JPAM User Guide

Version 1.0

Greg Luck

Contents

Chapter 1

Preface

1.1 Audience

The intended audience for this book is developers who use JPam. It should be able to be used to start from scratch, get up and running quickly, and also be useful for the more complex options.

JPam is a missing piece in the Java - Unix security puzzle. Another natural audience is therefore security specialists.

It is also intended for application and enterprise architects. JPam creates new possibilities. It is not necessary to front Java Application servers with Apache in order to plug in native security. You are no longer limited by the availability of Java implementations of security services.

1.2 Book Format

This content is suitable for use as an online PDF or printed. Blank pages have been deliberately left to give a good flow.

1.3 Acknowledgements

JPam has had contributions in the form of forum discussions, feature requests, bug reports, patches and code commits.

Rather than try and list the many hundreds of people who have contributed to JPam in some way it is better to link to the web site where contributions are acknowledged in the following ways:

- Bug reports and features requests appear in the changes report here:
- Patch contributors generally end up with an author tag in the source they contributed to
- Team members appear on the team list page here:

Chapter 2

Introduction

JPam is a Java-PAM bridge. PAM, or Pluggable Authentication Modules, is a standard security architecture used on Linux, Solaris, Mac OS X and other Unix systems. JPam is the missing link between the two.

JPAM permits the use of PAM authentication facilities by Java applications running on those platforms. These facilities include:

- account
- auth
- password
- session

Chapter 3

Java Requirements, Dependencies and Maven POM snippet

3.1 Java Requirements

JPam supports 1.4, 1.5 and 1.6 at runtime. JPam final releases are compiled with -target 1.4. This produces Java class data, version 48.0.

JPam does not work with JDK1.2 or JDK1.1. JAAS is not available for these JDKs. Moreover, JNI used a different interface prior to JDK1.2.

IBM 1.4.2.0 JVM is known to work although it places its native libraries in a different place. Add `-Dnative.java.library.path=/usr/lib/jvm/java-ibm/jre/bin` (or wherever the IBM JVM is installed) to your Java command line.

3.2 Mandatory Java Dependencies

JPam requires commons-logging commons-logging is a very common dependency, and is therefore not included in the distribution.

Jpam also requires JAAS. Originally introduced as an optional package (JAAS 1.0) to version 1.3 of the Java 2 SDK, JAAS has now been integrated into the Java 2 SDK, version 1.4.

3.3 Operating System Dependencies

It has been reported that JPam relies on the presence of pam-devel-0.77-66.2 or similar RPMs.

3.4 Maven pom.xml snippet

JPam releases are placed in the central Maven repository.

The Maven snippet for JPam 1.0 is:

```
<dependency>
   <groupId>net.sf.jpam</groupId>
   <artifactId>jpam</artifactId>
```

```
    <version>1.0</version>
</dependency>
```

Chapter 4

Getting Started

4.1 Supported Operating Systems and Architectures

JPam will create builds for the following:

- Linux x86
- Linux x86_64, including AMD64
- Mac OS X
- Solaris sparc
- HP-UX

  PAM is used on Unix and Unix-like operating systems. JPam should be readily portable to other *nixes.

4.2 Step-by-step Installation Instructions

The steps are:

1. Place the jpam-X.X.jar into your classpath.
2. Ensure that any libraries required to satisfy dependencies are also in the classpath.
3. As an optional step, configure an appropriate logging level.
4. Copy the native library to the Java Native Libary Path. See the table below.

OS and Architecture	Native Library File	Java Native Library Path
Linux AMD64 server	libjpam.so	$JAVA_HOME/lib/amd64/server
Linux i386 client	libjpam.so	$JAVA_HOME/lib/i386/client
Linux i386 server	libjpam.so	$JAVA_HOME/lib/i386/server
Linux x86 client	libjpam.so	$JAVA_HOME/lib/i386/client
Linux x86 server	libjpam.so	$JAVA_HOME/lib/i386/server
Mac OS X PPC	libjpam.jnilib	~/Library/Java/Extensions
Solaris sparc	libjpam.so	$JAVA_HOME/lib/sparc/client
Solaris sparc	libjpam.so	$JAVA_HOME/lib/sparc/server

Native Library Installation Location

Alternately, JPam will search for the native library in the same directory as the JPam jar is located.

5. If using the JAAS API, copy .java.login.config to your home directory.
6. Configure a PAM module for use by JPam.

Chapter 5

Configuration

The distribution contains an example pam.d configuration file called net-sf-jpam.

To configure jpam, edit net-sf-jpam and copy it to /etc/pam.d.

5.1 Format of pam.d files

PAM configuration files have four columns:

- facility

  Possible values are:

  - account - account management
  - auth - authentication
  - management
  - password - password management
  - session - session session management.

