iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2 **Issue:** IX **Month of publication:** September 2014

DOI:

www.ijraset.com

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

ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Java Applet Security

Chirag Gulati[1], Ayushi Mishra[2], Chetna Mahajan[3]

Dronacharya College Of Engineering,
Gurgaon,HR

Abstract: ***Java's early growth was spurred by code downloadable over a network, better known as APPLETS. Applet security has evolved with the growth of Java, and today is a source of frequent confusion due to the variety of Java versions, commercially available browsers, and plug-ins. Applets are automatically (an applet may not give any visual signal of its existence or execution) downloaded across the network and run on the host machine when a user accesses in a browser a web page containing applet. A user coming across malicious applets is within the realms of possibility.***

The Java Sandbox model provides you an environment, where you could welcome any code from any source. But when the code from an untrusted source runs Sandbox restricts the code from taking any actions that could possibly harm your system. It terms of applets the restrictions means that applets will be unable to determine information about each other; each applet is given its own memory space to run. Java byte code modification technique is used to insert extra run-time test into java applets. This technique is used to restrict applet behavior or, potentially, insert code appropriate to profiling and other monitoring efforts.

Keywords: Java applet security, browse-dependent, access control, security manager, digital signature, byte-code modification.

I. INTRODUCTION

1.) JAVA APPLETS

Java applets are applications written using the Java programming language that are embedded in web pages. Applets typically provide dynamic functionality that is not supported by plain HTML or a combination of HTML and JavaScript.

Perhaps ironically, the functionality of JavaScript is sometimes invoked from Java applets to achieve things that *applets* cannot do on their own. Further, the deployJava.js JavaScript provided by Oracle is designed to launch applets after checking a suitable minimum Java version is installed. While Java can do things that JavaScript cannot - most modern applets would not get very far if JavaScript was disabled in the user's browser!

While applets might seem easy to develop, they are actually quite tricky. To deploy an applet reliably to 'all comers' (or at least the vast majority) on the WWW is an order of magnitude more difficult again.

The Java applet API provides methods considered handy to web based applications. These include methods to gain images and audio clips, discover and communicate with other applets, ascertain the code base and document base to allow relative references to resources (images, clips, text files, etc.) that the applet might use.Applets could be embedded in web pages since Java 1.1.

Applet 'Hello World' Example

```
/* <!-- Defines the applet element used by the appletviewer. -->
<applet code='HelloWorld' width='200' height='100'></applet>
*/
import javax.swing.*;

/** An 'Hello World' Swing based applet.

To compile and launch:
prompt> javac HelloWorld.java
prompt> appletviewer HelloWorld.java  */
```

ISSN: 2321-9653


```
publicclassHelloWorldextendsJApplet {

publicvoidinit() {
// Swing operations need to be performed on the EDT.
// The Runnable/invokeAndWait(..) ensures that happens.
Runnable r = newRunnable() {
publicvoid run() {
// the crux of this simple applet
getContentPane().add( newJLabel("Hello World!") );
          }
       };
SwingUtilities.invokeAndWait(r);
   }
}
```

\

2.)Javaapplet security with the help of bytecode modification

JVM include byte code verifier that check the bytecode program before program execution,and a byte code interepter that perform run time task such as array bounds and null-pointers check,java applets may still behave in different way that may be annoying and potentially harmful for user.JVM verifies code property and perform additional operation at run-time. Java byte code modification is a technique by which we can put some restriction on the applet by inserting extra bytecode operation that may work at the run time. These additional bytecode instruction may monitor and control resource usage,limit applet functionality and provide control over inaccessible object.

Java applets downloaded from the Internet or from any remote sources are restricted from reading and writing files and making network connections on client host systems. They are also restricted from starting other programs, loading libraries, or making native calls on the client host system. In general, applets downloaded from a network or remote sources are considered untrusted. An applet can be considered trusted, based on the following factors:

- Applets installed on a local file system or executed on a localhost.
- Signed applets provide a way to verify that the applet is downloaded from a reliable source and can be trusted to run with the permissions granted in the policy file.

Applets implement additional security restrictions that protect users from malicious code too. This is accomplished through the java.lang.SecurityManager class. This class is subclassed to provide different security environments in different virtual machines.

