Global Information Assurance Certification Paper

SANS Security Essentials GSEC Practical Assignment
version 1.3

"Java Smart Cards Are Here To Stay: Benefits And Concerns"

Sonia Otero

February 16, 2002

Introduction
The combination of smart cards plus the Java Card architecture (Java smart cards) could potentially provide millions of consumers with multiple beneficial applications in terms of confidentiality, authentication, integrity and non-repudiation. Nevertheless, as with any new technology, they have stirred up a bit of controversy due to possible privacy issues and lack of a yet broad based understanding and infrastructure. However, smart Java cards seem to be bound to gain momentum and become a permanent fixture of our routine life. In this paper, I will describe the extensive security layers involved in this powerful combination, as well as their vulnerabilities. I will show that benefits seem to outweigh the disadvantages, since certain sectors of society have already accepted the risks.

What is a smart card?
A smart card is a credit card sized device, embedded with an integrated circuit chip providing memory storage and computational power. It is equipped with tamper-resistant features, protected from mechanical stress (i.e. bending) and the effects of static electricity. There are 3 types of smart cards:

- Memory card: Also known as store value cards because they only have memory capacity. They are the oldest and currently most widely used type of smart card; a phone card is an example.

- Microprocessor card: The chip on the microprocessor include RAM, ROM and EEPROM, typically 32KB of memory with an 8- or 16-bit processor. This card can add, delete and manipulate information as well as run applications. It also includes security features such as a Personal Identification Number (PIN), for circumstances where strong authentication or authorization are needed.

- Cryptographic card: A more sophisticated microprocessor specialized in cryptographic operations (manipulation of large numbers), such as hashing, digital signatures, private key capabilities.

Smart cards, unlike magnetic stripe cards, can carry all pertinent functions and information on the card. Therefore, they do not require access to remote databases in order to complete transactions.

The International Standards Organization (ISO) 7816 established industry standards of interoperability among smart cards, software and readers, so that a

smart card from a certain vendor works on a reader of another vendor. Although full compliance has not been reached, it should be noted that most of the key manufacturers (Oberthur, Schlumberger, Gemplus) are working towards this goal.

From the manufacturer to the cardholder, the production of a smart card goes through different phases, each with its own quality control:

1. Fabrication: The chip manufacturer produces the silicon integrated circuit chip. A fabrication key (KF) is added to protect the chip from fraudulent modification until it is assembled into the plastic card support. The KF is unique and derived from a master manufacturer key.

2. Pre-personalization: The card supplier mounts the chip on the plastic card and connects it appropriately with the printed circuit. For secure delivery of the card to the card issuer, the fabrication key is replaced by a personalization key (KP). Also, physical memory access instructions are disabled to protect modification of the fabrication area. From this point on, only logical memory addressing is available.

3. Personalization: The card issuer completes the creation of logical data structures and the loading of data files and applications. The card holder identity, PIN, and unblocking PIN are stored as well. A utilization lock is written to indicate the card is ready to be used.

4. Utilization: The cardholder enjoys full functionality while the card is activated, following the stipulated security policies.

5. End-of-life: It could be initiated by an application by writing an invalidation lock, or by an irreversible block produced because both the PIN and unblocking PIN become blocked. In the former situation, writing is disabled by the operating system leaving reading privileges for analysis purposes. In the latter one, all operations (writing and reading) are disabled. In most smart card operating systems, the PIN will be blocked after a fixed amount (around 3) of invalid PINs are presented consecutively.

In terms of storage, a smart card can be viewed like a disk drive with a hierarchical filesystem, where there is one master file (MF) under which there can be elementary files (EFs) and subdirectories called dedicated files (DFs) with their own EFs. However, to provide greater security control, each file is enhanced with access control attributes, ranging from no restriction to PIN-verification to forbidden access.
The smart card infrastructure requires the presence of a reader or card acceptor device (CAD), which serves as the power supply, and a terminal or host, which serves as an extension of the computational or functional power for the application, like a friendlier input/output interface. The communication between the chip and the CAD is a 9600-baud bi-directional half-duplex serial line, while the communication between the CAD and the host could be a 56K-baud or

faster dedicated serial line.

How is Java involved?
Java Card platform came into the picture 5 years ago as a means to eliminate the problem of multiple, noninteroperable platforms, applying the main advantages of the Java language to a new realm. Java Card is and will continue to be the product of shared experience in an open collaborative forum among the experts in the field, and so it complies with ISO specifications as well as with industry-specific standards such as GlobalPlatform, Europay-MasterCard-Visa (EMV), European Telecommunications Standards Institute (ETSI) and the Third Generation Partner Project (3GPP).

Java Card technology defines a runtime environment (JCRE) on top of the hardware and the smart card native operating system. This runtime environment provides a high-level, standard interface to smart card applications, and as a result, it allows for rapid application development. Java's platform-independence characteristic allows the same application to run on any card incorporating a Java Card interpreter. More than 100 million cards were estimated to have been shipped as of last year.

Obviously, the full-fledged Java 2 set of libraries is far too large to fit on a resource-constrained device such as a smart card. Therefore, Java Card's solution consists of a minimal subset of the Java API plus some special-purpose card commands, which deeply affect programming style and testing concerns.

Java Card has many features familiar to Java developers:
+ Small primitive data types: boolean, byte, short, int (optional)
+ One-dimensional arrays
+ Packages, classes, interfaces
+ Object-oriented features: inheritance, virtual methods, overloading, dynamic object creation, access scope, binding rules.
+ Exceptions

But it does not support other elements of Java:
- Large primitive data types: long, double, float
- Characters and strings
- Multidimensional arrays
- Dynamic class loading
- Security manager
- Threads
- Cloning
- Garbage collection
- Finalization
- Serialization

The Java Card virtual machine (JCVM) is implemented in two distinct pieces, the on-card bytecode interpreter and the off-card converter. Given a compiled

regular class file as input, the converter behaves as the standard class loader counterpart, with the difference that it produces a special file (CAP) per package. Once the CAP file is transmitted to the card, the interpreter simply executes the code found in it controlling memory allocation and ensuring runtime security.

The Java Card Runtime Environment (running inside the card), is responsible for resource management, network communications, applet execution and security. The communication between a Java smart card applet and a host application is performed via Application Protocol Data Units (APDUs), which are data packets containing either a command from the host to the applet or a response from the applet to the host.

From its inception, the Java Card platform was designed to enable multiple applications to reside on a single card plus complement the inherent security features of smart cards' functionality. All operations and transactions are atomic, an all-or-nothing proposition so that incomplete or failed operations are returned to their previous state. Each application is isolated and protected by an applet firewall to ensure their integrity and eliminate program tampering, either by individuals or through program interference. Only one applet can run at a time. Building on the fact that smart cards give the flexibility of updating information as the user's needs change, multiple applets from different vendors can also be updated dynamically for maximum customization after initial issuance.

Security model of a smart card

As explained earlier, unlike conventional computers, processor, I/O, data, programs and network in smart cards may be controlled by several, possibly hostile, parties. There is not a clear boundary. Let's look at the particular threats and risks derived from this situation, keeping in mind that a cardholder acts as if carrying a computer under somebody else's control, and therefore, there is no room for trust because any of the participants might feel inclined to cheat. These are the conventional role splits:

- Cardholder: chooses when and where to use it.
- Data owner: controls the data.
- Terminal: peripheral with which the smart card interacts with the world (keyboard, display).
- Card issuer: controls the operating system and any data initially stored.
- Card manufacturer: produces the physical smart card; could be subcontracted among third parties.
- Software manufacturer: produces the software that resides on the smart card; could also be subcontracted.

Let's imagine an attack by the terminal against the cardholder or data owner. Known as the "terminal problem", this situation is directly caused by the fact that smart cards rely totally on external display capabilities during critical transactions. This display must be trustworthy and unspoofable; otherwise, for

example, a displayed purchase believed to be $10 is authorized by an unwitting customer, resulting on a true charge of $100 in the smart card, due to a corrupt vendor. CADs could contain I/O enhancements, such as LEDs that show values directly read by the smart card owner, though this increases the cost of the smart card. Another suggestion has been to use a Personal Digital Assistant (PDA) to interact with the smart card as a trusted portable reader, but since these devices often connect with notoriously insecure PCs, they are equally vulnerable to similar attacks. In fact, one excellent reason for using smart cards at all is that PCs can not be trusted. Since information stored in the card can be kept completely confidential (PINs, sensitive personal data, private keys) and encryption/decryption can take place on the card itself, the secret data is never transmitted unsafely and is not so easily stolen if the PC is compromised. A possible prevention of the terminal problem is for the back-end system (card issuer) to monitor and detect suspicious behavior between cards and terminals, or better yet to establish a secure connection directly between the card and the back-end processing system through the terminal.

