A Synthesize Approach: Modeling for Java Security

Ms.Shilpi Singh
School of Computer Science
Lingaya's University
Faridabad, India
4. shilpi@gmail.com

Sahil
B.Tech in Computer Science (4th year)
School of CSE
Lingaya's University
Faridabad, India
sahilmd17@gmail.com

AkshayDalal
B.Tech in Computer Science (4th year)
School of CSE
Lingaya's University
Faridabad, India
akshay.charra@gmail.com

Rohan Tanwar
B.Tech in Computer Science (4th year)
School of CSE
Lingaya's University
Faridabad, India
rohantanwar94@gmail.com

Abstract—Java is a programming language which is developed by Sun Microsystems. Sun Microsystems claim that Java has a number of advantages over traditional programming languages. One of the benefit is the ability to execute un trusted programs in a secure environment. This paper investigates the problems that would arise when running un trusted programs. It then takes an in depth explore the answer provided by the Java security model in theory furthermore as in current implementations and calculates their potency and adaptability for present and future ranges of application.

Keywords—security,java,sandbox,JVM,bytecode,applet,type-safe,garbage collector,code,verifier

1.INTRODUCTION

Security is the way by which organizations and individuals protect their physical and intellectual property from all types of steal and attack. Even though security concerns are not new, there is re-energized awareness in the entire area of security in computing systems. The reason behind this is today's info. Systems have become the storehouses for both personal and corporate properties and networks are providing new levels of admittance for users. Therefore, new opportunities for unauthorized interaction and possible misuse may arise. In order to fight possible security threats, users need programs they can trust upon. Also, developers are looking for a development platform that has been deliberate with built-in security capabilities. That's why the Java comes in. Actually, Java is designed for network-based computing, and security is a vital part of Java's design[8][9].

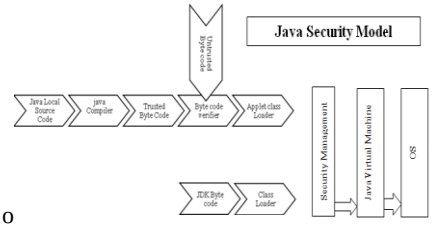
Java security model is appropriately outlined by mistreatment the figure of the Sandbox [1][2]. The sandbox contains variety of collaborating system elements, move from security managers that execute as some of the applying, to security measures designed into the Java Virtual Machine (JVM) and therefore the language itself[10]. Dr. Li Gong has categorised the Java security model into four layers, which are [1]:

1. Java language is designed to be type-safe, and easy to use. Java features such as garbage collection automatic memory management, and range checking of strings and arrays are examples of how the language helps the developer to writes after code.
2. Compilers and a bytecode verifier check only genuine Java code is executed. The bytecode verifier, together with the Java virtual machine, assurances language type safety at runtime.
3. A class loader describes a local namespace, which is used to confirm that an untrusted applet cannot interfere with the running of other Java programs.
4. Access to vital system resources is intermediated by the Java virtual machine and is checked inadvance by a Security Manager class that restricts to the minimum the actions of un trusted code.

Key Features that Make Java More Secure than Other Languages[6]-

- **Java's security model [7]**
  Java's security model is meant to assist and shield users from hostile programs downloaded from some un trusted resource at intervals a network through "sandbox". It allows all the code to run within the sandbox exclusively and avoids numerous activities from un trusted resources together with reading or writing to the native disk, making any new method or maybe loading any new dynamic library whereas occupation a native methodology.
- **No use of pointers**
  C/C++ language uses pointers, which can cause unauthorized access to memory blocks once alternative programs get the pointer values. in contrast to standard C/C++ language, Java never uses any kind of pointers. Java has its internal mechanism for memory management. It solely provides access to the data to the program if has suitably verified authorization.
- **Exception handling concept**
  The concept of exception handling allows Java to capture a series of errors that helps developers to get eliminate the

chance of crashing the system.



o

- **Defined order execution**

All the primitives are defined with a predefined size and every one the operations are defined in a very specific order of execution.

Therefore, the code executed in numerous Java Virtual Machines won't have a distinct order of execution.

- **Bytecode is another factor that creates Java safer**
  Every time a user compiles the Java code, the Java compiler creates a class file with Bytecode, that is tested by the JVM at the time of program execution for malicious files.

- **Tested code re-usability**
  Object encapsulation delivers support for the concept of "programming by contract". this enables the developers to re-use the code that has already been tested whereas developing Java enterprise applications.

- **Access control functionality**
  Java's access-control functionality on variables and methods inside the objects give a secure program by preventing access to the important objects from the untrusted code.

- **Protection from security attacks**
  It permits developers to declare classes or methods as FINAL. we have a tendency to all recognize that any class or method declared as final can't be overridden, that helps developers to guard code from security attacks like making a subclass and replacing it with the original class and override methods.
- **Garbage collection mechanism**
  It aids a lot of to the safety measures of Java. Garbage collector offers a clear storage allocation and recovering unutilized memory instead of deallocating memory through manual action. It facilitate developers to make sure the integrity of the program throughout its execution and avoids any JVM crash attributable to incorrect releasing of memory.
- **Type-safe reference casting in JVM**
  Whenever you utilize an object reference, the JVM monitors you. If you are attempting to forged a reference to a distinct type, it'll build the forged invalid.

2. JAVASECURITYARCHITECTURE

Java security model can be explained in Fig. 1 [5]. Both Java byte code and applets (observed as un trusted byte code) must pass the byte code verifier. Then the class loader is invoked to regulate how and when applets can load classes. A class loader also imposes namespace partition, and it ensures that one applet cannot affect the rest of the runtime environment. Finally,the security manager is employed to perform run-time verification of all questionable "dangerous methods", which are those methods that request file I/O, network access, or those that want to define a new class loader.

