Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

The ultimate control flow transfer in a Java based smart card

Guillaume Bouffard [a,b], Jean-Louis Lanet [c,*]

[a] Computer Science Department, University of Limoges, 123 Avenue Albert Thomas, 87060 Limoges, France
[b] ANSSI, SGDSN, 51, boulevard de La Tour-Maubourg, 75700 Paris 07 SP, France
[c] INRIA, LHS PEC, 263 Avenue Général Leclerc, 35042 Rennes, France

ARTICLE INFO

Article history:
Received 7 July 2014
Received in revised form
16 December 2014
Accepted 18 January 2015
Available online 7 February 2015

Keywords:
Java Card security
Control flow transfer
Countermeasures
Evaluation
Fault tree analysis
Smart card
Logical attack

ABSTRACT

Recently, researchers published several attacks on smart cards. Among these, software attacks are the most affordable, they do not require specific hardware (laser, EM probe, etc.). Such attacks succeed to modify a sensitive system element which offers access to the smart card assets. To prevent that, smart card manufacturers embed dedicated countermeasures that aim to protect the sensitive system elements. We present a generic approach based on a Control Flow Transfer (CFT) attack to modify the Java Card program counter. This attack is built on a type confusion using the couple of instructions jsr/ret. Evaluated on different Java Cards, this new attack is a generic CFT exploitation that succeeds on each attacked cards. We present several countermeasures proposed by the literature or implemented by smart card designers and for all of them we explain how to bypass them. Then, we propose to use Attack Countermeasure Tree to develop an effective and affordable countermeasure for this attack.

1. Introduction

A smart card can be viewed as a smart and secure device container which stores sensitive assets. It shall ensure a secure data exchange with the reader. Due to the sensibility of the assets contained in the smart cards, they are often the target of attacks. Security issues and risks of these attacks are ever increasing and continuous efforts to develop countermeasures against these attacks are sought. This requires a clear understanding and analysis of possible attack paths and methods to mitigate them through adequate software/hardware countermeasures. Often countermeasures are designed in a bottom-up approach, in such a way that they cut efficiently each attack path. The drawback of this design is to multiply the countermeasures. We propose here to use a top down approach to mitigate the attack by protecting the asset instead of blocking the attack path, having thus, a global approach for the design of the countermeasures.

Control Flow Transfer (CFT) is a technique exploited by an attacker to execute malicious code. The main idea is to modify the return address of a program with different techniques such that, while the program ends the current function, it transfers the control to the address set by the attacker. The return address is often stored in the stack and the attacker is able to access (read/write) to this memory location.

* Corresponding author.
E-mail addresses: guillaume.bouffard@ssi.gouv.fr (G. Bouffard), jean-louis.lanet@inria.fr (J.-L. Lanet).
http://dx.doi.org/10.1016/j.cose.2015.01.004

On smart cards, to execute a CFT attack, two approaches can be used: ill-typed applications and well-typed applications. With ill-typed applications (Iguchi-Cartigny and Lanet, 2010), the input file has been modified in order to illegally obtain information. To be executed, the code must be downloaded without any verification for example byte code verification or any static rule checkers. Therefore, all these attacks are only possible on development cards. Well-typed applications can also be split into two parts: permanent or transient. The first one (Mostowski and Poll, 2008), relies on some weaknesses of the specification but are now well understood and all modern cards have enough countermeasures. Transient well-typed application is a new research field (Barbu et al., 2010; Bouffard et al., 2011) where an application is correct while passing through validation test, static analysis or any rule checker but becomes ill-typed at the execution time. Well-typed application based attacks use fault injection which modifies dynamically the behavior of the application, they are often called combined attack.

Often software attacks are running on one or a couple of cards. In this paper, we present a generic software attack which runs on all the evaluated cards. In the set of evaluated cards, there are recent cards issued from major European smart card manufacturers. The attack is based on an ill-typed application, so it does not pass byte code verification process. A second version is proposed, which is a well-typed application, uses a variant of the attack proposed by Bouffard et al. (2011) to be transformed into an ill-typed application at runtime. To bypass specific byte code verifier we propose in a third version to use polymorphic code. This attack demonstrates that most of ill-typed software attacks can be improved to becomes transient well-typed application easily. This attack brings to the fore that the protection must focus on the assets and not on the attack path.

The rest of the paper is organized as follows. First, the Java Card security is presented in the Section 2. As this platform contains critical assets to protect, we explain in the Section 3, the abused mechanisms to obtain smart card assets and how to prevent that. Based on the CFT approach, we propose in the Section 4, a generic attack which is evaluated in the Section 5. Finally, the Section 6 presents a generic approach to protect our assets and the Section 7 concludes this paper.

2. Java Card security

Java Card is a kind of smart card that implements the specification Java Card 3 (Oracle and Java Card 3 Platform, 2011) in one of the two editions Classic Edition or Connected Edition. Such a smart card embeds a virtual machine that interprets codes already stored in the ROM area with the operating system or downloaded after issuance and stored in EEPROM area. Java Card which is a subset of Java technology, uses the same principles. One compiles the Java code to get the class file, one converts the class file into CAP (Converted APplet) file and then the program is executed using the Java Card Virtual Machine (JCVM). The CAP file is a more compact format designed to reduce the size of the applet image downloaded into the card and to minimize runtime memory requirements. The Java Card platform is a multi-application environment where the sensitive data of an applet must be protected against malicious access from another applet or from the external world. For this reason, the ability to download code into the card is strictly controlled by a protocol defined by GlobalPlatform (GlobalPlatform, 2011). If mutual authentication succeeds, it is possible to load new applications into the card. Loading application into a card is only possible for the one who owns the authentication keys as specified in the GlobalPlatform specification (GlobalPlatform, 2011). It is often done under the responsibility of the operator, which in turn must ensure that the candidate program is trustful. So, the security of the system is ensured by the platform (it has the adequate countermeasure), by the application (it has been coded according to the design rules) and by the issuer through a certification process.

2.1. *Security architecture*

Smart cards security depends on the underlying hardware and the embedded software. Embedded sensors (light sensors, heat sensors, voltage sensors, *etc.*) protect the card from physical attacks. While the card detects such an attack, it has the possibility to quickly erase the content of the EEPROM. This would enable to preserve the confidentiality of secret data or blocking definitely the card (card is terminated). In addition to the hardware protection, software are designed to securely ensure that applications are syntactically and semantically correct before installation and also sometimes during execution. They also manage sensitive information and ensure that the current operation is authorized before executing it.

The Byte Code Verifier (BCV) ensures the type correctness of code, which in turn guarantees the Java properties regarding memory access. For example, it is impossible in the Java-language to perform an arithmetic operation on reference. Thus, it must be proved that two elements on top of the stack are associated to primitive types before performing any arithmetic operation. On the Java platform, byte code verification is invoked at loading time by the loader. Due to the fact that Java Card does not support dynamic class loading, byte code verification is performed at the installation time, *i.e.*, before loading the CAP onto the card. However, most of the Java Card smart cards have not an on-card BCV as it is quite expensive in terms of memory consumption. Thus, a trusted third party performs an off-card byte code verification and signs it. On card, the digital signature is verified.

The firewall performs dynamically checks to prevent applets from accessing (reading or writing) data of other applets. When an applet is created, the system uses an unique Applet IDentifier (AID) from which it is possible to retrieve the name of the package in which it is defined. If two applets are instances of classes from the same Java Card package, they are considered belonging to the same context. The firewall isolates the contexts in such a way that a method running within a context cannot access any attribute or method of objects belonging to another context unless it explicitly exposes features via a Shareable Interface Object. Thus, at runtime, the interpreter verifies that the context of an accessed object is equal (or compatible) to the current context. Under some circumstances, the context can be different, *i.e.* the runtime has