To accomplish secure communication, protocols like SSH and SSL with or without PKI (Public Key Infrastructure) integration have started to provide an interface for smart cards. They are known to ensure both authenticity and secrecy without danger of man-in-the-middle attacks. The client versions ask the user to enter the PIN, which is sent encrypted from the terminal/PC to the card, where once authenticated by the card, allows the user's public key to be retrieved and verified with the corresponding private key.

Same prevention steps should apply in the case of a terminal attacking the card issuer, but what if the card issuer and terminal are owned by the same company joining forces against the cardholder? The problem becomes a matter of institution reputation towards its clients, just as it does in a normal business environment. In general, it is presupposed that the card issuer holds the best interests on behalf of the cardholder, but that is not necessarily the case when we think of privacy invasion attacks, a constant source of concern and alertness. To prevent these kinds of attacks, the system must be carefully designed to maintain the anonimity and unlinkability of the owner with respect to the issuer. A good example of this approach is the fingerprint recognition system chosen by O'Hare International Airport for its existing building access control. Employees will register their fingerprints via a sensor which captures and converts the fingerprints into templates (similar to 40-digit security codes) and stores each individual template on each employee's smart card. Privacy is protected because fingerprints cannot be reconstructed from the template, and fingerprint images are never stored anywhere on the system. Employees insert their smart cards into the readers and place their finger on the optical sensor. The comparison between the template generated by the sensor and what is stored on the card grants or denies access based on the employee's access permissions.

Card issuers are positioned with clear advantage at the time of influencing software updates, where the user lacks freedom of choice without the proper

knowledge or ability to discern the security impact of any change coming from "above". Some updates might even happen without the user's consent.

In commerce transactions, or access controls, the cardholder might not be allowed to know the data, and even if he or she does, it would certainly be prohibited from changing it, since he or she could effectively mint his own money or make copies of his access card. An attack of a cardholder against the data owner implies the construction of a fake card, which is difficult to achieve due to the expensive cost, with tools available only on well funded laboratories. Nevertheless, we need to be aware of certain research successes for the sake of defeating tamper-resistance or accomplishing reverse-engineering. I'm referring to exposing the card to unusual voltage and temperature levels to trap the secret key or alter the random generation capacity to result on keys containing almost all 1's. Other methods involve physically removing the chip from the plastic card and pour nitric acid on it to expose the silicon surface for manipulation, or erasing the security lock by shining ultraviolet light on the EEPROM. A quite complex but widely published algorithm, called Differential Power Analysis (DPA), deals with the measurement of energy activity or power consumption associated with card operations as a means to detect patterns during key encryption/decryption, leading to the discovery of the PIN and secret key. To counteract this approach, smart card manufacturers should include digital noise to mask power consumption, by adding extra calculations into the mix or randomizing the order of card computations, so they do not follow a fixed easy-to-isolate pattern.

Attacks by the cardholder or data owner against the card/software manufacturers are a long shot given the anti-tampering measures taken during the manufacturing process, as explained previously. More worrisome and probable is the reverse situation: manufacturers faced with the traditional challenge of providing a secure kernel. For the last thirty years, we have grown used to the irrefutable and ingrained idea that somebody will always find a successful way of breaking into conventional operating systems, so we cannot assume smart card operating systems would be any less susceptible to exploitable security holes (intentional or not). The vulnerability increases by allowing and encouraging the ability of running multiple programs on the same card. The I/O dependence of the smart card on an external device exacerbates the cardholder's lack of control when it comes to know what program is running when the card is inserted into a terminal, or how to ensure that your program is talking to the terminal and not through another program (i.e. a Trojan horse version). Yet another challenge is to ensure the card would truly invalidate itself (complete memory destruction) if it were able to detect the presence of a hostile environment. The fact that the smart card deals with far more secret or valuable information puts the manufacturer in an tremendously advantageous position.

Having the card stolen is probably the most obvious, frequent and fearsome attack that comes to people's mind, since it is something consumers are most familiar with. However, given the perimeter defense of a smart card, conventional credit card thieves would have a tougher time impersonating the

real cardholder to make fraudulent purchases. Having a PIN, that provokes a self-denial of service on the smart card after several incorrect PIN attempts, as the first layer in this perimeter defense will require thieves to become a lot more sophisticated, as we have previously seen.