In the remainder of this section, we'll shortly describe 3 components of the Java security model, that square measure byte code verifier, class loader, and security manager.

2.1. Java BytecodeVerifier

Java compiler compiles source programs into bytecodes, and a truthful compiler guarantees that Java source code doesn't interfere the security rules. At runtime, a compiled code portion will return from anyplace on net, and it's anonymous if the code fragment comes from a trustworthy compiler or not. So, much the Java runtime merely doesn't trust the incoming code and instead subjects it to a sequence of tests by bytecode protagonist.

The bytecode verifier may be a mini theorem prover, that verifies that the language ground rules are valued. It checks the code to verify that [5]:

- Compiled code is formatted correctly.
- Internal stacks will not over flow or underflow.
- No "illegal" data conversions will happen (i.e., the verifier will not permit integers to serve as pointers). This safeguards that variables will not be granted entree to restricted memory areas.
- Byte-code instructions will have appropriately-typed parameters.
- All class member accesses are "legal". For instance, an object's private data must always remain private.

The bytecode verifier assurances that the code passed to the Java interpreter is in a fit state to be executed and can run without f-ear of breaking the Java interpreter.

2.2. Java Class Loader

The class loader is outlined in Java by an abstract class, ClassLoader. As an interface, it will be used to describe a rule for loading Java classes into the runtime environment. functions of the class loader are [5]:

- It fetches the applet's code from the remote.
- Class Loader creates and imposes a namespace hierarchy. one amongst its additional necessary functions is to make sure that running application program doesn't replace system-level components inside the runtime environment. particularly, it prevents applets from making their own category loader.
- It stops applets from invoking the method, that is an element of the system's category loader.

Java Runtime environment (i.e., a running JVM), permits multiple ClassLoaders, every with its own namespace, to move at the same time, and namespaces enable the JVM to cluster classes supported wherever they originate (e.g., native or remote). This delineates and controls what alternative parts of the runtime setting the application program will access and modify. additionally, by putting boundaries on the namespace, the category loader prevents untrusted applets from accessing alternative machine resources (e.g., native files).

2.3. Java Security Manager

The Security Manager contains variety of ways that are planned to be referred to as to visualize specific kinds of actions. the security Manager class itself isn't supposed to be used directly, instead, it's supposed to be subclassed and put in because the System Security Manager. The subclasses Security Manager are often wont to instantiate the required security policy.

The Security Manager delivers a awfully versatile and powerful mechanism for not absolutely permitting access to resources. the security Manager methods that check access are passed arguments that are necessary to implement conditional access policies, furthermore as having the potential to visualize the execution stack to see if the code has been referred to as by native or downloaded code. a number of the security Manager's responsibilities embody [5]:

- Handling all socket operations.
- Safeguarding access to protected resources together with files, personal data, etc.
- Controlling the creation of, and every one access to, OS programs and processes.
- Preventing the installation of new ClassLoaders.
- Maintaining thread integrity.
- Controlling access to Java packages (i.e., groups of classes).

To ensure compliance, all methods that are a part of the essential Java libraries (i.e., those equipped by Sun) consult the security Manager before executing any dangerous operations, like network access and file I/O request.

Our objective of trusting execution of untrusted comes on a JVM obliges answers for numerous problems, for instance, characterizing the conduct of JVM execution, characterizing safe execution on the JVM, and demonstrating that perceived comes execute firmly on JVM.

3.CONCLUSION

Java security model provides us an excellent test bed for security verification. Presently, researchers are thinking about describing the formal semantics of Java byte code instructions and trying to prove their reliability. Verifying byte code by model checking is one of those works and it's different from the traditional theorem proving approach. Because there are many existing models checking tools, such as SMV, it gives us a chance to concentrate on creating the model for byte code, and let the model checker do the rest things,likesecurity verification.

4. ACKNOWLEDGEMENTS

We would like to thanks to our Vice chancellor Prof. Dr. R.K.Chauhan, Pro Vice chancellor Prof . Dr. G.V.Ramaraju Lingaya's university Faridabad, and faculty members of school of computer science for valuable guidance and support for writing this paper.

5.REFERENCES

[1] LiGong.Javasecurity:presentandnearuture,*IEEEMicro*,17(3):14-19,May/June1997.

[2] Li Gong. Going beyond Sandbox: an overview of the new security architecture in the Java Development kit 1.2, In *Proceedings of the USENIX Symposium on Internet Technologies and Systems* , Monterey, California, December1997

[3] Joseph A.Bank.Java security,PMG group at MIT LCS,December,1995,http://swissnet.ai.mit.edu/~jbank/javapaper/javapaper.html

[4] Rich Levin. Security grows up: the Java 2 platform security model,October1998,http://www.javasoft.com/features/1998/11/jdk.security.html

[5] Executive Summary Secure computing with Java:now and the future,1998,http://www.javasoft.com/marketing/collateral/security.htm

[6] http://www.cygnet-infotech.com/blog/key-features-that-make-java-more-secure-than-other-languages

[7]https://www.researchgate.net/publication/2644514_Java_Security Model_and_Java_Security_Model_and_Bytecode_Verification_ Bytecode_Verification

[8] Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, Alex (2014). The Java® Language Specification (PDF) (Java SE 8 ed.).

[9]Gosling, James; Joy, Bill; Steele, Guy L., Jr.; Bracha, Gilad (2005). The Java Language Specification (3rd ed.). Addison-Wesley. ISBN 0-321-24678-0.

[10] Lindholm, Tim; Yellin, Frank (1999). The Java Virtual Machine Specification (2nd ed.). Addison-Wesley. ISBN 0-201-43294-3.