Security for Java Applets

Yashwanth Byalla[1], B Vishnu Vardhan Reddy[2], B Sudeep[3]
Department of CSE
MGIT
yashwanthbyalla95@gmail.com[1], vishnu.1384@gmail.com[2], balinenisudeep@gmail.com[3]

***Abstract*--**This paper provides access to information related to Java security. We have customized this information for different types of Java users. This paper also deals with how do I control when an untrusted applet or application runs in my web browser? Oracle is committed to understanding and responding to your Java security. This also deals with the browser that can interpret Java byte code (such as Netscape Navigator or Internet Explorer) can download and locally execute applets that are embedded in a Web page. It is extremely unlikely that all users of Java enabled browsers will consider the security implications of surfing a site before each Web page access. If the mobile code paradigm is going to work, security concerns should be addressed in the language of the content itself.

Keywords: *applet, classpath, web server, byte code;*

*****

I. Introduction

A Java applet is a small application written in Java and delivered to users in the form of bytecode. It is a little application that performs one specific task that runs within the scope of a program. It is a small program that can be sent along with a Web page to a user. An applet is a Java program that runs in a Web browser. The important features of the applets are:

- An applet is a Java class that extends the java.applet.applet class.
- Applets are designed to be embedded within an HTML page.
- A main() method is not invoked on an applet.
- Applet class will not define main().
- A JVM is required to view an applet.

Importing the different packages

In the Java language, every class is in a package. If the source code for a class doesn't have a package statement at the top, declaring the package the class is in, then the class is in the default package.

As you can see, importing the JApplet and Graphics classes lets the program refer to them later without any prefixes. The javax.swing and java.awt prefixes tell the compiler which packages it should search for the JApplet and Graphics classes. Both the javax.swing and java.awt packages are part of the core java API always included in the Java environment.

The javax.swing package contains classes for building Java graphical user interfaces (GUI's), including applets. The java.awt package contains the most frequently used in the Abstract Window Toolkit (AWT).

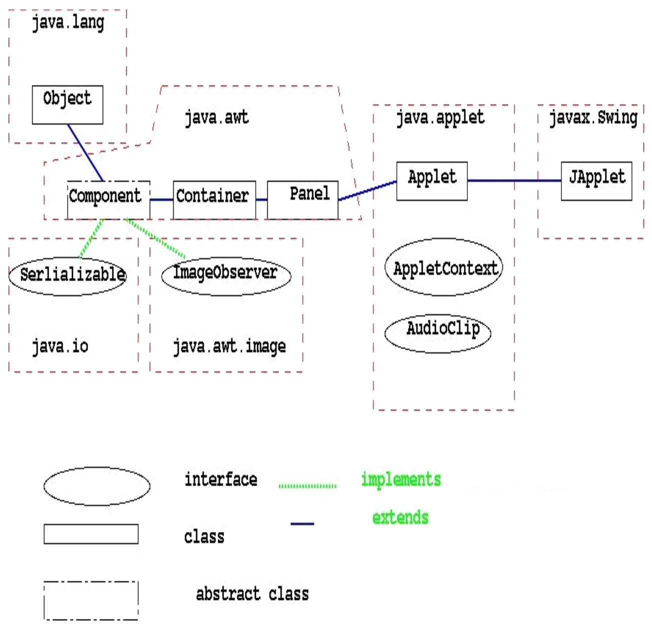


Fig 1. Life Cycle of an Applet

Applet execution

There is no main() method in an Applet. Applets do not begin execution at main(). Instead, an applet begins execution when the name ofits class is passed to an applet viewer or to a web browser. An applet program is written as a inheritance of the java.Applet class. An applet uses AWT for graphics. The applet is running and rendered on the web page. Every applet needs to implement one or more of the init(), the start() and the paint() methods. At the end of the execution, the stop() method is invoked, followed by the destroy() method to deallocate the applet's resources.

Two ways to run an applet:

(1) Executing the applet within a Java compatible web browser.

(2) Using appletviewer

- An appletviewer executes your applet in a window.
- This is generally the fastest and easiest way to test your applet.