Security model of Java Card

I will identify the characteristics that increase or decrease security in a smart card due to Java's involvement. The goal is to prevent any Java bytecode to subvert the interpreter to perform illegal or malicious operations. To begin with, for maximum transparency, Java's source code for the compiler and interpreter is totally available for public examination, subject to all kinds of independent scrutiny, a pivotal premise to achieve and maintain system security.

Java Card lessens security risks by several venues: Java is a very strictly defined language, meaning that all primitive types are guaranteed to be a specific size and all operations are guaranteed to be performed in a certain order. Also, Java forbids pointer arithmetic; there is no way a programmer can access a predetermined memory offset, all methods and instance variables in a class file are accessed via symbolic names.

Furthermore, because Java is a very strongly typed language, its extensive compile-time checking enables it to find bugs earlier by checking that objects are truly of the correct type as required by the methods and variables. The compiler also checks for permission violations classes might incur accessing methods and variables (distinguishing private, protected or public status), and ensures no access to uninitialized variables. The next layer of defense occurs when the virtual machine invokes the class loader, which is responsible for preventing untrusted classes from impersonating trusted classes. The class loader provides a protected unique name-space, such that two equally named classes cannot be loaded into the same name-space. Next, classes go through a bytecode verification process determining that they have the right format: there are no stack overflows or underflows, all memory accesses are valid, all parameters are correct and there is no illegal data conversion.

The absence of dynamic class loading (the ability of programs to acquire new dynamically loaded JVM code and extend their own functionality during execution) on Java Card makes type safety easier to enforce; there is less risk for the virtual machine to get confused about the type of object it is manipulating. It should be noted that many of the standard Java security problems arose from type safety resulting from dynamic class loading. Lack of threading is another advantage, as it makes the code security analysis much easier.

Turning our attention to the factors that increase security risks, Java Card increases security risks or poses special programming challenges by removing what are generally considered good features. Garbage collection is a particularly conspicuous loss. Limited memory capacity on smart cards compounded with the inability to free allocated memory sounds like an invitation

to memory leaks (memory becoming full when objects are inefficiently created and destroyed), which could lead to a denial-of-service attack by rendering the card useless through memory exhaustion. Hopefully, as chips become more powerful, one can expect the incorporation of garbage collection into the Java Card architecture.

Since Java Card allows multiple and possible competitive or cooperative applets to be resident on the same smart card, the potential for security problems is evident. This means we must trust (with a grain of salt) that applet firewalling and memory isolation management is perfectly implemented. An applet firewall confines an applet to its own designated area, preventing access to objects owned by other applets. However, a high-demand beneficial feature in terms of code reuse given the memory shortage, object sharing, has introduced a loophole, by allowing context switching between the currently active applet and the applet whose object needs to be accessed, and the context switching can be nested. The troublesome situation is this: applet A shares its object x with applet B; applet B shares its object y with applet C. A method of object y (shared with C) invokes a method on object x (since B and A allow sharing), so effectively, applet C is indirectly invoking an object of A (with which no explicit sharing had been established). Underneath, all depends on the quality of the underlying JCRE implementation, which enjoys the absolute privilege of being able to access any applet's context, making it an attractive single-point-of-failure target for breaking.

But, by far, the greatest risk lies on the potential ability for a vendor to add and use native code on the platform, which is usually necessary for performance reasons (Java's version is not fast enough). Though an applet would cease to be portable to other cards, all protection mechanisms provided by the Java virtual machine deteriorate or disappear when native code is executed in a lot less restrictive environment. Native method calls would most probably be an attackers' first choice.

Ideas for usage

Endless possibilities. Prices have decreased to the point of less than three dollars a piece, and readers costing just about magnetic-stripe readers do.

- Secure transactions over the web. Netscape and Microsoft are developing APIs for smart card interfaces. The idea is to use a smart card to store cryptographic data for use with existing protocols such as SSL, creating sort of a personal portable identity portal.

- The U.S. Department of Defense plans to issue 4.3 million cards as their access control to physical locations and computer network resources for all military personnel and eligible contractors. They will manage all credential data by implementing PKI certificates accessible through one PIN. It even facilitates email decryption.

Same concept could be applied to IT administrators searching to consolidate all

authentication (and non-repudiation) mechanisms a company has to manage: a picture ID card, PKI certificate, remote access token, a number of static passwords (along with their associated systems, resources). It eliminates the password login by creating an environment favorable for digital signing. Effective against the disgruntled employee.

- Smart mobile phones with built-in financial capabilities (m-commerce): mobile banking, stock trading or gambling.

- The Taiwan government plans to issue 24 million Java smart cards as their new health insurance ID card. Patients can carry their medical history; extraordinary reduction of administrative costs.

- Visa mostly and American Express spearhead the movement towards wallet cards, to integrate credit/debit card payment, loyalty programs or frequent flier miles applications. Service levels could be upgraded remotely transparently securely to cardholders, without having to issue a new card. The biggest motivation is the reduction of credit card fraud.

- Replacement for the paper form of passport and visas. A much more reliable form of identification than visual human inspection, and much harder to forge.

- College students can combine the services of library, gymnasium, dorm access and cash payment cards.

- Contactless mode for mass transit: tolls, parking.

- Set-top boxes attached to televisions with smart cards that not only secure them from piracy but also convert the television into an e-commerce center with a remote control (t-commerce).

- Rental cars: choose one, unlock it with a smart card and drive away without standing in line or filling out paperwork.

Conclusion

It is widely known that passwords are often the weakest link in any security scheme. If for nothing else, smart cards seem the best bet to strengthen this first defense perimeter. Since smart cards are intrinsically secure devices for storing valuable information, they may therefore become a convenient solution in many other desirable areas. It is unreasonable to expect anything to be 100% hacker-proof, and so it is a community requirement to continuously perform extensive testing and analysis of the whole smart card paradigm. As one of the estimated 2 million Java developers in the world, I'm looking forward to observing and benefiting from the growth of this promising development platform.

References

McGraw, Gary, and Felten, Edward. "Securing Java." Wiley Computers

Publishing, John Wiley & Sons, Inc., 1999.

Chen, Zhiqun. "Java CardTM Technology for Smart Cards: Architecture and Programmer's Guide." Addison Wesley, June 2000.

Schneier, Bruce, and Shostack, Adam. "Breaking Up is Hard To Do: Modeling Security Threats for Smart Cards." Proceedings of the USENIX Workshop on Smartcard Technology, May 1999. 175-185.

Elo, Tommi. "A Software Implementation of ECDSA on a Java Smart Card." Master's Thesis, Helsinki University of Technology, April 2000.

Yellin, Frank. "Low Level Security in Java." 1996.
URL: http://java.sun.com/sfaq/verifier.html (21 Jan. 2002).

Itoi, Naomaru, Fukuzawa, Tomoko, and Honeyman, Peter. "Secure Internet SmartCards." Java Card Workshop, Canes, France, September, 2000.
URL: http://www.citi.umich.edu/techreports/reports/citi-tr-00-6.pdf (21 Jan. 2002).

Chan, Siu-cheung Charles. "An Overview of the Java Security." 1997.
URL: http://home.hkstar.com/~alanchan/papers/javaSecurity/index.html (21 Jan. 2002).

Chan, Siu-cheung Charles. "An Overview of the Smart Card Security." 1997.
URL: http://home.hkstar.com/~alanchan/papers/smartCardSecurity/index.html (21 Jan. 2002).

Chan, Siu-cheung Charles. "Electronic Smart Passport/Visa." 1997.
URL: http://home.hkstar.com/~alanchan/papers/smartPassport/index.html (21 Jan. 2002).

Kayl, Kammie. "Java Card (tm) technology turns five: Celebrating Leading Smart Card Technology." 29 October 2001.
URL: http://java.sun.com/features/2001/10/javacardbday.p.html (21 Jan. 2002).

"O'Hare International Airport Chooses Fingerprint Recognition by SecuGen Corp to Beef Up Security." 3 October 2001.
URL: http://www.secugen.com/company/doc/01ohare.htm (21 Jan. 2002).

Dailey, Heidi. "Revolutionizing the United States Military Id Badge." 26 March 2001.
URL: http://java.sun.com/features/2001/03/card.p.html (21 Jan. 2002).

Dunlap, Charlotte. "The Smart Card Glitch." 7 December 2001.
URL:
http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2830921,00.html (21 Jan. 2002).

Last Updated: July 25th, 2017

Upcoming Training

SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
Mentor Session - SEC401	Arlington, VA	Oct 04, 2017 - Nov 15, 2017	Mentor
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event