To test the applet using an APPLETVIEWER, you may choose to use the

-J-DJAVA.SECURITY.POLICY=WRITEAPPLETPOLICY.POLICY option on the JVM command line, or you can explicitly specify your policy file in the JVM security properties file in the <JAVA_HOME>/jre/lib/security directory

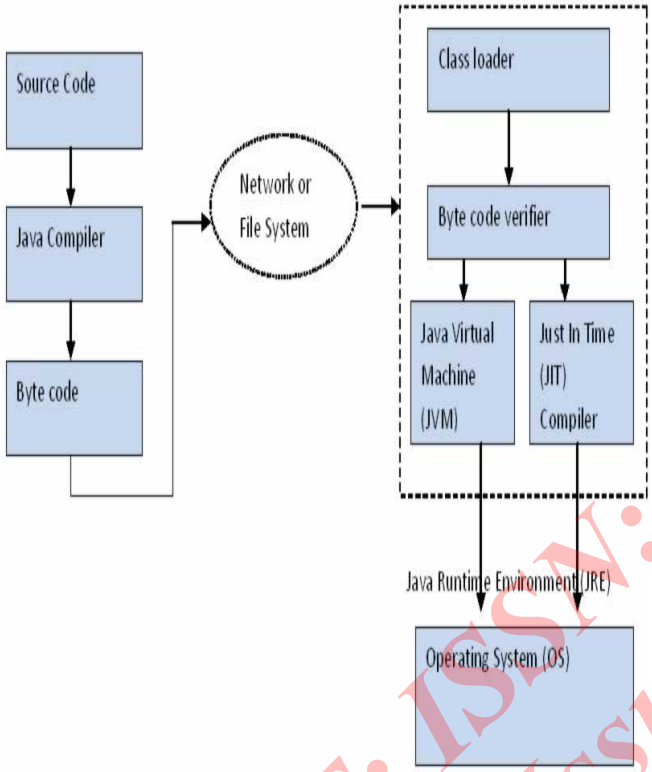
```
policy.url.3=file:/export/xyz/WriteAppletpolicy.policy
```

Java byte modification can be done by two general forms class level and method level modification.

2.1)Class level modification

In this method class such as Window can be replaced by subclasses window (which will called Safe$window) that restrict resource usage and functionality. Safe$window constructor method can put a limit on how many window's are open at a time. The

method allows the window to be created until a number of windows exceed the limit. Since Safe$window is the sub-type of window, and can appear where ever the window is expected. Hence, the applet should not notice the change, unless it attempts to create window



exceeding limit.

Class-level substitution requires a simple modification of constant pool entity,since it takes advantage of the property of class inheritance

2.2) Method level Modification

Method level modification provides more flexibility it can be used even when the method is final or is accessed through an interface,but requires more complicated modifications of method reference and invoking instructions.

II. HOW SANDBOX MODEL PROVIDE SECURITY IN JAVA APPLETS

In computer security, a sandbox is a security mechanism for separating running programs. It is often used to execute untested code, or untrusted programs from unverified third parties, suppliers, untrusted users and untrusted websites. Sandbox model is a directed avalanche system that exhibits self-organized criticality.The Java sandbox is responsible for protecting a number of resources, and it does so at a number of levels. Consider the resources of a typical machine as shown in Figure 1. The user's machine
has access to many things:

- Internally, it has access to its local memory (the computer's RAM).
- Externally, it has access to its filesystem and to other machines on the local network.
- For running applets, it also has access to a web server, which may be on its local (private) net or may be on the Internet.
- Data flows through this entire model, from the user's machine through the network and (possibly) to disk.

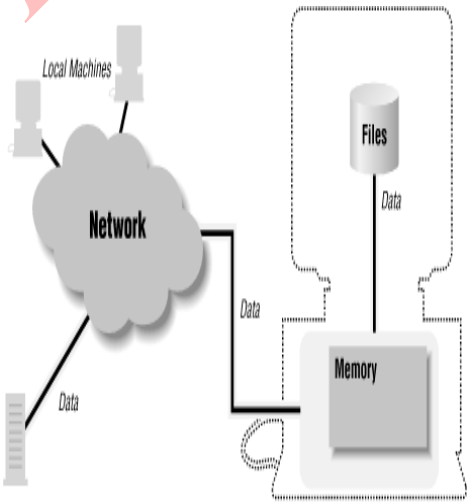


Figure 1 (A machine has access to many resources)

ISSN: 2321-9653

Each of these resources needs to be protected, and those protections form the basis of Java's security model. We can imagine a number of different-sized sandboxes in which a Java program might run:

- A sandbox in which the program has access to the CPU, the screen, keyboard, and mouse, and to its own memory. This is the minimal sandbox -- it contains just enough resources for a program to run.
- A sandbox in which the program has access to the CPU and its own memory as well as access to the web server from which it was loaded. This is often thought of as the default state for the sandbox.
- A sandbox in which the program has access to the CPU, its memory, its web server, and to a set of program-specific resources (local files, local machines, etc.). A word-processing program, for example, might have access to the docs directory on the local filesystem, but not to any other files.
- An open sandbox, in which the program has access to whatever resources the host machine normally has access to.

The sandbox, then, is not a one-size-fits-all model. Expanding the boundaries of the sandbox is always based on the notion of trust. And so it is with Java programs: in some cases, I might trust them to access my filesystem; in other cases, I might trust them to access only part of my filesystem; and in still other cases, I might not trust them to access my filesystem at all.A security measure in the Java development environment. The sandbox is a set of rules that are used when creating an applet that prevents certain functions when the applet is sent as part of a Web page. When a browser requests a Web page with applets, the applets are sent automatically and can be executed as soon as the page arrives in the browser. If the applet is allowed unlimited access to memory and operating system resources, it can do harm in the hands of someone with malicious intent. The sandbox creates an environment in which there are strict limitations on what system resources the applet can request or access. Sandboxes are used when executable code comes from unknown or untrusted sources and allow the user to run untrusted code safely.

The anatomy of a typical Java program is shown in Figure 2. Each of the features of the Java platform that appears in a rectangle plays a role in the development of the Java security model.The Java sandbox relies on a three-tiered defense. If any one of these three elements fails, the security model is completely compromised and vulnerable to attack:

byte code verifier -- This is one way that Java automatically checks untrusted outside code before it is allowed to run. When a Java source program is compiled, it compiles down to platform-independent Java byte code, which is verified before it can run. This helps to establish a base set of security guarantees.

applet class loader -- All Java objects belong to classes, and the applet class loader determines when and how an applet can add classes to a running Java environment. The applet class loader ensures that important elements of the Java run-time environment are not replaced by code that an applet tries to install.

security manager -- The security manager is consulted by code in the Java library whenever a dangerous operation is about to be carried out. The security manager has the option to veto the operation by generating a security eThe security package

The security package (that is, classes in the java.security package as well as those in the security extensions) allows you to add security features to your own application as well as providing the basis by which Java classes may be signed. Although it is only a small box in this diagram, the security package is a complex API and discussion of it is broken into several chapters of this book. This includes discussions of:

- The security provider interface -- the means by which different security implementations may be plugged into the security package
- Message digests
- Keys and certificates
- Digital signatures
- Encryption (through JCE and JSSE)
- Authentication (through JAAS)

The key database

The key database is a set of keys used by the security infrastructure to create or verify digital signatures. In

the Java architecture, it is part of the security package, though it may be manifested as an external file or database.

