iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2 **Issue:** X **Month of publication:** October 2014

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology(IJRASET)

Java Security

Saurabh Setia[1], Nipun Jain[2], Paras Thakral[3]

DCE, GGN

Abstract :The Java platform provides a number of features designed to improve the security of Java applications Java's security model is one of the language's key architectural features that makes it an appropriate technology for networked environments. Security is important because networks provide a potential avenue of attack to any computer hooked to them. The term "security" is somewhat vague unless it is discussed in some context, so it will be describing that how Java Security can be managed and is efficient.

I. INTRODUCTION

When Java was first released by Sun Microsystems, it attracted the attention of programmers throughout the world. These developers were attracted to Java for different reasons: some were drawn to Java because of its cross-platform capabilities, some because of its ease of programming (especially compared to object-oriented languages like C++), some because of its robustness and memory management, some because of Java's security, and some for still other reasons.

Just as different developers came to Java with different expectations, so too did they bring different expectations as to what was meant by the ubiquitous phrase "Java is secure." Security means different things to different people, and many developers who had certain expectations about the word "security" were surprised to find that their expectations were not necessarily shared by the designers of Java.

II. WHY SECURITY?

The term "security" is somewhat vague unless it is discussed in some context; different expectations of the term "security" might lead us to expect that Java programs would be:

Safe from malevolent programs : Programs should not be allowed to harm a user's computing environment. This includes Trojan horses as well as harmful programs that can replicate themselves-computer viruses.

Non-intrusive : Programs should be prevented from discovering private information on the host computer or the host computer's network.

Authenticated: The identity of parties involved in the program should be verified.

Encrypted: Data that the program sends and receives should be encrypted.

Audited: Potentially sensitive operations should always be logged.

Well-defined: A well-defined security specification would be followed.

Verified: Rules of operation should be set and verified.

Well-behaved: Programs should be prevented from consuming too many system resources.

C2 or B1 certified: Programs should have certification from the U.S. government that certain security procedures are included.

In fact, while all of these features could be part of a secure system, only the first two were within the province of Java's 1.0 default security model. Other items in the list have been introduced in later versions of Java: authentication was added in 1.1, encryption is available as an extension to 1.2, and auditing can be added to any Java program by providing an auditing security manager. Still others of these items will be added in the future. But the basic premise remains that Java security was originally and fundamentally designed to protect the information on a computer from being accessed or modified while still allowing the Java program to run on that computer.

The point driving this notion of security is the new distribution model for Java programs. One of the driving forces behind Java, of course, is its ability to download programs over a network and run those programs on another machine within the context of a Java-enabled browser. Coupled with the widespread growth of Internet use--and the public-access nature of the Internet--Java's ability to bring programs to a user on an as-needed, just-in-time basis has been a strong reason for its rapid deployment and acceptance.

The nature of the Internet created a new and largely unprecedented requirement for programs to be free of viruses and Trojan horses. Computer users had always been used to purchasing shrink-wrapped software. Many soon began downloading software via ftp or other means and then running that software on their machines. But widespread downloading also led to a pervasive problem of malevolent attributes both in free and in commercial software. The introduction of Java into this equation had the potential to multiply this problem by orders of magnitude, as computer users now download programs automatically and frequently.

A. Sandbox

The sandbox security model makes it easier to work with software that comes from sources you don't fully trust. Instead of security being established by requiring you to prevent any code you don't trust from ever making its way onto your computer, the sandbox model lets you welcome code from any source. But as it's running, the sandbox restricts code from untrusted sources from taking any actions that could possibly harm your system. The advantage is you don't need to figure out what code you can and can't trust, and you don't need to scan for viruses. The sandbox itself prevents any viruses or other malicious code you may invite into your computer from doing any damage.

1) The sandbox is pervasive

If you have a properly skeptical mind, you'll need to be convinced that a sandbox has no leaks before you trust it to protect you. To make sure the sandbox has no leaks, Java's security model involves every aspect of its architecture. If there were areas in Java's architecture in which security was weak, a malicious programmer (a "cracker") potentially could exploit those areas to "go around" the sandbox. To understand the sandbox, therefore, you must look at several different parts of Java's architecture and understand how they work together.

The fundamental components responsible for Java's sandbox are:

Safety features built into the Java virtual machine (and the language)

The class loader architecture

The class file verifier

The security manager and the Java API

RECENT JAVA HOW-TOs

Choice Format: Numeric range formatting

Google Go

Fast guide to Google Go programming

Auto boxing, Unboxing, and No Such Method Error

2) The sandbox is customizable

One of the greatest strengths of Java's security model is that two of the four components shown in the above list, the class loader and the security manager, are customizable. To customize a sandbox, you write a class that descends from java.lang.SecurityManager. In this class, you override methods declared in the superclass that decide whether or not to allow particular actions, such as writing to the local disk. You will want to establish a custom Security Manager when you are using custom class loaders to load class that you don't fully trust.

As a developer, you may never need to create your own customized sandbox -- you can often make use of sandboxes created by others. When you write and run a Java applet, for instance, you make use of a sandbox created by the developers of the Web browser that hosts your applet.

III. THE JVM

The binary form of programs running on the Java platform is not native machine code but an intermediate bytecode. The JVM performs verification on this bytecode before running it to prevent the program from performing unsafe operations such as branching to incorrect locations, which may contain data rather than instructions. It also allows the JVM to enforce runtime constraints such as array bounds checking. This means that Java programs are significantly less likely to suffer from memory safety flaws such as buffer overflow than programs written in languages such as C which do not provide such memory safety guarantees.

The platform does not allow programs to perform certain potentially unsafe operations such as pointer arithmetic or unchecked type casts. It also does not allow manual control over memory allocation and deallocation; users are required to rely on the automatic garbage collection provided by the platform. This also contributes to type safety and memory safety.

IV. SAFETY FEATURES AND SECURITY

Because of the safety features built into the Java virtual machine, running programs can access memory only in safe, structured ways. This helps make Java programs robust, but also makes their execution more secure. Why? There are two reasons.

First, a program that corrupts memory, crashes, and possibly causes other programs to crash represents one kind of security breach. If you are running a mission-critical server process, it is critical that the process doesn't crash. This level of robustness is also important in embedded systems such as a cell phone, which people don't usually expect to have to reboot.

The second reason unrestrained memory access would be a security risk is because a wiley cracker potentially could use the memory to subvert the security system. If, for example, a cracker could learn where in memory a class loader is stored, it could assign a pointer to that memory and manipulate the class loader's data. By enforcing structured access to memory, the Java virtual machine yields programs that are robust -- but also frustrates crackers who dream of harnessing the internal memory of the Java virtual machine for their own devious plots.

A. Safety is built in

The prohibition on unstructured memory access is not something the Java virtual machine must actively enforce on a running program; rather, it is intrinsic to the bytecode instruction set itself. Just as there is no way to express an unstructured memory access in the Java programming language, also there is no way to express it inbytecodes -- even if you write the bytecodes by hand. Thus, the prohibition on unstructured memory access is a solid barrier against the malicious manipulation of memory.

There is, however, a way to penetrate the security barriers erected by the Java virtual machine. Although the bytecode instruction set doesn't give you an unsafe, unstructured way to access memory, through native methods you can go around bytecodes.

SUMMARY

Security is a multifaceted feature of the Java platform. There are a number of facilities within Java that allow you to write a Java application that implements a particular security policy, Java-enabled browsers (including those like HotJava that are written in Java) are the ultimate proof of these features: these browsers have used the features of the Java platform to allow users to download and run code on their local systems without fear of viruses or other corruption.

But the security features of Java need not be limited to the protections afforded to Java applets running in a browser: they can be applied as necessary to your own Java applications. This is done most easily by incorporating those features into a framework designed to run Java applications within a specified sandbox.

In addition, the security package allows us to create applications that use generic security features--such as digital signatures--for many purposes aside from expanding the Java sandbox.