The Applet class

Every applet is an extension of the java.applet.Applet.class. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context. Ordinarily, the methods init() and the paint() are overridden. Others can be left alone, in which case, nothing specific will be done when they are invoked by the browser. This includes methods that do the following:

- Get applet parameters.
- Get the network location of the HTML file that contains the applet.
- Get the network location of the applet class directory.
- Print a status message in the browser.
- Fetch an image, an audio clip.
- Play an audio clip.
- Resize the applet.

Invoking an Applet

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java enabled browser. The <applet> tag is the basis for embedding an applet in an HTML file. Below is an example that invokes the applet:

```
/*<applet code=""
width=
height=  >
</applet>*/
```

The code attribute of the <applet> tag is required. It specifies te Applet class to run. Width and Height are also required to specify the initial size of the panel in which an applet runs. The applet directive must be closed with a </applet> tag. Non Java enabled browsers do not process <applet> and </applet>. So, anything that appears between the tags, not related to the applet, is visible in non Java enabled browsers.

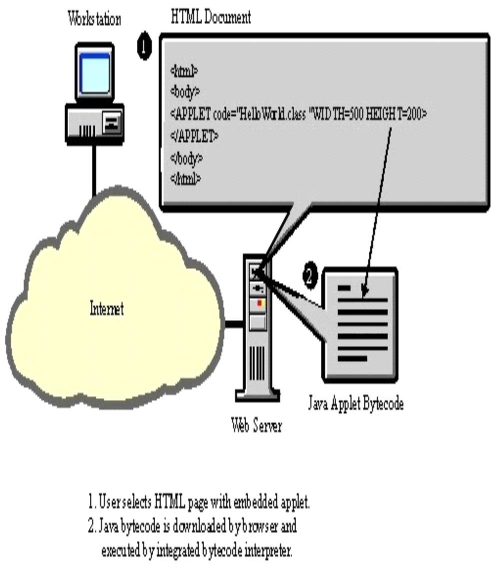

Workstation
HTML Document
1
<html>
<body>
<APPLET code="HelloWorld.class "WIDTH=500 HEIGHT=200>
</APPLET>
</body>
</html>
Internet
2
Web Server
Java Applet Bytecode
1. User selects HTML page with embedded applet.
2. Java bytecode is downloaded by browser and executed by integrated bytecode interpreter.

Fig 2: Interpreting the Byte Code which is downloaded from the browser

II. Applet Security

The two browsers security policies are, at the present time, very similar. Both are somewhat strict. The following rules apply to all untrusted applets running under Netscape Navigator and Internet Explorer:

- Applets cannot read or write files locally.
- If an applet is loaded using the file: URL and it does not reside in a directory in CLASSPATH, it is loaded by an Applet Class Loader.

If the mobile code paradigm is going to work, security concerns should be addressed in the language of the content itself. That way, users will not need to worry too much about security. Java was developed with key security issues in mind. It is clear that the Java development environment attempts to address the security problems introduced by the idea of dynamically downloading and running external, untrusted code. Security concerns have always been one of the major technical stumbling blocks to achieving safe mobile code. Java took these concerns seriously and made a good effort to protect Web users.

An alternative approach to handling mobile code is to run only code that is trusted. There are many ways to impose a sandbox model on executable content Java presents just one. Since the Java sandbox model has been widely distributed to millions of users with their Web browsers, it is safe to say that the java sandbox is today's most widely used sandbox model. Recently, it was correct to assume that a Java sandbox placed particular constraints on Java applets. JDK 1.1 changed all that and the notion of a sandbox is becoming ever more complex. With the introduction of JDK 1.1, Java's sandbox model underwent a state transition from a required model applied equally to all Java applets to a malleable system that could be expanded and personalized on an applet-by-applet basis. The addition of code signing to Java complicates things immensly.

What Untrusted Java Code can do?

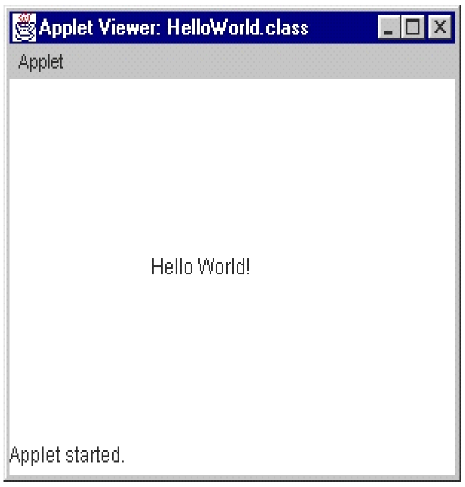
An alternative approach to presenting the sandbox is to define what untrusted applets can do. The default sandbox for untrusted applets also includes access to the Web server from which the applet was downloaded. Programmers not used to such constraints often complain that the default sandbox is too restrictive, for example, the inability to read and write temporary files must be designed around. Most security flaws exist because of programming bugs, anything that reduces the general level of bugginess in software is good for security. Java helps conscientious programmers reduce the odds that their code contains security bugs.

Applet for simple applet: HelloWorld

We can use applets to generate simple example of printing a message "HelloWorld". The following is a simple applet of HelloWorld:

```
import java.applet.*;
import java.awt.*;
/*<applet code="HelloWorld" width=400
height=400></applet>*/
public class HelloWorld extends Applet
{
    public void paint( Graphics g)
    {
        g.drawString("HelloWorld",10,100);
    }
}
```

Without these import statements, the Java compiler would not recognize the classes Applet and Graphics. The output of the above program is:



```
Hello World!
```

```
Applet started.
```

III. Advantages of Applets

A Java applet can have any or all of the following advantages:

- The same applet can work on "all" installed versions of Java at the same time, rather than just the latest plug inversion only. However, if an applet requires a later version of the Java Runtime Environment (JRE) the client will be forced to wait during the large download.
- The applet naturally supports the changing user state, such as figure positions on the chessboard.
- Java applets are fast and can even have similar performance to native installed software.

IV. Disadvantages of an Applet

A Java applet may have any of the following disadvantages:

- It requires the Java plugin.
- Some applets require a specific JRE. This is discouraged.
- If an applet requires a newer JRE than available on the system, or a specific JRE, the user running it the first time will need to wait for the large JRE download to complete.

Conclusion

We discussed about the Applet security in two browsers Netscape Navigator and Internet Explorer. The recent zero-day security flaw allowed applets to bypass the Java security sandbox, granting themselves permission to execute arbitrary code. Web developers are plagued by the inconsistent browser rendering of HTML and JavaScript, yet most continue to use HTML forms to build GUI frontends. If the mobile code paradigm is going to work, security concerns should be addressed in the language of the content itself. Further, java applets run within what was known as the sandbox - a secure environment which restricted access of the applet to the user's computer. Ultimately, the attractiveness of Applets comes down to their usability: given the wide range of Java technologies.

References

[1] V. Anupam and A. Mayer, "Security of Web Browser Scripting Languages: Vulnerabilities, Attacks, and Remedies, " Proc. Seventh USENIX Security Symp., pp. 187-199, Jan.1998.

[2] D. Balfanz and E. W. Felten, " A Java Filter, " Technical Report 567-97, Dept. of Computer Science, Princeton Univ., Oct. 1997.

[3] D. Balfanz and L. Gong, "Experience with Secure Multiprocessing in Java, " Proc. 18th Int'l Conf. Distributed Computing Systems, May 1998.

[4] D. B. Chapman and E. D. Zwicky, Building Internet Firewalls. O'Reilly & Associates, Sept. 1995.

[5] W. R. Cheswick and S. M. Bellovin, Firewalls and Internet Security, Repelling the Wily Hacker. Addison-Wesley, 1994.

[6] G. Czajkowski and T. von Eicken, " JRes: A Resource Accounting Interface for Java," Proc. ACM OOPSLA Conf., Oct. 1998.

[7] D. Flanagan, Java in a Nutshell, second ed. O'Reilly & Associates, 1997.

[8] L. Gong, "Java Security: Present and Near Future, "IEEE Micro, vol. 17, no. 3, pp.14-19, May/June 1997.