The Evolution of the JAVA Security Model

Nikolaj Cholakov, Dimo Milev

Abstract: *In this paper the basic security features of the Java platform, including the language-level security, the Java Virtual Machine-level Security, the sandbox model, and the Java Security APIs, are discussed. In search of the proofs that Java is really secure and reliable, the main specialties and advantages of these features are considered and summarized, with special attention to their evolution within the dynamic and fast deployment of the Java platform.*

Key words: *Java, JSDK, Java Security, Security API, Security Model*

INTRODUCTION

Since 1995, when Java was first presented, there has been strong and growing interest around the security of the Java platform. Here are some of the most commonly posed questions in this matter [2]: Is Java secure? Who is at risk? What are the risks? How common are security breaches? Is this problem ever going to go away?

Of course there are no simple answers to these questions, but one thing is very clear: nothing (including Java) is completely secure. A lot of hackers are permanently trying to break out each security system, and they use more and more sophisticated ideas and approaches. So software systems producers have also to improve their products permanently, and make them more reliable, secure and proof to different kinds of attacks.

LAYOUT

From the technology provider's point of view, Java security includes two aspects [3]:

- Provide the Java platform as a secure, ready-built platform on which to run Java-enabled applications in a secure fashion.
- Provide security tools and services implemented in the Java programming language that enable a wider range of security-sensitive applications, for example, in the enterprise world.

Both aspects are organically related and together they ensure the desired security level for the Java platform itself and for all kinds of Java applications.

1. Language-level security

Overall security is enforced through a number of mechanisms, developed at different times. The foundations of the Java security can be seen in some basic language features:

- **Simplified and easy to use**. Java is much simpler in comparison with other languages like C++. Thus the burden on the programmer is smaller and so the probability of making subtle mistakes is lessened;
- **Strictly object-oriented**. There are wrapper classes defined even for the simple data types, and there can be no structures outside classes. Thus all security-related advantages of the object-oriented paradigm can be used;
- **Final classes and methods**. This feature disallows subclassing when applied to class definitions and disallows overriding when applied to method definitions, and prevents the undesired modification of certain functionality;
- **Strongly typed**. Polymorphism is a very powerful object-oriented feature, but it holds potential risks of masking hostile objects. Both the compiler and the runtime checking disallow such possibilities, because no assignment can be made if object types are incompatible;
- **Automated memory management with no direct use of pointers and address arithmetic**. This feature disallows incorrect memory access and minimizes the probability

of memory leaks, unauthorized data access and runtime crashes;

- **Clearly defined behavior to uninitialized variables**. All heap-based memory is automatically initialized. However, all stack-based memory isn't. So, all class and instance variables are never set to undefined values, and all local variables must definitely be assigned before use or the source compiler is obligated to give you an error.
- **Strict exception-handling mechanism**. When using a method, which potentially can cause severe errors, the programmer is forced by the compiler to handle all possible exceptions. Thus the behavior of the Java program is always predictable and the program becomes more “fool-proof”;

2. Java Virtual Machine-level Security

The Java Virtual Machine is responsible for the execution of Java programs, and also for the implementation of a very important part of the Java Security features:

- **Bytecode verification**. Java compilers produce platform-independent bytecode, executed by the Java Virtual Machine. Compilers and a bytecode verifier ensure that only legitimate Java bytecodes are executed. The bytecode verifier, together with the Java Virtual Machine, guarantees language safety at run time;
- **Dynamic class loading**. The Class Loader finds and loads the byte codes for the class definitions. Once loaded, they are verified before the creation of actual classes. The Class Loader also ensures that the JVM isn't tricked into using false representations of the core class libraries-ones that could break the Java security model. Finally, the Class Loader provides separate name spaces for classes loaded from different locations, which prevents untrusted classes to interfere with the running of other programs;
- **Runtime safety checks**. Because of the late binding provided by the JVM, additional late (runtime) type checking is done of assignments and array bounds. Thus the JVM ensures that only properly assignable operations are performed, and no access outside correct array bounds can be realized;
- **Control over the access to crucial system resources**. Each running JVM has at most one SecurityManager installed. The SecurityManager checks in advance every statement and restricts the actions of a piece of untrusted code to the bare minimum;
- **Access Controller and permissions**. Applications cannot install their own Security Managers, but it is possible to give them individual permissions to very specific operations using the policytool program. With the AccessController class the programmer can check if the user has permission to perform a given operation.

The software implementation of the JVM security features is known as "sandbox" in which Java programs can run safely, without potential risk to systems or users. This sandbox evolved seriously since its first version.

3. The evolution of the sandbox model

The evolution of the sandbox model is illustrated on figures 1-3 [3]:

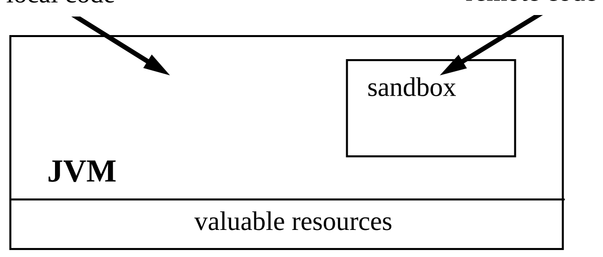

Figure 1. The JDK1.0 Sandbox model

The original sandbox model was deployed through the Java Development Kit (JDK), and was generally adopted by applications built with JDK 1.0, including Java-enabled web browsers. This model provided a very restricted environment in which to run untrusted code obtained from the open network. The essence of the 1.0 sandbox model is that local code is trusted to have full access to vital system resources (such as the file system) while downloaded remote code (an applet) is not trusted and can access only the limited resources provided inside the sandbox.

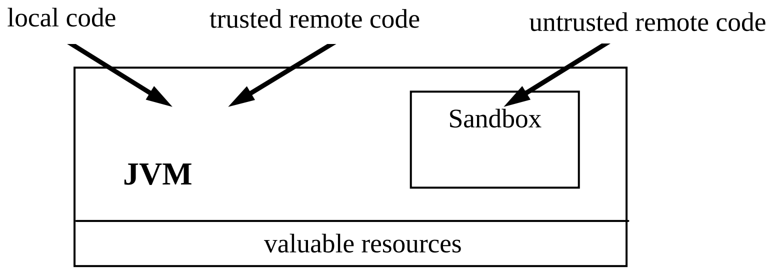

Figure 2. The JDK1.1 Sandbox model

JDK 1.1 introduced the concept of a "signed applet", as illustrated by the figure above. In that release, a correctly digitally signed applet is treated as if it is trusted local code if the signature key is recognized as trusted by the end system that receives the applet. Signed applets, together with their signatures, are delivered in the JAR (Java Archive) format. In JDK 1.1, unsigned applets still run in the sandbox.

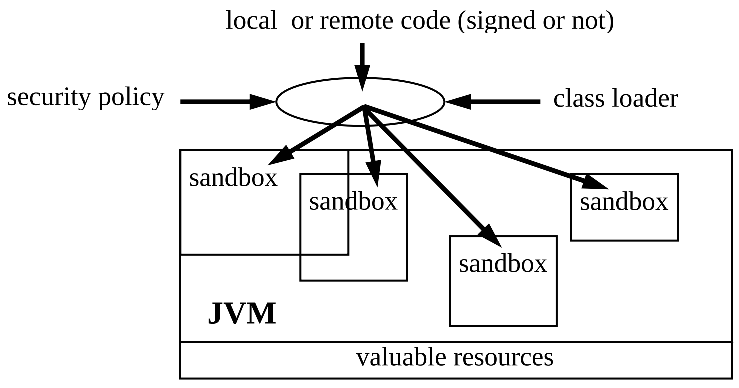

Figure 3. The Java 2 Platform Sandbox Model

The Java 2 Platform sandbox model introduced the capabilities to enhance security to very finely control access and implemented a new philosophy: applications run with different permissions, no built-in notion of trusted code. The Java 2 Platform sandbox ensures:

- Dynamic, extensible security model;
- Fine-grained access control to protect resources;
- Easily configurable security policy enforced by security manager;
- Easily extensible access control structure;
- Extension of security checks to all Java programs, including applications as well as applets.

4. Java Security APIs

In addition to the language and virtual machine features already discussed, Java technology has a number of APIs that provide classes useful for writing secure applications. Some of them first came as optional packages but now are integrated into the Java 2 SDK. These APIs provide a rich feature set, with support for a wide range of tasks:

- **Java Cryptography Architecture (JCA)** [1] is a framework for providing cryptographic capabilities to Java programs. JCA includes support for message digests, digital signatures, key pairs management, authentication and certificates. The design of this framework follows the MVC (Model-View-Controller) model, which separates concepts from implementation. The implementation is delivered from the security provider. JCA comes with a set of algorithms provided by the Sun's standard security provider, but there is also a possibility for pluggable use of third party security providers;
- **Java Cryptography Extension (JCE)** is an extension of the JCA. JCE provides a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. Secure streams and sealed objects are also supported. The Java 2 SDK includes a standard JCE provider named "SunJCE", which comes pre-installed and registered, and as in JCA, providers signed by a trusted entity can be plugged into the JCE framework, and new algorithms can be added seamlessly;
- **Java Authentication and Authorization Service (JAAS)** is a framework for user-based authentication and authorization. JAAS implements a Java version of the standard Pluggable Authentication Module (PAM) framework and can be used for two purposes:

  - authentication of users, to reliably and securely determine who is currently executing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet. JAAS authentication is performed in a pluggable, permitting Java applications to remain independent from underlying authentication technologies;

  - authorization of users to ensure they have the access control rights (permissions) required to do the actions performed. JAAS policy extends the Java 2 policy with the relevant Subject-based information. Permissions recognized and understood in Java 2 are equally understood and recognized by JAAS. Although the JAAS security policy physically resides separately from the existing Java 2 security policy, the two policies should be treated as one logical policy;
- **Java Secure Socket Extension (JSSE)** enables secure Internet communications. It provides a framework and an implementation for a Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols and includes functionality for data encryption, server authentication, message integrity, and optional client authentication. Using JSSE, developers can provide for the secure passage of data between a client and a server running any application protocol, such as Hypertext Transfer Protocol (HTTP), Telnet, or FTP, over TCP/IP. JSSE uses the same provider architecture defined in the JCA.
- **Java CertPath API** comprises classes, methods, and interfaces to build and validate an ordered list of certificates, referred to as *certification paths* (or certificate chains). A Certificate Authority (CA) vouches the digital identity by signing the certificate with the CA's private key. To verify a certificate's identity, the relying party should have a trusted copy of the CA's public key. In the absence of such a trusted copy, a certificates chain, the certification path, is needed, with each certificate vouching for the previous one until a certificate the relying party implicitly trusts is found. Based on the validation results, users can associate a public key with a subject. CertPath API also supports pluggable provider architecture.
- **Java Generic Security Services (JGSS) API** contains the Java bindings for the Generic Security Services API (GSS-API) - a uniform API for mutual authentication of the client and the server and secure exchange of messages regardless of the underlying technology. JGSS offers features similar to these of JSSE, but there are differences in the

underlying security mechanisms. JGSS uses Kerberos version 5 instead of SSL/TLS protocols; the Communications API in JGSS is token-based instead of socket-based; JGSS also uses Credential delegation and Selective encryption, which are unsuitable for JSSE;

- **XML security** comprises **XML Digital Signature API** for parsing, generating, and validating XML signatures according to the W3C Recommendations, and **XML Digital Encryption API** – for standard set of APIs for XML digital encryption services. XML encryption can be used to perform fine-grained, element-based encryption of fragments within an XML document as well as encrypt arbitrary binary data and include this within an XML document.

Usually some of the Security APIs are commonly used. For example JAAS authentication is typically performed prior to secure communication using Java GSS-API. Thus JAAS and Java GSS-API are related and often used together. Of course this is not obligatory: it is possible for applications to use JAAS without Java GSS-API, and it is also possible to use Java GSS-API without JAAS.

5. Java Security features evolution

Some of considered Java security features were introduced with the original version of the JDK, other were included lately, but all of them evolved since their first version. Typical changes include the addition of new interfaces, classes and methods; method updates; improvements of the command-line security tools; provider enhancements, and of course bug fixing.

Table 1 summarizes the evolution of some basic features in the Java security model:

Table 1. Changes in security features

Security Feature	JDK 1.0	JDK 1.1	JDK 1.2	J2SDK 1.3	J2SDK 1.4	J2SDK 1.5
Language Security Features	**O**	**+**	**+**	**+**	**+**	**+**
Java Virtual Machine	**O**	**+**	**+**	**+**	**+**	**+**
Sandbox Model	**O**	**+**	**+**			
Signed content		**O**	**+**	**+**	**+**	**+**
Security tools			**O**	**+**	**+**	**+**
Security policy			**O**		**+**	**+**
Java Cryptography Architecture (JCA)		**O**	**+**	**+**	**+**	**+**
Java Cryptography Extension (JCE)			**O**		**+**	**+**
Java Authentication and Authorization Service (JAAS)				**O**	**+**	
Java Secure Socket Extension (JSSE)			**O**		**+**	**+**
Java Certification Path API					**O**	**+**
Java GSS-API					**O**	

Legend: **O** : the feature is introduced
+ : the feature is enhanced

An important element of the security model advancing is the regular bug fixing and known problem removing. But to keep the users informed about these problems and their fixes is also very important. A chronology of security-related bugs and issues can be found at [5]. An advanced search in this scope can be made using the Sun Alert Notifications at [6].

CONCLUSIONS AND FUTURE WORK

With an eye on the facts mentioned above, the following conclusions can be drawn:

- The Java platform relies on powerful, dynamic and extensible security architecture. This architecture is based on some strong and interconnected standards;
- The Java language features such as automatic memory management, garbage collection, and range checking on strings and arrays help the programmer to write safe code, and to keep more his attention to the logic of the program;
- Numerous security APIs, mentioned above, give the developers means to perform wide range of security-related operations in their applications. The Java2 SDK comes with standard security feature providers holding a set of built-in algorithms for many security-related issues. At the same time the provider architecture, supported by these APIs, allows user programs to use the same API, but with different providers plugged;
- The new releases of the JSDK come with numerous enhancements of existing security features along with new ones; the evolution of the Java Security model advances permanently;
- Every Java security feature, like the whole platform, is very well documented, and the users can easily find actual information about new issues in this matter. The developers can rely on the technology provider's support and assistance at any time;
- Considering all facts mentioned above, the following final conclusion could be formulated: the Java security model grows and improves permanently; the issues of this evolution give the Java platform the opportunity to face the challenges of the today's global information environment.

REFERENCES

1. Knudsen, J. Java Cryptography, O'Reilly, 1998;
2. Wutka, M. Hacking Java. QUE, Indianapolis, 1998
3. Http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc1.html
4. Http://java.sun.com/j2se/1.5.0/docs/guide/security/index.html
5. Http://java.sun.com/sfaq/chronology.html
6. Http://sunsolve.sun.com/pub-cgi/search.pl

ABOUT THE AUTHORS

Nikolaj Ivanov Cholakov, PhD, Department of Information Technologies, "St.st. Cyril and Methodius" university of Veliko Turnovo, Phone: +359 62 649831, E-mail: n.cholakov@uni-vt.bg

Dimo Milev Milev, Department of Information Technologies, "St.st. Cyril and Methodius" university of Veliko Turnovo, Phone: +359 62 649831, E-mail: d.milev@uni-vt.bg