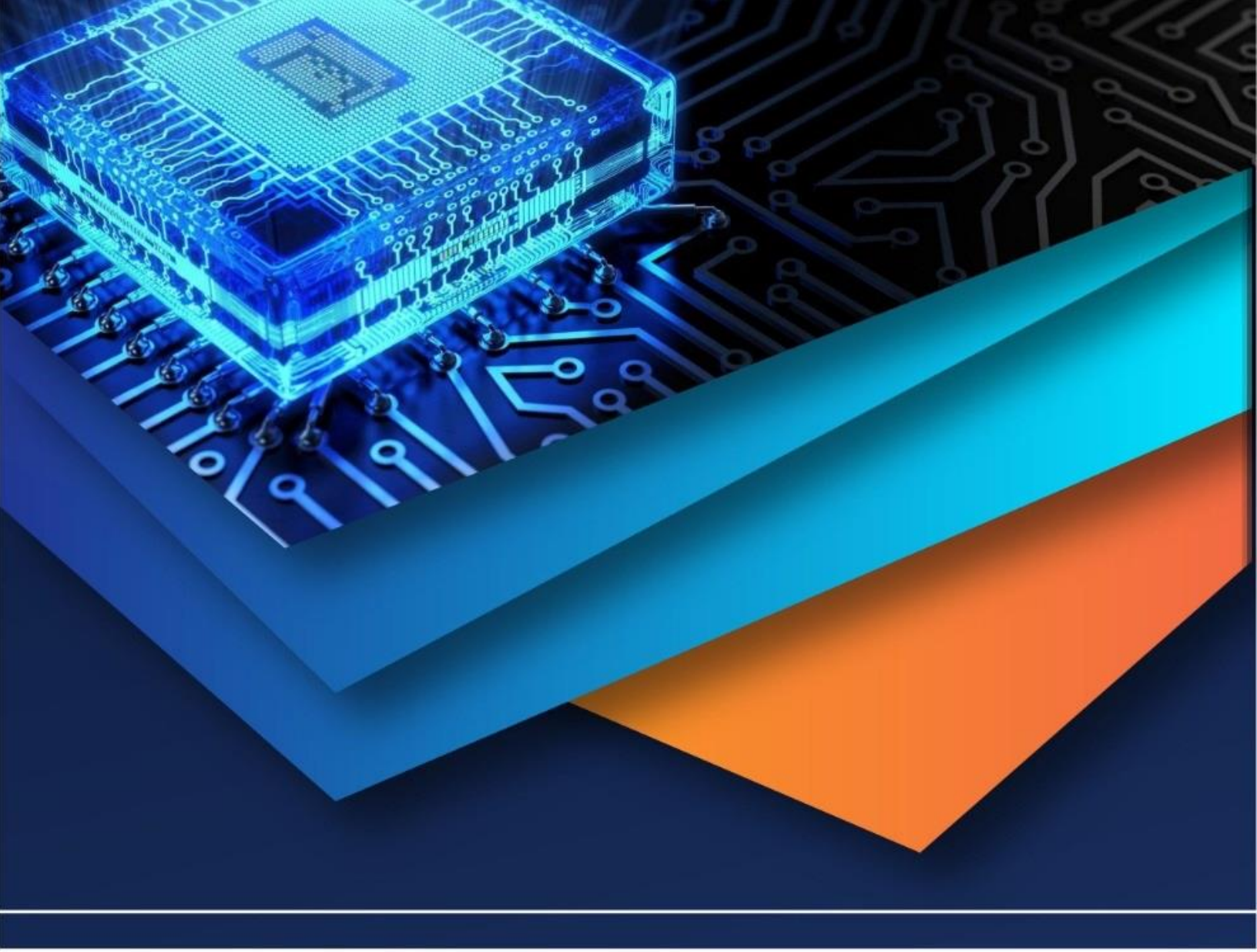
REFERENCES

[1] Secure Computing with Java Now and the Future http://www.javasoft.com/marketing/collateral/security.html

[2] The Java Security Home Page http://www.javasoft.com/security/

[3] The book *The Java virtual machine Specification*(http://www.aw.com/cp/lindholm-yellin.html), by Tim Lindholm and Frank Yellin .

[4] Li Gong. Java security: present and near future, IEEE Micro, 17(3):14-19, May/June 1997.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY