16
Sicurezza programmi Java

16.1 Meccanismi sicurezza

Meccanismi sicurezza:

- caratteristiche linguaggio (verifica confini array, no aritmetica puntatori, controllo casting)
- verifica del bytecode (inizializzazione variabili, chiamate dei metodi, controllo accesso dati e metodi privati, controlli sull'accesso allo stack)
- controllo accesso alle risorse (file, reti)
- firma del codice

16.2 Il caricatore delle classi

Il compilatore Java traduce il sorgente in linguaggio macchina per una macchina ipotetica, la macchina virtuale Java.

Il codice nel linguaggio macchina per la macchina virtuale viene memorizzato nei file .class

La macchina virtuale dispone di un meccanismo per il caricamento dei file delle classi

Se la classe da caricare contiene campi o sovracclassi di altre classi vengono caricati anche i relativi file (risoluzione delle classi)

La macchina virtuale esegue il metodo main (essendo statico non è necessario creare istanze della classe)

Se il metodo main o altri metodi lo richiedono vengono caricate eventuali classi aggiuntive

Le classi di sistema vengono caricate da un caricatore di classi bootstrap (scritto in C)

I browser utilizzano per ogni pagina una diversa istanza del caricatore di classi per le applet
E' possibile creare caricatori di classe personalizzati.

Utilizzando file .class crittografati è possibile autenticare l'utente della classe (senza la chiave non e' possibile eseguire il codice)

Esempio

```
/** @version 1.00 1997-09-10   @author Cay Horstmann          */

 /* chiave: X          chiave inversa: 256-X*/

import java.io.*;

public class Caesar
{  public static void main(String[] args)
   {  if (args.length != 3)
      {  System.out.println("USAGE: java Caesar in out key");
         return;
      }

      try
      {  FileInputStream in = new FileInputStream(args[0]);
         FileOutputStream out = new FileOutputStream(args[1]);
         int key = Integer.parseInt(args[2]);
         int ch;
         while ((ch = in.read()) != -1)
         {  byte c = (byte)(ch + key);
            out.write(c);
         }
         in.close();
         out.close();
      }
      catch(IOException e)
      {  System.out.println("Error: " + e);
      }
   }
}
```

```
/* @version 1.10 1999-10-04     @author Cay Horstmann          */

import java.util.*;  import java.io.*;    import java.lang.reflect.*;
import java.awt.*;  import java.awt.event.*;

//i file vanno compilati col java 1.1 e tutti i .class ottenuti vanno crittografati

public class ClassLoaderTest
{  public static void main(String[] args)
   {  Frame f = new ClassLoaderFrame();
      f.show();
   }
}

class ClassLoaderFrame extends Frame
{  public ClassLoaderFrame()
   {  setTitle("ClassLoaderTest");
      setSize(300, 200);
      addWindowListener(new WindowAdapter()
                  {  public void windowClosing(WindowEvent e)
                  {  System.exit(0);  }} );
      setLayout(new GridBagLayout());
      GridBagConstraints gbc = new GridBagConstraints();
      gbc.weightx = 0;      gbc.weighty = 100;  gbc.fill = GridBagConstraints.NONE;
      gbc.anchor = GridBagConstraints.EAST;
      add(new Label("Class"), gbc, 0, 0, 1, 1);
      add(new Label("Key"), gbc, 0, 1, 1, 1);
      gbc.weightx = 100;      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.anchor = GridBagConstraints.WEST;
      add(nameField, gbc, 1, 0, 1, 1);
      add(keyField, gbc, 1, 1, 1, 1);
      gbc.fill = GridBagConstraints.NONE;
     gbc.anchor = GridBagConstraints.CENTER;
      Button loadButton = new Button("Load");
      add(loadButton, gbc, 0, 2, 2, 1);
      loadButton.addActionListener( new ActionListener()
                        {public void actionPerformed(ActionEvent event)
                        {runClass(nameField.getText(), keyField.getText());}});
   }
```

```
  public void add(Component c, GridBagConstraints gbc, int x, int y, int w, int h)
  {gbc.gridx = x;    gbc.gridy = y;       gbc.gridwidth = w;          gbc.gridheight = h;
    add(c, gbc);
  }

  public void runClass(String name, String key)
  {  try
    {  ClassLoader loader = new CryptoClassLoader(Integer.parseInt(key));
       Class c = loader.loadClass(name);
       String[] args = new String[] {};

       Method m = c.getMethod("main", new Class[] { args.getClass() });
       m.invoke(null, new Object[] { args });
    }
    catch (Throwable e) { System.out.println(e);}
  }

  private TextField keyField = new TextField("3", 4);
  private TextField nameField = new TextField(30);
}

class CryptoClassLoader extends ClassLoader
{
 // private Map classes = new HashMap();
 private Hashtable classes = new Hashtable();
 private int key;

public CryptoClassLoader(int k)
  {key = k;}
```

```
protected synchronized Class loadClass(String name,boolean resolve) throws
                                                    ClassNotFoundException
  {  // check if class already loaded
     Class cl = (Class)classes.get(name);

     if (cl == null) // new class
     {  try
        {  return findSystemClass(name);}   // check if system class
        catch (ClassNotFoundException e) {}
        catch (NoClassDefFoundError e) {}

        // load class bytes--details depend on class loader

        byte[] classBytes = loadClassBytes(name);
        if (classBytes == null)  throw new ClassNotFoundException(name);
        cl = defineClass(name, classBytes,0, classBytes.length);
        if (cl == null) throw new ClassNotFoundException(name);
        classes.put(name, cl); // remember class
     }
     if (resolve) resolveClass(cl);
     return cl;
  }

  private byte[] loadClassBytes(String name)
  {  String cname = name.replace('.', '/') + ".caesar";
     FileInputStream in = null;
     try
     {  in = new FileInputStream(cname);
        ByteArrayOutputStream buffer = new ByteArrayOutputStream();
        int ch;
        while ((ch = in.read()) != -1)
        {  byte b = (byte)(ch - key);
           buffer.write(b);
        }
        in.close();
        return buffer.toByteArray();
     }
     catch (IOException e)
     {  if (in != null) {  try { in.close(); } catch (IOException e2) { }}
        return null;
     }
  }
}
```