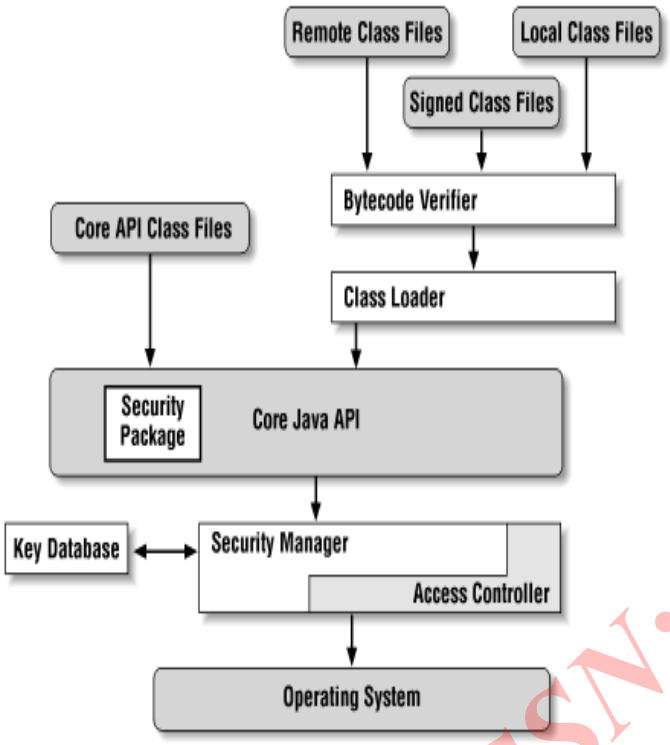


Figure 2. Anatomy of a Java application

With respect to the sandbox, digital signatures play an important role because they provide authentication of who actually provided the Java class.

Inside and Outside Sandbox---Stack Based Access Control

In general, the applet should not damage hardware, software, or information on the host machine, should not pass unauthorized information to anyone, shouldn't cause the host machine to become unusable through resource depletion.

The simplest way to do this is, applets loaded over the net are prevented from reading and writing files on the client file system, and from making network connections except the originating host.

In addition, applets loaded over the net are prevented from starting other programs on the client. Applets load over the net are also not allowed to load libraries, or to define native method calls. If an applet could define native methods calls, that would give the applet direct access to the underlying computer.

There are other specific capabilities denied to applets loaded over the net, but most of the applet security policy called sandbox is described by those two paragraphs above.

The sandbox model is easy to understand and to be implemented, but it prevents many kinds of useful programs from being written. The problem here is that this policy don't address any difference of which applet come from. Some applets may come from that you trust, others may come from that you didn't know. To tell the difference of applet does make sense to security concerns. Now suppose that the browser know exactly where the applet come from, and know that the applet isn't affected in transmission, should it allow the applet to go beyond the sandbox? Definitely. The stack based access-control decisions boil down to who is allowed to do what in where.

In this kinds of model, a principal represents the "who", sometime it is grouped several zones; such as Local Machine, Intranet, Trusted Web sites, the Internet and Untrusted sites(Microsoft Explorer 4).

The target represents the "where", such as a local file or directory.

The privilege is the "what", such as User Directed File I/O, Network Connection, Executing other application on the clients, Exit VM, Read system properties.

The stack based Access Control works like this. Four fundamental primitives are necessary to use stack inspection: enablePrivilege(), disablePrivilege(), checkPrivilege(), revertPrivilege(). When a target T (such as the file system) needs to be protected, the system must be sure to call checkPrivilege(T) before accessing T. The algorithm searches the frames on the caller's stack in sequence, from newest to oldest. The search terminates, allowing access, upon finding a stack frame that has an enabled-Privilege(T) annotation. The search also terminates, forbidding access, upon finding a stack frame that is either forbidden by the local policy from accessing the target or that has explicitly disabled its privileges.

When an applet wishes to use T, it must first call enablePrivilege(T). This consults the local policy to see whether the principal of the caller is permitted to use T. Sometimes, the decision is based on the setup, sometimes, it will need human-machine interactive procedure. If it is permitted, an enabled-Privilege(T) annotation is made on the current stack frame. The applet may then use T normally. Afterward, the code may call

revertPrivilege(T) or disablePrivilege(T) to discard the annotation.
This stack based access control has been proved to be a useful tool to expressing and managing complex trust relationships. The stack inspection is formalized using a logic developed by Abadi, Burrows, Lampson, and Plotkin. Java's access control decisions correspond to proving statements in ABLP logic.
secure system could require a digitally signed request before it sent out the payroll information.

Two forms of Java Applets

- Abstract Window Toolkit (AWT): AWT classes and components are contained in the java.awt package hierarchy. It is a set of APIs that provides GUI for java program like buttons, checkbox etc. AWTs are considered obsolete. Swings are preferred over AWT components.

- Swings: The Swing classes and components are contained in the javax.swing package hierarchy. Swing components are known as lightweight because they do not require allocation of native resources in the operating system's windowing toolkit, whereas AWT components are referred to as heavyweight components.

Swings are considered better than AWT:

1. Swing components are not implemented by platform-specific code, they are written entirely in
2. Java and therefore are platform-independent unlike AWT components.
3. It has more flexible components than AWT.
4. In J2SE 1.2, Swings superseded AWT's widgets as it provides a richer set of UI widgets.
5. Swing draws its own widgets
6. Not yet obsolete, the same used by me in previous article.

The advent of executable contents such as Java applets exposes World Wide Web (WWW) users to a new class of attacks that were not possible before. Despite an array of security checking, detection, and enforcement mechanisms built into the language model, the compiler, and the run-time system of Java, serious security breach incidents due to implementation bugs still arose repeatedly in the past several years. Without a provably correct implementation of Java's securityarchitecture specification, it is difficult to make any conclusive statements about the security characteristic of current Java virtual machines. The Spout project takes an alternative approach to address Java's security problems. Rather than attempt a provable secure implementation, we aim to confine the damages of malicious Java applets to selective machines, thus, preventing the machines behind an organization's firewall from being attacked by malicious or buggy applets. More concretely, Spout is a distributed Java execution engine that transparently decouples the processing of an incoming applet's application logic from that of graphical user interface (GUI), such that the only part of an applet that is actually running on the requesting user's host is the harmless GUI code. A unique feature of the Spout architecture that does not exist in other similar systems, is that it is completely transparent to and does not require any modifications to WWW browsers or class libraries on the end hosts.

III. JAVA APPLET SECURITY STANDARD---ACCESS CONTROL AND SOFTWARE PUBLISH LEVEL

Several concerns in constructing this kind of standard.

The principal hierarchy standard: How many kinds of place Java applet may come from? The Microsoft Explorer groups the Java applet sources as local machine, intranet, trusted sites, internet, untrusted site. Every group has been assigned some privileges to some targets. So if an appletes originated from a grouped

site, it won't ask the user in runtime to grant the privilege. It's better in performance and the ergonomics than those ask user frequently.

The privilege hierarchy standard: How many kinds of operation a principal may apply to the target? To a file system, the basic operation may be read, write. How many operation could be applied to the thread? Network connection?

The target classification: How many kind of resource and information can be classified in user local machine? Is the classification like CPU computation capability, hard disk space, file system, display system? How to handle the granulation?

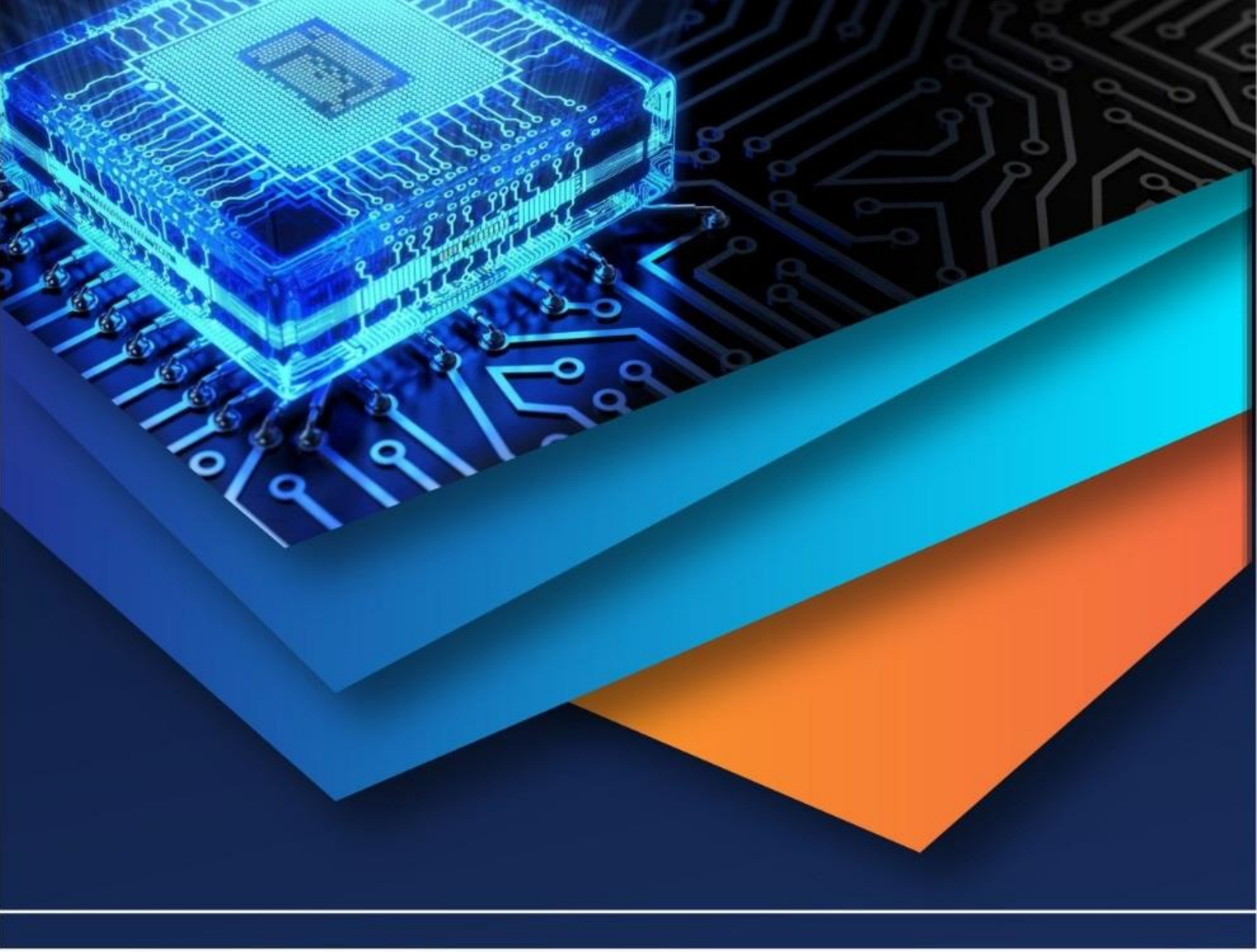
The human-machine interactive interface standard: What kinds of information that an applet must get in runtime? What content and frequency should be prompted to user when the local resource be accessed?

