ISSN 22773061

Evaluation of Detection System of Fault Attacks based on Neural Network into a Java Virtual Machine

Ilhame El Farissi[1], Mostafa Azizi[1], Jean-Louis Lanet[2], Mimoun Moussaoui[1]
1MATSI LAB, ESTO, Mohammed First University, Oujda, Morocco
ilhame.elfarissi@gmail.com
azizi.mos@gmail.com
m.moussaoui@ump.ma
2SSD Team Xlim, University of Limoges, Limoges, France
jean-louis.lanet@unilim.fr

ABSTRACT

The Java Card technology provides a secure environment for developing smart card application based on Java while also respecting some constraints such as the limited memory and processing card. In addition to the security and cryptography APIs offered by the Java Card technology, the smart card is protected against some threats. But, the fault attacks based on the variation of the physical parameters are able to disrupt its operation. In order to enhance the smart card security, we thought to add an intelligent component able to distinguish between the smooth functioning and the attack. This component is a Neural Network that we developed in C language and integrated in open source Virtual Machine (Avian) in order to simulate the attack effect and the network behavior. In this context, the detection rate of the attacks is 96% with no false positive.

Indexing terms/Keywords

Detection, Classification, Mutant, Java Card, Neural Network

Academic Discipline And Sub-Disciplines

Artificial Inteligence, Security

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY
Vol 10, No 3
editor@cirworld.com
www.cirworld.com, member.cirworld.com

1. INTRODUCTION

Developed by Bonech, DeMillo and Lipton [1], the fault attack aims to disrupt the physical environment of the processor operation in order to produce errors. For smart card, the context of execution concerns several variables. Some of these variables are internal, such as the location of the different component of the chip (CPU, RAM, ROM, EEPROM). Others are external such as the signals provided by the terminal (clock, I/O) or the physical environment. Initially, the fault attack targeted public-key cryptographic algorithms such as RSA and DES. However, this type of attack can affect every code fragment. The fault attack can be induced into the chip by different ways, namely, variations of the card supply voltage, electromagnetic emissions, and optical fault induction. In the context of vulnerability analysis on smart cards, the use of the optical induction of errors in the chip through laser beams is widely spread, mainly because it allows attackers to be sufficiently precise with regards to both timing and location of the fault injection. Also the use of electromagnetic radiation generator is spreading nowadays to induce faults in chips. Designing efficient countermeasures, both in term of coverage or memory footprint, against these attacks is today a real challenge.

As part of our research, we are interested particularly in smart cards like Java Card. Being a subset of Java technology, Java Card runs by the same principles:

- Compile the java code to get the file .class or .cap,
- Execution of the .class or .cap file by a virtual machine,
- Runtime environment,
- Programming APIs.

In brief, following the Java Card applet compilation in a Java Virtual Machine and Java Card environment, the executable file .cap or .class which contains the byte code equivalent to the initial program is obtained, and then is loaded in the card to be executed by the Java Card Virtual Machine(JCVM). During the execution, the code pointer interprets the instructions sequentially, using the operand stack. The fault attacks targeting the Java Card, aim to modify data or code. It can affect one or more instructions to execute, the returned value of a method, invert the result of a test, suppress a method call,... This class of attack can be used, for example, to avoid the PIN verification by replacing the instruction corresponding by the instruction 'nop' equivalent to 'nothing'.

During our research, we seek to ensure that the executed code corresponds to the developed one by putting a fault detection system based on Neural Networks; test this one on a Java Virtual machine, in order to bring it subsequently, in a smart card system.

In this paper, we present the fault attacks and their effect on a Java Card. In a next section, we will discuss the existing protection mechanisms. Then we present our solution on the integration on a Neural Network into the main loop of the interpreter. In a last section we evaluate our proposal, thanks to a mutant generator framework.

2. FAULT ATTACKS

In this section, we will give an overview over actual physical methods to induce faults. This will show that there are numerous ways to induce faults into physical devices. Fault attack is an old research field. Researches in avionics or space travel brought to the fore that cosmic rays can flip single bits in the memory of an electronic device. Such faults are still an issue until now for such devices (c.f. the Curiosity Rover). In the smart card field, researches focused on power spikes, clock glitches and optical attacks. A smart card is a portable device without any own power supply neither clock and thus requires a smart card reader providing power and clock in order to work. The reader can be replaced by an adversary with laboratory equipment, able of tampering with the power supply. With short variations of the power supply, which are called spikes, one can use it to induce errors into the computation of the smart card. Spikes allow to induce both memory faults but also faults in the execution of a program. The memory cells, i.e., EEPROM memory and semiconductor transistors, have been found to be sensitive to light. This happens if the photon energy of the applied light is transformed in electron in the semiconductor. Modern green or red lasers can be focused on relatively small regions of a chip, such that faults can be targeted fairly well. The last method use changes in the external electrical field and is considered as a method for inducing transient faults into smart cards. Here, faults are sought to be induced by placing the device in an electromagnetic field, which may inflence the transistors and memory cells. Faults can be induced into the chip by the perturbation of its execution environment. Consequences of fault attacks can be perturbation of the chip registers (e.g., the program counter, the stack pointer,...), or the writable memories (variables and code modifications). These perturbations can have various effects, and in particular, they can allow an attacker to gain illegally access to data or services if not detected. In the literature [3,4,5,6], we can find different manners to produce fault attacks but currently laser beam attack is the most difficult to tackle with.

The process is the following, an attacker physically injects energy in a memory cell to change its state. Thus and up to the underlying technology, the memory physically takes the value 0x00 or 0xFF. If memories are encrypted, the physical value becomes a random value (more precisely a value which depends on the data, the address and an encryption key). Then the attacker observes the effect of the fault characterized in terms of temporal and spatial parameters. If the result did not provide him any valuable information he restarts the process.

Smart card manufacturers have been aware of the danger of faults for long time now, hence, they have developed a large variety of hardware countermeasures. Major hardware countermeasures are sensors and filters, which aim to detect attacks, e.g., using anomalous frequency detectors, anomalous voltage detectors, or light detectors. Other countermeasures are to use redundancy, i.e., dual-rail logic, where memory is doubled, doubled hardware, capable of computing a result twice in parallel. If two results are computed, they are considered to be error-free if both values match. This is a very expensive countermeasure, and hence, it is not often implemented in smart cards. Using only hardware countermeasures has two drawbacks. Highly reliable countermeasures are very expensive and low cost countermeasures only detect specific attacks. Since new fault attacks are being developed frequently these days, detecting only currently known forms of physical tampering is not sufficient and for long term applications (an e-passport must be valid for 10 years) it is definitely not sufficient.

Software countermeasures are introduced at different stages of the development process; their purpose is to strengthen the application code against fault injection attacks. Current approaches for software countermeasures include checksums, randomization, masking, variable redundancy, and counters. The most efficient are those implemented in the system which harden the system by checking that applications are executing in a safe environment. The main advantage is that the system and the protections are stored in the ROM, which is a less critical ressource than the EEPROM and cannot be attacked thanks to checksum mechanisms that allow to identify modification of data that are stored in the ROM. Thus, it is easier to deal with integration of the security data structures and code in the system.

3. MUTANT APPLICATION

The purpose of the fault attack is to disrupt the card behavior. The fault attacks targeting the Java Card, aim to generate mutants by modifying one or more instructions in order to execute what the attacker wants instead of the original program. This class of attack can be used, for example, to access to services without the required credentials. There are many types of mutants and multiple mechanisms allow their detection partially. The mutant generation and detection is a new research field introduced by [15] using the concepts of combined attacks. To define a mutant application, we use an example on the following debit method that belongs to a wallet Java Card applet. In this method, the user PIN (Personal Identi cation Number) must be validated prior to the debit operation.

```
private void debit (APDU apdu ) {
if ( pin.isValidated( )){
// make the debit operation
} else {
ISOException.throwIt(SW_PIN_VERIFICATIO
N_REQUIRED) ;
}
}
```

Listing 1: Original Java code

The corresponding byte code representation is the following table, with on the left the byte before the attack and on the right the resulting code while a laser hit the memory cell corresponding to the ifeq instruction.