16.3 Gestori sicurezza e autorizzazioni (JDK 1.2)

Dopo il caricamento delle classi e la verifica del bytecode i gestori di sicurezza verificano l'ammissibilità delle operazioni del programma

I programmi non hanno un gestore predefinito, ma è possibile istallarlo con setSecurityManager della classe System

JDK 1.0: classi locali hanno autorizzazioni illimitate, classi remote non hanno accesso alle risorse locali

JDK 1.1: il codice remoto firmato da una entità di fiducia ottiene le stesse autorizzazioni delle classi locali

JDK 1.2: è possibile nuove classi di autorizzazione

La politica di sicurezza è definita nei file di politiche:

java.policy (nella directory principale della piattaforma Java)
.java.policy (nella directory principale dell'utente)

le posizioni dei file sono definite (e possono essere cambiate) nel file jva.security nella sottodirectory jre/lib di JDK

file di politiche possono anche essere definiti localmente

```
java -Djava.security.policy=WeatherAppication.policy  WeatherAppication
appletviewer -J-Djava.security.policy=WeatherApplet.policy  WeatherApplet.html
```

= aggiunge il file alle altre politiche in atto
== viene usato solo il file di politiche specificato

```
grant
{ permission java.net.SocketPermission "iwin.nws.noaa.gov:80", "connect";};
```

```
appletviewer -J-Djava.security.policy=WeatherApplet.policy  WeatherApplet.html
```

le applicazioni non installano un gestore di sicurezza; per farlo:

```
        System.setSecurityManager(new SecurityManager());
```

oppure

```
        java -Djava.security.manager -Djava.security.policy =
                WeatherAppication.policy  WeatherAppication
```

Struttura file di politiche:

grant [<originecodice>]
{ <autorizzazione_1>
<autorizzazione_2>
. . .
};

<originecodice>::= **codeBase** “url” | **signedBy** “firma”
omesso se si riferisce a tutte le origini per il codice

<autorizzazione>::= **permission** <destinazione> **,** <azioni>**;**

Classi	Destinazione	Azioni
java.io.FilePermission	<file>	read, write, delete, execute
java.net.SocketPermission	<host>[porta]	accept, connect, listen, resolve
java.util.PropertyPermission	<target>	read, write
java.lang.RuntimePermission	. . .	
java.awt.AWTPermission	. . .	
java.net.NetPermission	. . .	
java.lang.reflect.ReflectPermission	. . .	
java.io.SerializablePermission	. . .	
java.security.SecurityPermission	. . .	

<file>::= nomefile | directory**/** | directory**/*** | ***** | directory**/-** | **-** | **<<ALL FILES>>**
<host> ::= nomehost | indirizzoIP | **localhost** | **“”** | ***.**suffissoDominio | *****
<porta> ::= **:**n | **:-**n | **:**n**-** | **:**n1**-**n2
<target> ::= proprietà | PrefissoProprietà**.***

Esempi

permission java.io.FilePermision “/myapp/-“, “read, write, delete”;
permission java.io.FilePermision “c:\\myapp\\-“, “read, write, delete” ;

permission java.io.FilePermision “${user.home}${/}“, “read, write, delete” ;

permission java.net.SocketPermission "iwin.nws.noaa.gov:80", "connect";

permission java.net.SocketPermission "*.aaa.com:8000-8999", "connect";

i file di politiche possono essere scritti utilizzando policytool

16.4 Sunto di un messaggio

sunto (digest): impronta digitale elettronica di un blocco di dati

- cambiando anche pochi bit del messaggio cambia il sunto
- un falsario che possieda l'originale del messaggio non può generare un falso con lo stesso sunto

algoritmi: SHA1
MD5

sunti di 20 byte (2^{160} impronte possibili)

Esempio

** @version 1.10 1999-10-05 @author Cay Horstmann*/

```
import java.io.*; import java.security.*; import java.awt.*;  import java.awt.event.*;

public class MessageDigestTest
{  public static void main(String[] args)
   {Frame f = new MessageDigestFrame();   f.show(); }
}

class MessageDigestFrame extends Frame
{  public MessageDigestFrame()
  {setTitle("MessageDigestTest");          setSize(400,200);          setLayout(new
BorderLayout());
     addWindowListener( new WindowAdapter()
       {  public void windowClosing(WindowEvent e){System.exit(0);}});

     Panel panel = new Panel();
     CheckboxGroup group = new CheckboxGroup();
     ItemListener listener =  new ItemListener()
       {  public void itemStateChanged(ItemEvent event)
             {Checkbox b = (Checkbox)event.getItemSelectable();
             setAlgorithm(b.getLabel());}
       };
     addCheckbox(panel, "SHA-1", group, true, listener);
     addCheckbox(panel, "MD5", group, false, listener);

     add(panel, "North");    add(message, "Center");   add(digest, "South");
     digest.setFont(new Font("Monospaced", Font.PLAIN, 12));
```

```
  setAlgorithm("SHA-1");
  MenuBar menuBar = new MenuBar();
  Menu menu = new Menu("File");
  MenuItem fileDigestItem = new MenuItem("File digest");
  fileDigestItem.addActionListener(  new ActionListener()
    {  public void actionPerformed(ActionEvent event)
      {  loadFile();} });
  menu.add(fileDigestItem);
  MenuItem textDigestItem = new MenuItem("Text area digest");
  textDigestItem.addActionListener(  new ActionListener()
            {  public void actionPerformed(ActionEvent event)
                  {  String m = message.getText();
                        computeDigest(m.getBytes());} });
  menu.add(textDigestItem);
  menuBar.add(menu);
  setMenuBar(menuBar);
}

public void addCheckbox(Panel c, String name,
  CheckboxGroup g, boolean selected, ItemListener listener)
{  Checkbox b = new Checkbox(name, selected,g);
  c.add(b);
  b.addItemListener(listener);
}

public void setAlgorithm(String alg)
{  try
  {  currentAlgorithm = MessageDigest.getInstance(alg);
    digest.setText("");
  }
  catch(NoSuchAlgorithmException e) {  digest.setText("" + e); }
}

public void loadFile()
{  FileDialog chooser = new FileDialog(this,"File dialog");
  chooser.setDirectory(".");
  chooser.setMode(FileDialog.LOAD);
  chooser.show();
  String nomef = chooser.getFile();
  String nomed = chooser.getDirectory();
  if (nomef != null)
     computeDigest(loadBytes(nomed+nomef));
}
```

```
  public byte[] loadBytes(String name)
  {  FileInputStream in = null;

     try
     {  in = new FileInputStream(name);
        ByteArrayOutputStream buffer
           = new ByteArrayOutputStream();
        int ch;
        while ((ch = in.read()) != -1)
           buffer.write(ch);
        return buffer.toByteArray();
     }
     catch (IOException e)
     {  if (in != null)
        {  try { in.close(); } catch (IOException e2) {} }
        return null;
     }
  }

  public void computeDigest(byte[] b)
  {  currentAlgorithm.reset();
     currentAlgorithm.update(b);
     byte[] hash = currentAlgorithm.digest();
     String d = "";
     for (int i = 0; i < hash.length; i++)
     {  int v = hash[i] & 0xFF;
        if (v < 16) d += "0";
        d += Integer.toString(v, 16).toUpperCase() + " ";
     }
     digest.setText(d);
  }

  private TextArea message = new TextArea();
  private TextField digest = new TextField();
  private MessageDigest currentAlgorithm;
}
```

16.5 Firme digitali

Crittografia a chiave pubblica

Chiave pubblica e chiave privata, l'una inversa dell'altra

Algoritmi:
- RSA (www.rsa.com)
- DSA

Un messaggio cifrato con una chiave pubblica può essere decifrato solo dal proprietario della corrispondente chiave privata

Un messaggio firmato con una chiave privata può essere autenticato solo con la corrispondente chiave pubblica

Si genera una coppia di chiavi inserendo il risultato di un processo casuale in una procedura deterministica

Firma e autenticazione:

tizio scrive un messaggio e ne firma il sunto con la sua chiave privata

caio, utilizzando la chiave pubblica corrispondente verifica che il messaggio originale non è stato contraffatto e proviene veramente da tizio

Esempio

```
/** @version 1.00 1997-09-10 @author Cay Horstmann */

import java.security.*;

public class SignatureTest
{  public static void main(String[] args)
   {  try
      {  KeyPairGenerator keygen = KeyPairGenerator.getInstance("DSA");
         SecureRandom secrand = new SecureRandom();
         keygen.initialize(512, secrand);
//generazione prima chiave
         KeyPair keys1 = keygen.generateKeyPair();
         PublicKey pubkey1 = keys1.getPublic();
         PrivateKey privkey1 = keys1.getPrivate();
```

```
//generazione seconda chiave
        KeyPair keys2 = keygen.generateKeyPair();
        PublicKey pubkey2 = keys2.getPublic();
        PrivateKey privkey2 = keys2.getPrivate();
//firma utilizzando prima chiave
        Signature signalg = Signature.getInstance("DSA");
        signalg.initSign(privkey1);
        String message   = "Pay authors a bonus of $20,000.";
        signalg.update(message.getBytes());
        byte[] signature = signalg.sign();
//verifica utilizzando prima chiave
         Signature verifyalg = Signature.getInstance("DSA");
         verifyalg.initVerify(pubkey1);
        verifyalg.update(message.getBytes());
        if (!verifyalg.verify(signature)) System.out.print("not ");
        System.out.println("signed with private key 1");
//verifica utilizzando seconda chiave
        verifyalg.initVerify(pubkey2);
        verifyalg.update(message.getBytes());
        if (!verifyalg.verify(signature))  System.out.print("not ");
        System.out.println("signed with private key 2");
      }
     catch(Exception e)
     {  System.out.println("Error " + e);}
   }
}
```

per stabilire l'identità di un estraneo ci si avvale della garanzia di una terza parte di fiducia (certificato): si garantisce che una data entità possiede una data chiave pubblica

Verisign (www.verisign.com) ID classe 1 garantisce l'indirizzo di e-mail
ID classe 3 richiede di comparire davanti ad un pubblico ufficiale

JDK1.2

X.509

keytool consente la generazione e la gestione di gruppi di certificati

keytool gestisce raccolte di chiavi, cioè database di certificati e chiavi; ogni voce della raccolta ha un alias:

keytool -genkey -keystore angela.store -alias angela

keytool -export -keystore angela.store -alias angela -file angela.cert

il certificato viene trasmesso; fabrizio lo importa

keytool –printcert –file angela.cert

keytool -import -keystore fabrizio.store -alias angela -file angela.cert

keytool -list -keystore fabrizio.store
Owner: CN=angela rossi, OU=corso, O=unipg, L=perugia, ST=pg, C=it
Issuer: CN=angela rossi, OU=corso, O=unipg, L=perugia, ST=pg, C=it
Serial number: 3a6dc25b
Valid from: Tue Jan 23 18:41:47 GMT+01:00 2001 until: Mon Apr 23 19:41:47 GMT+02:00 2001
Certificate fingerprints:
MD5: 23:C6:9F:02:19:D1:57:A8:1C:DB:EF:2B:AF:44:E2:66
SHA1: B3:59:AC:44:F9:8B:2A:B7:92:35:74:69:89:56:CF:13:CF:CB:4B:E6

Fabrizio può telefonare ad angela a chiederle di leggerne l'impronta

angela ora può mandare fabrizio documenti firmati:

jar cvf documento.jar documento.txt

jarsigner –keystore angela.store documento.jar angela

fabrizio riceve il documento e ne verifica la firma:

jarsigner –verify –keystore fabrizio.store documento.jar

fabrizio estrae il documento:

jar xvf documento.jar
extracted: META-INF/MANIFEST.MF
extracted: META-INF/ANGELA.SF
extracted: META-INF/ANGELA.DSA
created: META-INF/
extracted: documento.txt

In generale un certificato viene firmato da un intermediario di fiducia (p.e. ACME Software)

Angela → Sara

La ACME manda il proprio certificato a Sara

keytool –genkey –keystore acmesoft.store –alias acmeroot

keytool –export –alias acmeroot –keystore acmesoft.store –file acmeroot.cert

che lo inserisce fra le sue chiavi (dopo averne verificato l'autenticità)

keytool -import -keystore sara.store -alias acmeroot -file acmeroot.cert

il programma seguente legge un file contenente un certificato e lo firma con una chiave privata proveniente da una raccolta (nell'esempio è un rappresentante di ACME ad eseguire il programma per controfirmare il certificato inviato da Angela):

java CertificateSigner –keystore acmesoft.store –alias acmeroot –infile angela.cert –outfile angela_signedby_acmeroot.cert

ora Alice può inviare il file angela_signedby_acmeroot.cert a Sara

quando Sara importa il certificato si verifica che questo è stato controfirmato con una chiave già presente; il certificato di Angela viene aggiunto senza ulteriori conferme

keytool -import -keystore sara.store -alias angela -file angela_signedby_acmeroot.cert

```
/*@version 1.00 1999-10-23     * @author Cay Horstmann        */

import java.io.*; import java.security.*;
import java.security.cert.*; import java.util.*;

import sun.security.x509.X509CertInfo;
import sun.security.x509.X509CertImpl;
import sun.security.x509.X500Name;
```

```
import sun.security.x509.CertificateIssuerName;

public class CertificateSigner
{  public static void main(String[] args)
   {  String ksname = null; // the keystore name
      String alias = null; // the private key alias
      String inname = null; // the input file name
      String outname = null; // the output file name
      for (int i = 0; i < args.length; i += 2)
      {  if (args[i].equals("-keystore"))  ksname = args[i + 1];
         else if (args[i].equals("-alias")) alias = args[i + 1];
         else if (args[i].equals("-infile")) inname = args[i + 1];
         else if (args[i].equals("-outfile")) outname = args[i + 1];
         else usage();
      }
      if (ksname == null || alias == null || inname == null || outname == null) usage();
      try  {  PushbackReader console =
                     new PushbackReader(newInputStreamReader(System.in));
         KeyStore store = KeyStore.getInstance("JKS", "SUN");
         InputStream in = new FileInputStream(ksname);
         char[] password = readPassword(console, "Keystore password");
         store.load(in, password);         Arrays.fill(password, ' ');     in.close();
         char[] keyPassword = readPassword(console, "Key password for " + alias);
         PrivateKey      issuerPrivateKey          =      (PrivateKey)store.getKey(alias,
keyPassword);
         Arrays.fill(keyPassword, ' ');
         if (issuerPrivateKey == null)  error("No such private key");
         in = new FileInputStream(inname);
         CertificateFactory factory = CertificateFactory.getInstance("X.509");
         X509Certificate inCert = (X509Certificate)factory.generateCertificate(in);
         in.close();
         byte[] inCertBytes = inCert.getTBSCertificate();
         X509Certificate issuerCert = (X509Certificate)store.getCertificate(alias);
         Principal issuer = issuerCert.getSubjectDN();
         String issuerSigAlg = issuerCert.getSigAlgName();
         FileOutputStream out = new FileOutputStream(outname);
         X509CertInfo info = new X509CertInfo(inCertBytes);
  info.set(X509CertInfo.ISSUER, new CertificateIssuerName((X500Name)issuer));
         X509CertImpl outCert = new X509CertImpl(info);
         outCert.sign(issuerPrivateKey, issuerSigAlg);
         outCert.derEncode(out);
         out.close();
      }
```

```
    catch (Exception exception) {  System.out.println(exception);}
  }

public static char[] readPassword(PushbackReader in,  String prompt)
        throws IOException
  {  System.out.print(prompt + ": ");
    System.out.flush();
    final int MAX_PASSWORD_LENGTH = 100;
    int length = 0;
    char[] buffer = new char[MAX_PASSWORD_LENGTH];

    while (true)
    {  int ch = in.read();
      if (ch == '\r' || ch == '\n' || ch == -1 || length ==
MAX_PASSWORD_LENGTH)
      {  if (ch == '\r') // handle DOS "\r\n" line ends
        {  ch = in.read();
          if (ch != '\n' && ch != -1) in.unread(ch);
        }
        char[] password = new char[length];
        System.arraycopy(buffer, 0, password, 0, length);
        Arrays.fill(buffer, ' ');
        return password;
      }
      else
      {  buffer[length] = (char)ch;
        length++;
      }
    }
  }

  public static void error(String message)
  {  System.out.println(message);
    System.exit(1);
  }

  public static void usage()
  {  System.out.println("Usage: java CertificateSigner"
      + " -keystore keyStore -alias issuerKeyAlias"
      + " -infile inputFile -outfile outputFile");
    System.exit(1);
  }
}
```

16.6 Firma del codice

- si utilizza l'autenticazione per verificare la provenienza del codice
- si esegue il codice con la politica di sicurezza appropriata

Firma di applet per intranet

L'amministratore installa i certificati ed i file di politiche sulle macchine locali

Le nuove applet vengono firmate ed installate sul server Web

La gestione dei file di politiche spetta agli amministratori di sistema

jar cvf FileReadApplet.jar *.class

jarsigner –keystore acmesoft.store FileReadApplet.jar acmeroot

il file di politiche assume la seguente forma:

keystore "file:acmesoft.store", "JKS";

grant signedBy "acmeroot"

{ permission java.io.FilePermission "<<ALL FILES>>", "read";};

appletviewer –J-Djava.security.policy=applets.policy FileReadApplet.html

<APPLET CODE="FileReadApplet.class" ARCHIVE="FileReadApplet.jar" WIDTH=400 HEIGHT=300></APPLET>

/*@version 1.00 1999-10-23 * @author Cay Horstmann */

import java.awt.*; import java.awt.event.*; import java.io.*;
import java.util.*; import java.applet.*;

```
public class FileReadApplet extends Applet
{
  TextField fileNameField;
  TextArea fileText;

public FileReadApplet()
  {  fileNameField = new TextField(20);
     Panel panel = new Panel();
     panel.add(new Label("File name:"));      panel.add(fileNameField);
     Button openButton = new Button("Open");       panel.add(openButton);
     openButton.addActionListener(new ActionListener()
        {  public void actionPerformed(ActionEvent event)
           {  loadFile(fileNameField.getText());}});
     add(panel, "North");
     fileText = new TextArea();  add(fileText, "Center");
  }

  public void loadFile(String filename)
  {  try
     {  fileText.setText("");
        BufferedReader in = new BufferedReader(new FileReader(filename));
        String s;
        while ((s = in.readLine()) != null)
        fileText.append(s + "\n");
        in.close();
     }
     catch (IOException e)
     {  fileText.append(e + "\n");}
     catch (SecurityException e)
     {  fileText.append("I am sorry, but I cannot do that.");}
  }
}
```

E' possibile spostare i file di politiche sul server intranet modificando il file ${java.home}/lib/security/java.security aggiungendo:

policy.url.3=http://intranet.acmesoft.com/admin/applet.policy
keystore "http://intranet.acmesoft.com/admin/acmesoft.store" ,"JKS";

per provare l'applet utilizzando il plug-in Java è necessario modificare il file jre/lib/security/java.security aggiungendo:
policy.url.3=file://home/test/applet.policy

Netscape ed Internet Explorer richiedono strumenti specifici:
www.securingjava.com/appdx-c/

Firma di applet per Internet

I produttori di software ottengono certificati controfirmati da autorità di certificazione (Verisign, Thawte, a pagamento)

Le pagine web che contengono applet firmate visualizzano una finestra di dialogo che identifica il produttore e consente di scegliere se eseguire l'applet con pieni privilegi o no

Esempio (firma per Internet Explorer)

1) ottenere un certificato
2) scaricare Microsoft Java SDK.
   www.microsoft.com/msdownload/java/sdk/31f/SDK-JAVA.asp.
   C:\SDK-Java.31.
   PATH=%PATH%;C:\SDK-Java.31\Bin\PackSign
3) generare un file .cab (non .jar!)
   cabarc N test.cab file1.class file2.class
4) firmare il codice:

signcode -j JavaSign.dll -jp High -spc a:\Cert.spc -v a:\Key.pvk -n "My Applet"
-i http://www.mywebpage.com/ test.cab

-j JavaSign.dll: si sta firmando del codice Java
-jp High: tipo di privilegio; (High, Medium, Low); è un parametro per la dll
-spc -v: per indicare i file del certificato e della chiave
-n: nome del sodtware
-i: dove trovare informazioni sul codice

5) verifica che tutto è andato bene:
   chkjava test.cab
6) aggiornare il tag HTML:

```
<APPLET CODE="MyApplet.class">
<PARAM name="cabbase" VALUE="myapp.cab"></APPLET>
```

Generazione di un certificato di esempio:
makecert -sk Key.pvk -n "CN=Your Name" Cert.cer
abilitare il certificato per la firma:
cert2spc Cert.cer Cert.spc
alla fine si utilizzano i due file:
Key.pvk Cert.spc