Security Verification of Smartcard Scripts

Petr Hanáček[1]
hanacek@dcse.fee.vutbr.cz

Abstract: The article deals with the problematics of security verification of the script programs used in smart cards. There is a growing interest for using user-programmable smart cards, because of the independence they offer their users by having an on-card microprocessor programmable in hardware-independent programming language. A good example of such card is a script-programmable card (ScripCard) that is programmed in the Pascal-like script programming language. Script verification ensures that script code can be trusted to preserve the security requirements for the card. Because of difficulty of man-made verification of the script code we propose an automated tool for script verification.

Key Words: security verification, smart card, electronic payment protocols

1 Introduction

Integrated circuit Cards, or smart cards, is a generic term denoting credit card sized computing devices that can do more than just store data (see [8]). Typically they contain a microprocessor and a few applications that can be invoked from the outside using one the ISO 7816 standard commands. This standard specify how the electric connection between a CAD (Card Accepting Device) such as an automatic teller machine, and the smart card works; in particular how an application on the smart card is invoked with some parameters, and how the result is returned, meaning that only data is exchanged. Having applications on the card is ”smart” since it is the logic on the card which decides which data are exported; hence data can be kept confidential in a very secure manner.

Current conventional smart cards support only fixed set of external commands and this set of commands is not extensible. This property makes conventional not very flexible for rapid prototyping of new applications. One of possible solutions is to allow programming of the smart card functions in a higher programming language by the developer.

The programs written in this higher programming language are usually very simple and mostly contain only calling of predefined smartcard library functions. Because of their simplicity they are often called "scripts". One of the implementation of this concept is ScripCard developed on the TU Brno.

ScripCard programming system defines an open smart card application program interface based on a subset of the Pascal programming language (e.g. no floating point functions). The script source code is compiled outside the smart card into the intermediate code, that is loaded into the smart card. A secure virtual machine interprets an intermediate code. Smart card specific libraries such as EEPROM management, security, serial communication and data protection and sharing allow application programmers to develop applications independent of the underlying specific hardware and microcode implementation. ScripCard enables developers to quickly and easily build a variety of applications for smart cards that:

1 Department of Computer Science and Engineering, Božetěchova 2, CZ-612 66 Brno

- are chip independent
- are based on an easy-to-use and widely accepted programming environment
- provide enhanced security
- are capable of running in small memory environments

The ScripCard will enable a wide variety of smart card applications for both businesses and consumers. Examples include putting medical information on a card, using smart cards as a tool for tracking repair work in a complex engineering environment (such as airline or plant maintenance), using smart cards to provide secure access to cellular phones, and creating frequent-buyer programs that automatically provide upgrades at the checkout stand.

One of the more interesting applications for smart cards is electronic commerce. The ScripCard enhances electronic commerce capabilities on the client side, because the user can now keep trusted data - private digital keys, digital certificates, transactional information, and money - on a card in his or her physical wallet. The user is no longer chained to his/her terminal in order to enjoy the benefits of electronic commerce.

One of the important security problems of the ScripCard is the security correctness of the program. Although the algorithms are simple, the possibility of programmer error is high. Thus we need an automated tool that can check the script programs. The security checking typically answers questions like:

- Can be data X on the card changed by the external attacker? (Data integrity checking), or
- Can be data Y on the card disclosed to the outside world? (Data confidentiality checking).

2 ScripCard

ScripCard software architecture is designed to separate the functionality of the Operating System from the communication protocol employed to utilize it. Thus it is relatively easy to change the protocol without the need to rewrite the operating system. Currently a protocol conforming to ISO/IEC 7816-3 T=0 has been implemented.

The ScripCard looks almost exactly like a credit card but is in fact a small computer. The card contains a microprocessor to provide processing capability and memory for storing instructions and data. The card can be broken down into the following functional areas (see Fig. 1):

- EEPROM memory containing script code and long-life data
- RAM memory for temporary storage of data
- Script Virtual Machine (SVM)
- Script libraries

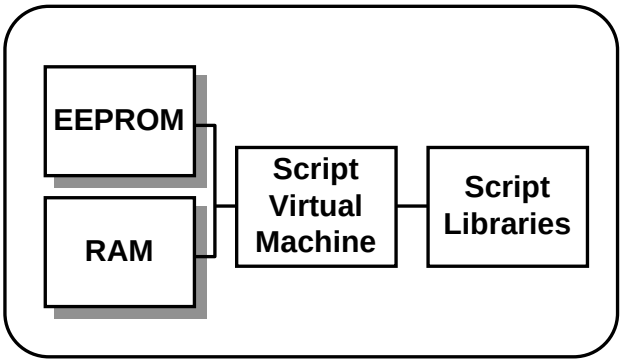


Fig. 1 Internal structure of ScripCard

ScripCards store data in files. All files have several properties. Files have a name assigned to them when they are created. This name is 2 bytes long according to ISO. Files can be also referenced by short identifier. The size of a file is fixed when the file is created. It's size can be anything from 0 ...64K bytes. In practice however the size of file must be a multiple of 8 bytes.