Byte	Byte Code	Byte	Byte Code
00 : 18	00 : aload_0	00 :18+	00 : aload_0
01 : 83 00 04	01 : getfield #4	01 : 83 00 04	01 : getfield #4
04 : 8B 00 23	04 : invokevirtual #18	04 : 8B 00 23	04 : invokevirtual #18
07 : 60 00 3B	07: ifeq 59	07 : 00	07 : nop
08 : ...	08 : ...	08 : 00	08 : nop
09 : ...+	09 : ...	09 : 3B	09 : pop
...	...	...	...
59 : 13 63 01	59 : sipush 25345	59 : 13 63 01	59 : sipush 25345
63 : 8D 00 0D	63 : invokestatic #13	63 : 8D 00 0D	63 : invokestatic #13
66 : 7A	66 : return	66 : 7A	66 : return

Table 1:Byte Code representation before and after the attack

```
private void debit (APDU apdu ) {
// make the debit operat ion
ISOException . throwI t
(SW_PIN_VERIFICATION_REQUIRED) ;
}
```

Listing 2:Mutant Java code

The verification of the PIN code is bypassed, the debit operation is made and an exception is thrown but too late because the attacker will have already achieved his goal. This is a good example of dangerous mutant application: ‘’an application that passes undetected through the virtual machine interpreter but that does not have the same behavior than the original application’’. This attack has modified the control flow of the application and the goal of the countermeasure described in this paper is to detect when such modifications happen. We propose hereafter to improve the protection techniques against the mutants by adopting artificial intelligence methods like the Neural Network which is able to detect changes made to the system.

4. EXISTING COUNTERMEASURES

The detection of such mutant applications is a field that has been mainly tackled in the scope of Ahmadou Séré's PhD. thesis [7]. As pointed out in [16] , the first countermeasure against mutant applications is either to modify the value associated to the nop instruction in order to make 0 an impossible byte code or at least to double-check the read byte code when it turns out to be 0. The following describes the other detection methods exposed in the field-of-bit method, the basic-block method and the path-check method.

4.1 The Basic Block Method

This mechanism is based on code division to elementary blocks. For each one, we have to precise the input, the output and the checksum. The checksum is the result of the XOR operation between all the bytes of the bloc. Stored in a table, the information is compared with the new checksum, recalculated during the byte code interpretation. If they are not identical, we deduce that it is an attack and the program stops.

The method consists then in statically determining the basic blocks composing each method defined in the method component of the CAP file and computing a checksum for each basic block. The authors propose to use a simple XOR as checksum operation. They subsequently add a custom component made of a table containing for each basic block:

- The offset of the entry point in the byte code array representing the methods,
- The offset of the exit point in the same byte code array,
- The value of the XOR checksum.

At runtime, the JCVM is then responsible for dynamically computing the basic blocks, updating incrementally the XOR checksum and checking the coherency with the stored entry point, exit point and checksum. That is to say, it should check:

- When a basic block is entered, if the entry point is known (i.e. is in the table);
- When a basic block is leaved, if the exit point is known and matches the last seen entry point and if the checksum is correct.

Although this method leads to a relatively high computing overhead, it would detect any single-byte error in the byte code array.

4.2 The field of bit detection mechanism

The idea is based on the fact that the modification of a byte in a methodâ ™s byte code array is likely to modify its nature. For example, an instruction byte becomes a parameter one or the reverse. The Field of Bit Detection Mechanism consists in associating a field of bit to each byte, knowing that the bit value depends on the nature of the byte: either executable (X) or readable (R). Then, during the class execution, the mechanism checks the consistency between the bit table and the instructions interpreted. The principal drawback of this method is then that it will not detect a fault if it does not modify the nature of a byte, such as turning an sload_1 into an aaload for instance.

