links between two levels are proved using an abstraction property. There are two majorstate-of-the-art approaches to formal verification: theorem proving and model checking.

SPIN is a generic model checking tool that supports the design and verification of asynchronous process systems. SPIN verification models are focused on proving the correctness of *process interactions*, and they attempt to abstract as much as possible from internal sequential computations. Process interactions can be specified in SPIN with rendezvous primitives, with asynchronous message passing through buffered channels, through access toshared variables, or with any combination of these. In focusing on synchronous control in software systems, rather than synchronous control in hardware systems.

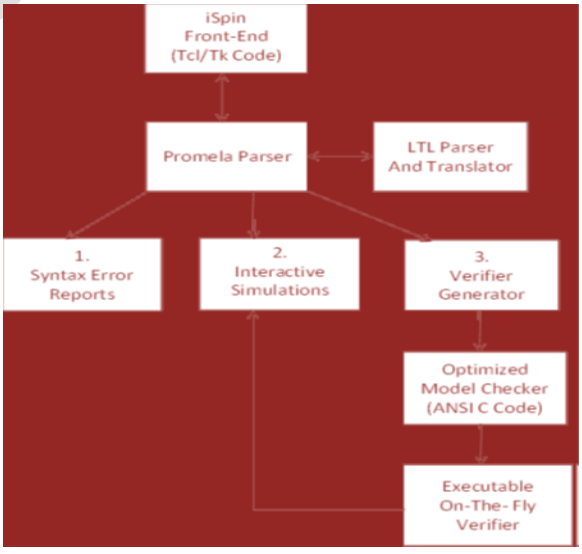

Fig.2 Basic structure of SPIN

The basic structure of the SPIN model checker is illustrated in Fig. 2. The typical mode of working is to start with the specification of a high level model of

a concurrent system, or distributed algorithm, typically using SPIN's graphical front-end iSPIN. After fixing syntax errors, interactive simulation is performed until basic confidence is gained that the design behaves as intended. Then, in a third step, SPIN is used to generate an optimized on-the-fly verification program from the high level specification.

V. SPIN SIMULATION

As a formal methods tool, SPIN aims to provide:
1)An intuitive, program-like notation for specifying design choices unambiguously, without implementation detail.
2)A powerful, concise notation for expressing general correctness requirements, and
3)A methodology for establishing the logical consistency of the design choices from 1) and the matching correctness requirements from 2).

SPIN PARAMETERS USED FOR SIMULATION

S.No.	Name of parameter	value
1.	Safety	
	(i) Invalid endstate (Deadlock)	+
	(ii) Assertion Violation	+
2.	Storage Mode	Exhaustive
3.	Search Mode	Depth First Search(Partial Order Reduction)
4.	Never Claim	-
5.	Hash Factor	1.5
6.	State Vector Size in Bytes	512
7.	Minimum Search Depth	100
8.	Maximum Search Depth	10000
9.	Number of Local CPU-Cores	4
10.	Maximum Memory Per Run	512M

SIMULATION RESULTS

S.No.	Name of parameter	value
1.	State Vector	60 Bytes
2.	Depth Reached	21
3.	State Stored	11
4.	Atomic Steps	6
5.	Hash Conflicts	0
6.	Tripping	0

VI. CONCLUSION

In the smart cards world where security and performance are the main business criteria, formal verification activity becomes a mandatory step. Building high level models of the system being developed to prove correctness properties is useful but is still expensive, as it requires experts. Moreover, a formal link between the models and the actual system implementation is lacking. The goal is then to build tools generating secure code from verified high level models. But those tools have to be improved to take into account the scarce resources of smart cards. Another immediate solution is to reason directly on the source code. This method could be handled by the developer, but reasoning at this low level limits the expressiveness of the properties to prove. The method we proposed here allows providing a formal verification at source code level. The originality of our method is to use the program specification already defined in the annotations, to derive a high level model allowing the definition and verification of high level security properties. The model is thus automatically generated from an existing formal specification.

REFERENCES

1.J-R. Abrial. The B-Book: Assigning Programs to Meanings. Cambridge University Press, 1996.

2.Fr´ed´eric Badeau and Arnaud Amelot. Using B as a High Level Programming Language in an Industrial Project: Roissy VAL. In ZB 2005: Formal Specification and Development in Z and B, number 3455 in Lecture Notes in Computer Science. Springer, 2005.

3.Patrick Behm, Paul Benoit, Alain Faivre, and Jean-MarcMeynadier. M´et´eor: A Successful Application of B in a Large Project. In Jeannett M. Wing, Jim Woodcock, and Jim Davies, editors, Proceedings of FM'99: World Congress on Formal Methods, Lecture Notes in Computer Science, pages 369– 387. Springer-Verlag, 1999.

4.Dines Bjørner. New Results and Current Trends in Formal Techniques for the Development of Software for Transportation Systems. In Proceedings of the Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'2003), Budapest/Hungary. L'Harmattan Hongrie, May 15-16 2003.

5.Christel Baier and Joost-Pieter Katoen. Principles of Model Check- ing. The MIT Press, 2008. Special issue on the Mondex challenge. Formal Aspects of Comput- ing, 20(1), 2008.

6.John Fitzgerald and Peter Gorm Larsen. Modelling Systems – Prac- tical Tools and Techniques in Software Development. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, Second edition, 2009.

7.S. Gerhart, D. Craigen, and T. Ralston. Observations on industrial practice using formal methods. In Proceedings of the 15th Inter- national Conference on Software Engineering, pages 24–34. IEEE Computer Society Press, April 1993.

8.Anne E. Haxthausen, Marie Le Bliguet, and Andreas A. Kjær. Modelling and Verification of Relay Interlocking Systems. In Christine Choppy and Oleg Sokolsky, editors, 15th Monterey Workshop: Foundations of Computer Software, Future Trends and Techniques for development, number 6028 in Lecture Notes in Computer Science. Springer, 2010. Invited paper.

9.Till Mossakowski, Anne Haxthausen, Donald Sannella, and Andrzej Tarlecki. Casl, the Common Algebraic Specification Language. In Logics of Specification Languages, EATCS. Springer, 2008.

10.Anne E. Haxthausen and Jan Peleska. Formal Development and Verification of a Distributed Railway Control System. IEEE Trans action on Software Engineering, 26(8):687–701, 2000.

11.S. Rajan, N. Shankar, and M. K. Srivas. An integration of model checking with automated proof checking. pages84–97. Springer- Verlag, 1995.

et, Electronic Purse Applet Certi_cation, inWorkshops on Secure

12.Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal Methods: Practice and Experience. ACM Com puting Surveys, 41(4):1–36, October 2009.

13.L. Aertryck, L. Benveniste, D. Le Metayer, CASTING: A Formally Based Software Testing Generation Method, IEEE Computer Society, Nov. 1997.

14.M. Alberda, P. Hartel, E. de Jong, Using Formal Methods to Cultivate Trust in Smart Card Operating System, In Proceeding of CARDIS'96, pp. 111-132, Amsterdam, Netherlands, Sept. 1996.

15.P. Bieber , J. Cazin, V. Wiels, G. Zanon, P.Girard, J-L. Lan