All files in the system have attributes associated with them. The attributes control access to the data contained within the file. Basically there are three types of permission to a file. Permission to read, permission to write and permission to use. Further restrictions may be placed on these activities. Also it may be protected by a password.

The file system of a has a tree structure and can be compared with the file structures of a PC's hard disk. It may contain following file types (see Fig. 2):

- Master File (MF) - also referred to as the root directory,
- Dedicated Files (DF) - also referred to as application directories - containing the smart card applications consisting of data and in some cases also executable programs,
- Elementary Files (EF) - also referred to as data files - containing the actual application data.

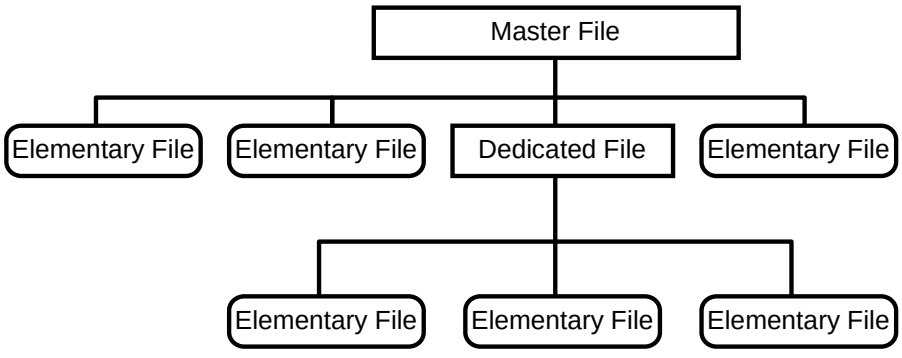


Fig. 2 File system structure of ScripCard

2.1 ScripCard Security

Access to files is secured by cardholder verification values (CHVs). The ScripCard has two CHVs - CHV1 (usually user PIN) and CHV2 (usually Security Officer PIN). The level of protection granted to a file for a given access type (that is read, write, etc.), is defined by

access conditions.

To achieve reliable and secure data transfer data encryption is based on the ANSI DES (X3.92-1981) algorithms, ANSI 3-DES algorithm, SkipJack algorithm (according to FIPS) and proprietary TEAX algorithm. Message authentication is based upon ANSI X9.9-1982.

2.2 Script Language

ScripCard is a smart card that runs programs written in the Pascal-like script programming language. The external script compiler translates the script program into platform-independent target code, that is loaded into the card and then executed by Script Virtual Machine (SVM). The Script Virtual Machine is a stack-oriented abstract machine.

The script syntax is largely derived from the Pascal language, but it also differs from Pascal in significant ways. In the current version the script compiler supports these features:

- Single byte variables and arrays of variables
- Use of program EEPROM for long term storage of variables
- 8 bit arithmetic capability, integers range from -127 to 128 (signed) and 0 to 255 (unsigned)
- Full range of arithmetic functions including logical and bit operations
- Multiple byte arithmetic using arithmetic libraries
- Structured Pascal including conditions, loops, switches, subroutines and user defined functions
- Functions and subroutines may have parameters

Following example (Fig. 3) shows a piece of program written in script language that performs the ISO smartcard command UPDATE BINARY. The right part of the example contains the source code of script program. In the left part is shown the program translated by script compiler into the code that is executed by SVM.

```
                      const
                        INS_UPDBIN  = $D6;
00 4E     SBLOAD $0E      begin
01 1A 00 16 GOTOLONG $0016  case e_INS of
                         INS_UPDBIN:       { ISO command UPDATE BINARY }
                           begin
04 28     Library call #28     CheckLEN8;      { Expected LEN = 8 }
05 2B     Library call #2B     e_P2 := OpenFile;{ Open requested file }
06 02 10   BSTORE $10
08 25     Library call #25     ACKRxStrRes;  { Ack INS and receive data }
09 4F     SBLOAD $0F          if e_P1 and $20 then
0A A0     SBIPUSH $20           EncryptECB;   { Secure Messaging }
0B 0A     BAND
0C 1B 00 10 JZLONG $0010
0F 2F     Library call #2F
10 50     SBLOAD $10          WrXEEBlock(e_P2);{ Write the data }
11 2D     Library call #2D
12 22     Library call #22     ExitOK;        { Exit with SW 9000 }
                           end;
13 1A 00 1E GOTOLONG $001E
16 19 01 D6 SWITCH $01
19 00 04   $0004
1B 00 1D   $001D
1D 20     Library call #20   else DefaultProc; {Pprocessing of other command.}
                         end; {Case}
                       end.
```

Fig. 3 An example of translated script

3 Script Security Verification

The aim of the script verifier is to check the security properties of script programs, not their functionality. Checking of functionality would mean to check only meaningful combination of inputs for the script. The task of verifier is more complex - its task is to check all possible inputs of script and get information about all possible side effects.

Script verification is inherently a control flow and data flow analysis problem applied to individual ISO commands that are processed by script program. The run of script program is a sequence of executed instructions, defined by the control flow diagram of script code. These instructions perform operations on the stack, on the memory, and they are executing operating system library functions.