4.3 The Path Check Method

The last proposed method enhances the previous basic-block method. This method also uses the notion of basic blocks but integrates them in the Control Flow Graph (CFG) representing the execution flow of a given method where each vertex correspond to a basic block and each oriented edge to a jump from one basic block to another. The method proposed consists then in statically computing and encoding the valid paths in the CFG with the following convention:

- The tag 01 denotes the beginning of a path;
- An edge is denoted 0 if it links a basic block ending by a branch instruction to another basic block;
- An edge is denoted 1 if the entry point of the next basic block directly follows the exit point of the current basic block.

The list of valid paths can then be stored in binary format in a custom component of the CAP file. At runtime, the JCVM is then responsible for dynamically computing the basic blocks, encoding the path it executes and checking the path whenever entering in a new basic block. Indeed, if the JCVM produces a path that is not one listed in the custom component, an error is detected.

5. ATTACK DETECTION BY NEURAL NETWORK

5.1 Illustration

Java Card program is compiled off card to get the executable file format; .class or .CAP which will be loaded in the smart card to be interpreted in the JCVM [2]. In order to guarantee the code integrity, we propose to add new options to the card allowing the attack detection. The main detection approaches are the scenario approach and the behavioral one.

The scenario approach is based on the collection of the signatures of the existing attacks in order to exploit them to detect intrusions. But, this approach is unable to detect new attacks.

The second approach consists on testing and evaluating the systemâ ™s behavior. Any deviation is interpreted as a possible intrusion. Thus, the behavioral approach can detect new intrusions.

A fault attack targets the byte code contained in executable file in order to execute what the attacker wants instead of the original program. For that, we propose to verify the code integrity by adopting a behavioral approach. In fact, we thought about identifying the basic blocks of the program, constructing the control flow graph representing all paths possible taken by the program during its execution and putting a system able to verify the path taken by the program during its execution and check whether it is a valid path or not i.e. an attack. Then, the first step consists to divide the byte code into elementary blocks whose entry point does not contain code that is the target of a jump instruction and the exit point is an instruction:

- Which starts a method
- An unconditional branch target (goto, goto_w, jsr, jsr_w and ret)
- A conditional branch target(ifeq, iflt, ifle, ifne, ifgt, ifge, ifnull, ifnonnull, if_icmpeq, if_icmpne, if_icmplt, if_icmpgt, if_icmple, if_icmpge, if_acmpeq, if_acmpne, lcmp, fcmpl, fcmpg, dcmpl, dcmpg)
- A conditional composed branch target (lookupswitch et tableswitch)
- Following the return instruction type (ireturn, lreturn, freturn, dreturn, areturn, et return).

There are many methods which belong the behavioral approache, such as Neural Network and Bayesian Network. In [17], it was proved that the Neural Network is more suitable than Bayesian Network in mutant detection. Consequently, in this paper we aim to test and evaluate the Neural Network.

5.2 Creation of the Neural Network

5.2.1 Neural Network realization

The human Neural Network contains approximately 10^{11} neurons and from 10^{14} to 10^{15} connections. There are several types of neurons, but they all have the following functional properties:

- The neuron receives signals from other neurons or from the external environment.
- It is capable of manipulating the signals and processing the information.
- Its output transmits the information to other neurons.

Inspired from the biological Neural Network, the artificial Neural Network is composed of several interconnected elements capable of receiving processing and manipulating signals, and sending the result. Thus, the artificial Neural Network is used to solve diagnosis problems, prediction, classification ones...

There are several types of artificial network. Based on the simple Perceptron, the multilayer Perceptron (MLP) network use intermediate layers called also hidden layers and located between the input and the output, that ameliorate the network computational capacity and allow obtaining best result [8].

The strengths of the Neural Network reside in learning, objects identification and a better interpretation. Because of that, there has been a reason for using it to detect intrusions [9]. Since, a strong and intelligent Neural Network is not necessary large, we thought to implement it into a smart card in order to detect the fault attacks. This is what we have done previously when we have used the physical parameters of the card to aliment the network. Theoretically, the principle of implementation is possible and the system designed in [17] is able to detect any changes made in the electrical properties. But practically, these parameters are under the control of the attacker. Therefore, it is necessary to change the entry points of the network.

5.2.2 Learning algorithm

Learning is the mechanism by which the free parameters (learning weights) of a Neural Network are adapted and modified in order to obtain network outputs which matches the desired ones. The kind of learning is determined by how the parameters changes are implemented, it is the role of the learning algorithm. There are several types such as the Heaviside function, the sign function and the back-propagation one which minimizes the error in a remarkable degree [8]. After determining the Neural Network type and the learning algorithm, it is necessary to specify the network architecture, namely, the inputs, the output, the number of neurons in the hidden layer and the synaptic weights.

5.2.3 The input neurons

As regards the network inputs and since the fault attacks disrupt the program execution by changing the byte code, we affected the vertices of the CFG to the network's inputs. Precisely, we used the numbers of the first instructions of the vertices.

5.2.4 Training and validation patterns

During the learning phase, the network adjusts the synaptic weights, based on models supplied to the inputs, so that the output obtained corresponds to the desired one. Concerning the patterns used, we deduce the possible paths of program execution from the control flow graph and we generate the corresponding patterns (we called them the normal patterns). And we consider that any path which is different from the normal patterns is an attempted attack. Then, we deduce the abnormal patterns corresponding to attacks. Consequently, we used the normal patterns generated from the control flow graph to learn the network the valid paths. A part of the abnormal patterns has been used for training and another one for the network validation.

5.2.5 Output neuron

The objective of our system is to detect if the entry is a valid path or an attack. For that, we used a single neuron in the output layer that takes the value '0 'in the normal case and '1' otherwise.

5.2.6 Hidden neurons

The hidden or intermediate neurons are a primordial element in Neural Network of MLP type. In fact, they reinforce its computational capacity. But, there is no rule to determine the number of hidden neurons. For that and following the same approach as described in [17] and after several attempts. We opted for the network with a single hidden layer containing three neurons.

5.2.7 Synaptic weights

The correction of the connections or learning weights is made according to the difference between the desired and obtained outputs. The algorithm back-propagation, based on the sigmoid function, allows minimizing the error [8].

5.2.8 Network construction

Since neuroph package provides the ability to create several types of Neural Network including MLP and operate the back-propagation algorithm to determine the synaptic weights. We opted for the use of this package, under the editor eclipse, in order to build the Neural Network, to learn it the different patterns and return the final synaptic weights.

5.2.9 Learning phase of the network

Under the editor eclipse, and using the package neuroph [13], all patterns dedicated to learning are presented, with the desired outputs, once the network. It calculates an output and compares it with the desired one. According to the difference between the two output values, the Neural Network corrects the synaptic weights in order to reduce the error by using the back-propagation algorithm.

5.3 Tests and Results

Once the network is validated, we thought to simulate its operation under a Java Virtual Machine. For that, we have developed the network in C language and we have integrated it in a Java virtual machine open source called Avian [14]. According to the same rules of control flow graph construction, we modified the virtual machine Avian, so as to return just the entry points to blocks executed during the interpretation of the Java class and use it to aliment in the integrated network in the same machine.

After specifying the synaptic weights, it is necessary to validate the network by getting new patterns and verify the detection capability of the network and the error rate. By exploiting the package neuroph under the editor eclipse, we tested the network with new patterns and only 5 among 131 mutants were not detected.

One method often used for evaluating the effect of the fault consists to inject the attack into the system. We talk about the system mutation. Thus, in order to illustrate the suggested protection mechanism and since the Java Card technology runs by the same principle of Java, we used a concrete example with a Java method on a virtual machine. So, using the ASM plugin [11] of the eclipse editor [12], we generated the control flow graph, shown in the following figure, of the Java method that we intend to secure.

We consider that the normal way of interpreting is C1: B1, B2, B3, B4, and all the others paths are considered as an attempt of attack. We designed the Neural Network on this basis.