The standard software publish file format standard: Which compressed standard should be used, what content should be put in the file?

After the above standard works, the Java applet should be browser-independent. However, it won't be easy not only for pure technique reason but also for commercial benefit diversity.

REFERENCES

1. Laura lemay, Charles L. Perkins, and Michael Marrison, Teach yourself JAVA in 21 days, Sams.net publishing, 1996.
2. Frank Yellin, Low Level Security in Java, http://java.sun.com:80/sfaq/verifier.html.
3. Dan S. Wallach and Edward W. Felten, Understanding Java Stack Inspection, 1998 IEEE Symposium on Security and Privacy.
4. Dan S. Wallach, Dirk Balfanz, Drew Dean and Edward W. Felten, Extensible Security Architectures for Java, the 16th Symposium on Operating Systems Principles.
5. Drew Dean, Edward W. Felten and Dan S. Wallach, Java Security: From HotJava to Netscape and Beyond, 1996 IEEE Symposium on Security and Privacy.
6. Netscape, Netscape object signing—Establishing trust for downloadedsoftware,http://developer.netscape.com/library/documentation/signedobj/trust/owp.html.
7. Microsoft, Trust-based Security for Java, http://www.microsoft.com/java/ security/ jsecwp.html.
8. VeriSign, Inc. VeriSign Digital ID for Netscape Object Signing, http://digitalid.verisign.com/nosintro.html.
9. http://en.wikipedia.org/wiki/Java_applet
10. http://docs.oracle.com/javase/tutorial/deployment/applet/security.html
11. http://www.securingjava.com/chapter-two/
12. http://www.javaworld.com/article/2076262/core-javaava-security-evolution-and-concepts--part/core-java/java-security-evolution-and-concepts--part-3--applet-security.html
13. http://books.google.co.in/books/about/Java_security.html?id=EhX9BjHj9M4C
14. http://www.oracle.com/technetwork/java/index.html
15. http://www.eclipse.org/
16. http://books.google.co.in/
17. W.R. Cheswick and S.M. Bellovin, Firewalls and Internet Security,Repelling the Wily Hacker. Addison-Wesley, 1994
18. Securing Java: Getting Down to Business with Mobile Code(Gary McGraw and Ed Felten John Wiley and Sons
19. Java Security: Hostile Applets, Holes, and Antidotes(Gary McGraw ,Ed Felten and John Wiley and Sons.
20. Web Security sourcebook: A Complete Guide to Web Security (Avirubin, Daniel Geer and marcusranum sons



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY