5-2008

I Resign! Resigning Jar Files with Initium

Douglas A. Lyon
Fairfield University, dlyon@fairfield.edu

Published Citation

Douglas Lyon, "I Resign! Resigning Jar Files with Initium.", Journal of Object Technology, Volume 7, no. 4 (May 2008), pp. 17-27

JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©JOT, 2008

Vol. 7, No. 4, May-June 2008

I Resign! Resigning Jar Files with Initium.

Douglas Lyon, Ph.D.

Abstract

This paper describes how to resign Jar files. Jar files (Java archives) are used by technologies (like Java Web Start) to deploy applications that are run with increased privileges. The Jar files are signed with certificates that generally expire after a year. The annual resigning of the files is therefore an event that occurs after the signer has forgotten how resigning is done.

Manual resigning of Jar files is a tedious and error-prone task. All the more so when there are many of them. This article shows how to automate the task. Considering that I have over 250 Jar files that have to be resigned each year, a manual task is not an option.

The methodology for resigning a Jar file cannot include signing the Jar file twice. Jar files that are signed twice create an error during verification. Unsigning a Jar file is not a straightforward task. Thus, our approach is to expand the Jar file, remove the expired certificate and then repack and resign the Jar file with the new certificate. We also show how to obtain a new certificate, from a free certificate provider.

This paper addresses a sub-problem of the *Initium* project, a joint, on-going project between the Fairfield University and the DocJava, Inc. Initium is a Latin word that means: “at the start”.

1 THE PROBLEM

Every year the number of Jar files that we have to deploy grows. Some of the Jar files are primary containers of applications (with a *main* method). Other Jar files are containers of commonly used libraries. Still other Jar files are containers of native methods. Regenerating the Jar files, with a new certificate, is what we term *resigning the Jars*. This annual event requires that we:

1. get a new certificate,
2. load the certificate into our keystore(s),
3. unjar the already signed Jar files,
4. rejar the directories into Jar files of the same name,
5. sign the Jar files and
6. verify the Jar file signatures.

The procedure takes several minutes. Even so, we have often performed the procedure on the server, as this is where most of the Jar files reside. Further, the new certificate

Cite this column as follows: Douglas A. Lyon “I Resign! Resigning Jar Files with Initium.”, in *Journal of Object Technology*, vol. 7, no. 4, May-June 2008, pp. 9-27

needs to be distributed to various machines. Development machines and the web-server, all have key stores. Each of these machines is capable of signing Jar files. As a result, each key store must be a recipient of the new certificate. This is typically done by installing the certificate into one key store, then copying the key store file from one machine to another.

2 GETTING A NEW CERTIFICATE

The first assumption behind the renewal of a certificate is that you already have an old one. If this is not the case, you should probably make reference to [Lyon 2004].

In order to get a new certificate, proceed to the Certificate Authority (CA) web site. In my case, I go to the Thawte web site at https://www.thawte.com/cgi/personal/cert/enroll.exe and login.

Figure 2-1. Request a Certificate

Select the link labeled “Developers of New Security Applications ONLY”. Then select “test”. In the next dialog, select “Paste-in CSR Certificate Enrollment”, as shown in Figure 2-2.

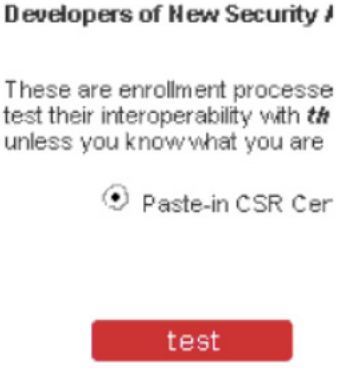


Figure 2-2. Select a “Paste-in CSR Certificate”

Figure 2-2 shows the pop-up dialog that asks to create a CSR (Certificate Signing Request). Select “Test” in order to past in the CSR.

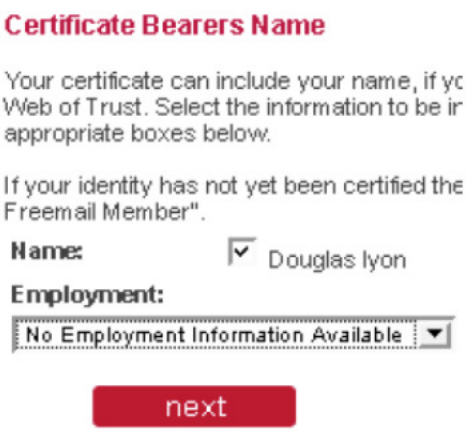


Figure 2-3. Confirm the Certificate Name

Figure 2-3 confirms that the recipient is correct.

Figure 2-4. E-mail confirmation

Figure 2-4 confirms the e-mail address is correct.

Figure 2-5. Final Confirmation

Figure 2-5 shows that a "final confirmation" is needed.

Figure 2-6. CSR Dialog

Figure 2-6 shows the CSR dialog, where the CSR is to be pasted in.

Generate a public key using *keytool*.

keytool -genkey -keyalg RSA -alias docjavaInc

Now to get my certificate:

keytool -certreq -file keystore.txt -alias docjavaInc

Here is the output from the certificate request:

more keystore.txt

-----BEGIN NEW CERTIFICATE REQUEST-----

MIME Encoded Data

-----END NEW CERTIFICATE REQUEST-----

This is pasted into the Thawte web form to yield a distinguished certificate.

Now I proceed to my Thawte accounts' certificate manager page, at: https://www.thawte.com/cgi/personal/cert/status.exe and check the status of my X509 certificate request. After a few minutes, it changes status from "*pending*" to "*issued*" and I fetch my new X.509 certificate.

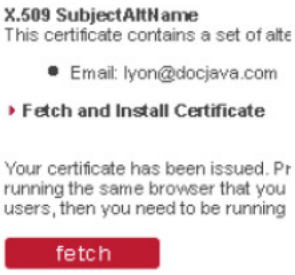


Figure 2-7. Fetch Your Certificate

Figure 2-7 shows the area that enables retrieval of the certificate.

In order to import the certificate into the key store, the PKCS7 part of the certificate must be saved into a file and then altered so that it looks like the following:

-----BEGIN PKCS #7 SIGNED DATA-----

MIII7AYJKoZIhv…

The *Initium* GUI has a *cleanThawtes* menu item that automates the processing of certitificates returned by Thawte. It is based on the work of Richard Dallaway [Dallaway].

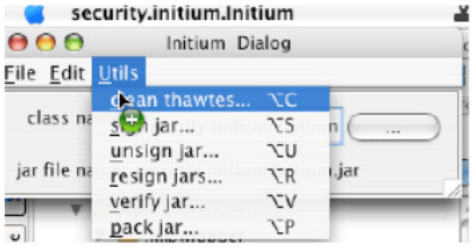
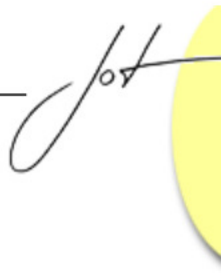


Figure 2-8. The Initium Dialog

Figure 2-8 shows the menu item being selected for cleaning the returned certificate under to the *pk8.cert format*. This is imported into the keystore file using:

keytool -import -file pk8.cert -alias docjavainc –trustcacerts



Errors in the parameters passed to the *keytool* command result in cryptic exceptions. For example:

```
keytool -import  -file pk8.cert -trustcacerts
Enter keystore password:
keytool error: java.lang.Exception: Input not an X.509 certificate
```

The default location for the keystore file is "*.keystore"* and is located in the users' home directory. In order to deal with the case of an improperly formatted certificate, I have created a more sophisticated way to get certificates, one that invokes the Thawte cleaner (that is, the API can import the improperly formatted certificates form Thawte). Thus we are able to run the Initium key wizard under a series of different situations. For example, when no *.keystore* file appears in the users' home, the program will prompt you for a key store file. If you don't have one, the program will offer to make one for you, adding a self-signed certificate. The program will also offer to create a certificate request file, which can be used to obtain a trusted certificate. Thus easing key management, a little.

3 RESIGNFILE – SHELL SCRIPTS AND ANT

A shell script can be used to resign Jar files. Here are the basic steps to resigning a Jar file:

1. Create a temporary directory (e.g., *foo*).
2. Move the Jar file into the *foo* directory.
3. Decompress the Jar file.
4. Remove the META-INF directory and old Jar file.
5. Create a new Jar file and sign it with the new certificate.

A csh script, called *resignFile* follows:

```
#! /bin/csh -f

# Usage: for_each_file file(s)

if ($#argv == 0) then          # arguments?
    echo "Usage: $0 file(s)"
    exit 1
endif
rm -rf foo
mkdir foo
foreach file ($argv[*])        # expand argument word list
    echo $file
    mv $file foo
    cd foo
    jar -xf $file
```

```
        rm -rf META-INF
        rm $file
        jar -cf $file .
        jarsigner -storepass plainTextPasswordHere $file docjava
        mv $file ..
        cd ..
    end
```

There are several problems with such a script. First, the *plain-text* password is embedded in the source code (as well as the alias for the certificate). These problems can be resolved by changing these elements from literals embedded in the code into command-line parameters. Worse, there is no verification of the Jar file and the script is not cross-platform.

Existing technologies for resigning Jar files are typically based in *ant*. For example:

```
<signjar jar="${dist}/lib/ant.jar"
alias="docjava" storepass="plainTextPasswordHere"/>
```

The *ant*-based technology is cross-platform, but it suffers from an embedded plain-text password. A variant on the use of the *signjar* task avoids (or at least moves) the plain-text parameters:

```
<signjar keystore="${keystore.file}" storepass="${keystore.passwd}"
alias="${alias}" lazy="true" jar="${jarFile}" />
```

We note that alias names can be longer than 8 characters, but the *jarsigner* truncates the rsa and sf file base name to 8. For example, in the META-INF directory, the docjavaInc alias is used to create two files, DOCJAVAI.RSA and DOCJAVAI.SF.

If two aliases are not unique for the first 8 characters, the *jarsigner* will fail.

Even worse, webstart will fail if the Jar file is signed by more than one signature. It will also fail if the Jar files in a project are signed by different signatures.

In summary, shell scripts are not cross-platform, and ant scripts (as well as shell scripts) encourage the embedding of plain-text passwords in files.

4 PROGRAMMATIC RESIGNING OF THE JAR FILE

This section describes how we implemented the resigning of Jar files using a custom Java API. The tool addresses the weakness of ANT, in that it does not hold passwords in plaintext files. All information is held in user preferences and is password protected by the users login id. Thus, it should be as secure as the users' account (in theory). The other weakness that this program addresses is one of portability. The shellscript approach only works under Unix. Our approach is deployed as a webstart application and should work on several platforms. Here are the system requirements:

1. JDK 1.5 or better is installed,
2. The key store is on the local disk
3. The user has the key store password and

4. A new certificate is in the key store with a known alias

Figure 4-1. Key store Settings

Figure 4-1 shows that the key store settings populating the key store dialog box after having been saved in preferences. The location of the JDK is set in another dialog (and also persisted in preferences). These things do not change often, and a GUI becomes more user-friendly if it makes use of reasonable defaults.

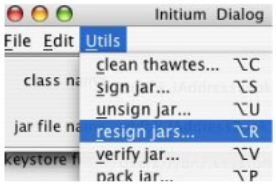


Figure 4-2. Resign jars

Figure 4-2 shows the menu item that brings up the directory-choice dialog box. This allows the user to select the root directory that contains all the Jar files to be resigned. Once the process begins, it is automatic, resigning all Jar files in the root directory and its sub-directories. We have created a class, called *SignUtils* and it contains several overloaded *resignJars* methods. For example:

```
public static void resignJars() throws NoSuchAlgorithmException,
        IOException, CertificateException, KeyStoreException {
    File f = Futil.getReadDir("select a root resign directory");

    DirList dl = new DirList(f, "jar");

    File[] fa = dl.getFiles();
    String[] s = SelectorDialog.getStringSelectorDialog(fa);
    for (String aFile:s){
      resignJar(new File(aFile));
    }
    PrintUtils.print(fa);
  }
```

The *DirList* class does a recursive search for all files whose file name suffix is "jar".

```
public static void resignJar(File file) throws
        NoSuchAlgorithmException, IOException,
        CertificateException, KeyStoreException {
```

```
        unsignJar(file);
        WebStartBean wsb = WebStartBean.getWsbFromPreferences();
        signJar(file, wsb);
    }
```

The *resignJar* method makes use of the *WebStartBean*, parameter for the *signJar* operation. We are unable to modify the contents of a Jar directly, so we decompress the Jar file, delete the signatures in the *META-INF* directory and then re-compress:

```
public static void unsignJar(File inputJar) {
    //File out = Futil.getReadDir("select output dir");
    File out = new File(inputJar.getParent(), "tmp");
    Futil.deleteDirectory(out);
    out.mkdir();
    ZipUtils.unjar(inputJar, out);
    Futil.deleteDirectory(new File(out, "META-INF"));

    File newUnsignedJar = new File(inputJar.getName());
    ZipUtils.createJar(newUnsignedJar, out);
    Futil.deleteDirectory(out);
}
```

We gain access to the *Jar* and *JarSigner* tools, by executing processes on the host operating system. Their location is obtained from:

```
public static File getJarSignerPath() {
    // first try the windows location
    String command = "jarsigner";
    return SystemUtils.getSdkCommand(command);

}

public static File getJarPath() {
  // first try the windows location
  String command = "jar";
  return SystemUtils.getSdkCommand(command);

}
```

Static methods like *getSdkCommand* are ad-hoc implementations of JDK tool locators, custom built for each operating system. Presently, we are only able to run on MacOS, Linux and Windows.

5 CONCLUSION

Programmatic use of the JDK tools has enabled us to create an API that automates the signing of Jar files. This eases maintenance, somewhat. Our technique for locating JDK tools is ad-hoc, and probably does not work reliably on all platforms. This is a problem that could be solved if Sun would release an API that supported the features of these tools. Lacking that, we could write our own (given additional time).

It would be really nice if there were a way to programmatically retrieve certificates from the Certificate Authority.

REFERENCES

[Lyon 2004] "The Initium X.509 Certificate Wizard" by Douglas A. Lyon, *Journal of Object Technology*, vol. 3, no. 10, November-December 2004, pp. 75-88. http://www.jot.fm/issues/issue_2004_11/column6/

About the author

Douglas A. Lyon (M'89-SM'00) received the Ph.D., M.S. and B.S. degrees in computer and systems engineering from Rensselaer Polytechnic Institute (1991, 1985 and 1983). Dr. Lyon has worked at AT&T Bell Laboratories at Murray Hill, NJ and the Jet Propulsion Laboratory at the California Institute of Technology, Pasadena, CA. He is currently the Chairman of the Computer Engineering Department at Fairfield University, in Fairfield CT, a senior member of the IEEE and President of DocJava, Inc., a consulting firm in Connecticut. Dr. Lyon has authored or co-authored three books (Java, Digital Signal Processing, Image Processing in Java and Java for Programmers). He has authored over 30 journal publications. Email: lyon@docjava.com. Web: http://www.DocJava.com.