Since any path different from C1 is an invalid path. We executed the Java class under the modified virtual machine. Thus, we have two cases:

- If the program follows the path C1 in the interpretation, the network returns 0.
- Otherwise, if the program takes a different path, the network returns 1.

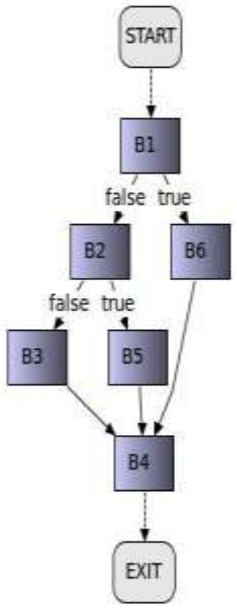

Figure 3: The Control Flow Graph

6. CONCLUSION

In this paper, we propose a new detection approach of fault attacks. In order to achieve this,first of all, we have created a Java class, then we have developed a Neural Network which is able to differentiate between the paths taken by the program during its execution and integrate the network in an open source virtual machine to test it.

According to the study, we found that the Neural Network is able to classify, detect and identify more than 96% of attacks. And as it was possible to integrate the Java virtual machine, we can enhance the security of the system of smart card by a Neural Network. For that, it is necessary to adapt the program developed in C according to the constraints of the Java Card.

REFERENCES

[1] Bonech, D., Lipton, R., 1996. New Threat Model Breaks Crypto Codes: Bellcore Press Release.

[2] Sun Microsystems. 2006. Java Card TM 2.2.2 Virtual Machine (JCVM) Specification.

[3] Skorobogatov, S.P., Anderson, R.J., 2002. Optical fault induction attacks: Springer-Verlag, pp.2—12.

[4] Aumuller, C., Bier, P., Fischer, W., Hofreiter, P., Seifert, J.P., 2003. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. Lecture Notes in Computer Science, pp. 260-275.

[5] Kmmerling, O., Kuhn, M.G., 1999. Design principles for tamper-resistant smartCard processors: WOST'99 Proceedings of the USENIX Workshop on SmartCard Technology on USENIX Workshop on SmartCard Technology, pp.9—20.

[6] Quisquater, J.J., Samyde, D., 2002. Eddy current for magnetic analysis with active sensor. In Proceedings of Esmart, volume 2002.

[7] Sere, A.A., Iguchi-Cartigny, J.-L, Lanet, J.L., 2009. Automatic detection of fault attack and countermeasures : proceedings of the 4th workshop on Embedded Systems Security. ACM., pp. 1-7.

[8] Rennard, J.P., 2006. Neural networks,chapter 4. MultiLayer Networks, Vuibert, ISBN 2711748308, First Edition;

[9] Beghdad, R.? 2008; Critical study of neural networks in detecting intrusions: Computers & security 27, pp. 168—175.

[10] El Farissi, I., Azizi, M., Moussaoui,M. 2012. Detection of smartcard attacks using neural networks: International Conference on Multimedia Computing and Systems (ICMCS), pp. 949—954.

[11] Plugin ASM, http://asm.ow2.org/index.html

[12] Eclipse editor, http://www.eclipse.org/

[13] Neuroph Package, http://neuroph.sourceforge.net/

[14] Open Source Virtual Machine Avian, http://oss.readytalk.com/avian/index.html

[15] G. Barbu, H. Thiebeauld, and V. Guerin. 2010. Attacks on Java Card 3.0 Combining Fault and Logical Attacks. Smart Card Research and Advanced Application, Cardis 2010, LNCS 6035:148--163.

[16] Séré, A.A., Lanet,J.-L. and Cartigny 2011. J. Evaluation of Countermeasures Against Fault Attacks on Smart Cards. International Journal of Security and Its Applications, 5(2):49—61.

[17] Ilhame EL FARISSI, Mostafa AZIZI, Mimoun MOUSSAOUI, Jean-Louis Lanet, Neural Network vs. Bayesian Network to Detect Java Card Mutants, 2013 AASRI Conference on Intelligent Systems and Control, AASRI Procedia, Accepted.