Because the script commands are allowed to directly modify any place in the any memory of the smart card, the access control mechanism of the smart card can be bypassed. The verifier must ignore the file structure of the smart card and its access control attributes and it must perform the analysis on the memory level. The file is for the verifier only view of some portion of memory.

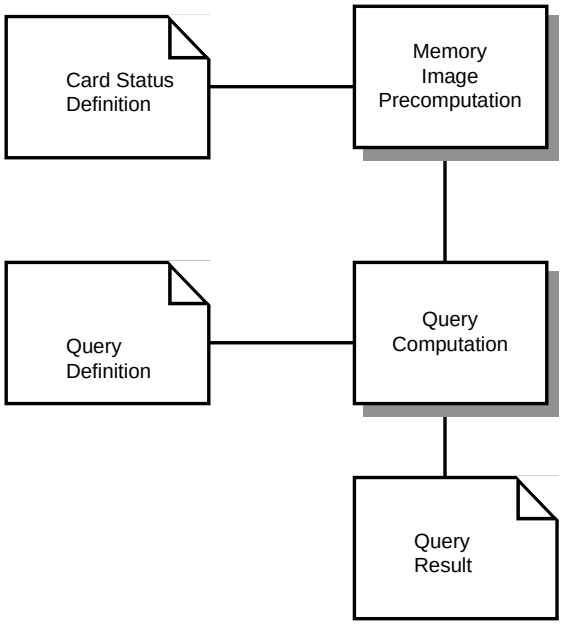
The security verification of the script is performed in several phases (see Fig. 4). In the first phase is processed the card status definition file, that describes the initial smart card status, the structure of files and directories, the content of card files, the attributes of files, the values of CHV and cryptographic keys. The result of this phase is so called memory image. Memory image is an image of all memory locations of the card (i.e. RAM and EEPROM) that for every byte of memory contains a set of possible values and other attributes. The card status definition is written in the special card status definition language. The description of the syntax and semantics of this language is beyond of scope of this article.

In the next phase of verification are answered the verification queries. The query specifies the ISO command that is analysed (or set of commands) and data integrity condition or data

confidentiality condition that is checked.

The fundamental steps of verification are transformations of the memory image according to the control flow graph. For every instruction on the control flow graph is computed a transfer function that describes the appropriate transformation of the memory image. There are three kind of transfer functions: (1) Transfer functions for SVM instructions, (2) transfer functions for calling of library functions, and (3) transfer functions for non-standard events (e.g. undefined instruction, stack overflow and underflow).

The verifier has to check all feasible paths in the control flow graph of every ISO command. Depending on the query also all possible combinations of analysed ISO commands should be analysed. An important and not completely analysed problem is when halt the analysis. Several approaches to this problem are currently examined. Although this problem seems to be very serious, the special nature of script programs makes the program not so hard. The script programs are usually very simple and usually contain no loops (i.e. in the control flow graph usually are no backward arcs). After a small number of iterations usually the memory image stops to change and further analysis is not necessary.



4 Conclusion

The designed security verifier of script programs for user programmable smart cards is on our institute still the subject of several research activities, including some student projects and diploma thesis. All these projects follow the above defined system architecture and share the important common interfaces (e.g. syntax of definition files and output results). The project differ in the method of the implementation of the verification engine that allow us to compare different approaches to this problematics.

This work has been supported by the Grant Agency of Czech Republic grants No. 102/98/0552 " Research and application of heterogeneous models".

References

1. Security of Electronic Money, Report by the Committee on Payment and Settlement Systems and the Group of Computer Experts of the central banks of the Group of Ten countries, Basle, August 1996
2. Anderson, R.: 1993, Why cryptosystems fail, First ACM Conference on Computer and Communications Security, Fairfax, VA, pp. 215¾227. A shortened version of this paper appeared in Communications of the ACM, 11/94.
3. Cousot, P. and Cousot, R.: 1977, Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, POPL å77Œ4th Annual ACM Symposium on Principles of Programming Languages, ACM Press, Los Angeles, California, pp. 238¾252.
4. Garfinkel, S. and Spafford, G.: 1997, Web Security & Commerce, Risks, Technologies, and Strategies, OåReilly & Associates, Inc.
5. Goldberg, A.: A Specification of Java Loading and Bytecode Verification, Proceedings of 5th ACM Conference on Computer and Communications Security, San Francisco, October 1998, (to appear). Kestrel Institute Technical Report KES.U.97.1, December 1997.
6. Lindholm, T., Yellin, F.: The Java Virtual Machine Specification by Addison-Wesley, 1996, ISBN 0-201-63452-X.
7. Qian, X., Goldberg, A.: Bounds Analysis by Abstract Interpretation. First Workshop on Automated Analysis of Programs, Paris, France, 1997.
8. Rose, E.: Towards Secure Bytecode Verification on a Java Card, DIKU, University of Copenhagen, September 14, 1998
9. The ISO 7816 Specification Parts 1-6
10. The EMV '96 (Europay, MasterCard, Visa) Integrated Circuit Card Specifications for Payment Systems.