- control flag

  The control flag is how the result of the operation is to be interpreted. Possible values are:

  - required - If the module succeeds, the rest of the chain is executed, and the request is granted unless some other module fails. If the module fails, the rest of the chain is also executed, but the request is ultimately denied.
  - requisite - If the module succeeds, the rest of the chain is executed, and the request is granted unless some other module fails. If the module fails, the chain is immediately terminated and the request is denied.
  - sufficient - If the module succeeds and no earlier module in the chain has failed, the chain is immediately terminated and the request is granted. If the module fails, the module is ignored and the rest of the chain is executed.

- module path

  The path to the module

- module arguments

  This column is optional. It is for any options to be passed to the module.

5.2 Example 32 bit net-sf-jpam file

An example which uses the standard Unix password PAM module is shown below.

```
########################################################################
# Unix PAM Module
# ===============
#
# If using pam_unix you may need to change /etc/shadow to be readable by
# the user executing Jpam.
########################################################################
auth required /lib/security/pam_unix_auth.so
account required /lib/security/pam_unix_acct.so
password required /lib/security/pam_unix_passwd.so
session required /lib/security/pam_unix_session.so
```

5.3 Example 64 bit net-sf-jpam file

64 bit Linux distributions such as RedHat, Fedora, Suse and Novell Linux Desktop have adopted a convention of placing their 64 libraries in /lib64. The net-sf-jpam configuration file for these would look like:

```
########################################################################
# Unix PAM Module
# ===============
#
# If using pam_unix you may need to change /etc/shadow to be readable by
# the user executing Jpam.
########################################################################
auth required /lib64/security/pam_unix_auth.so
account required /lib64/security/pam_unix_acct.so
password required /lib64/security/pam_unix_passwd.so
session required /lib64/security/pam_unix_session.so
```

5.4 Configuring Popular Security Services

However, there are many more approaches than these two. There are hundreds of authentication systems accessible through PAM. See a list fo Linux here. Many of these are installed by default in the Linux distributions. For example Fedora Core 3 has 55 PAM modules in its /lib/security directory by default.

Some notable examples of PAM modules are:

Name	Module	Use
SecurId	pam_securid.so	Authenticates SecurId hardware tokens with the ACE Server. Available from RSA.
Unix	pam_unix_+.so	Authenticates using the configured auth Unix scheme. e.g. shadow passwords or NIS.
RADIUS	pam_radius.so	Authenticates using RADIUS servers.
CryptoCard	pam_smxs.so	Authenticates using CryptoCard RB1 hardware tokens and similar.
Samba	pam_winbind.so	Authenticates using Windows and Samba servers.
Kerberos	pam_krb5.so	Authenticates with Kerberos/Active Directory.
LDAP	pam_ldap.so	Authenticates with LDAP servers (from Java you could also use the JNDI API).
SafeWord	pam_safeword.so	Authenticates SafeWord tokens.

Popular Security Services

The path to the module shown above becomes the third column in the net-sf-jpam configuration line.

5.5 Installing Additional PAM Modules

JPam will dynamically link to any Pam module which is installed on the operating system and specified in its configuration. No recompilation is required.

5.6 More information

See http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html for information on configuring PAM.

Chapter 6

Features

- High Quality
  - High Test Coverage
  - Automated Load, Limit and Performance System Tests
  - Production tested
  - Fully documented
  - Conservative Commit policy
  - Full public information on the history of every bug
  - Responsiveness to serious bugs
- Open Source Licensing
  - Apache 2.0 license

6.1 High Quality

6.1.1 High Test Coverage

The JPam team believe that the first and most important quality measure is a well designed and comprehensive test suite.

JPam has a relatively high test coverage of source code. This has edged higher over time.

6.1.2 Automated Load, Limit and Performance System Tests

The JPam JUnit test suite contains some long-running system tests which place high load on different JPam subsystems to the point of failure and then are back off to just below that point. The same is done with limits such as the amount of Elements that can fit in a given heap size. The same is also done with performance testing of each subsystem and the whole together. The same is also done with network tests for cache replication.

The tests serve a number of purposes:

- establishing well understood metrics and limits
- preventing regressions

- reproducing any reported issues in production
- Allowing the design principle of graceful degradation to be achieved. For example, the asynchronous cache replicator uses SoftReferences for queued messages, so that the messages will be reclaimed before before an OutOfMemoryError occurs, thus favouring stability over replication.

6.1.3 Specific Concurrency Testing

JPam also has concurrency testing, which uses 15 concurrent threads hammering a piece of code. The test suites are also run on multi-core or multi-cpu machines so that concurrency is real rather than simulated. Additionally, every concurrency related issue that has ever been anticipated or resulted in a bug report has a unit test which prevents the condition from recurring. There are no reported issues that have not been reproduced in a unit test.

Concurrency unit tests are somewhat difficult to write, and are often overlooked. The team considers these tests a major factor in JPam's quality.

6.1.4 Production tested

JPam has been in production for 18 months.

6.1.5 Fully documented

A core belief held by the project team is that a project needs good documentation to be useful.

In JPam, this is manifested by:

- comprehensive written documentation
- Complete, meaningful JavaDoc for every package, class and public and protected method. Checkstyle rules enforce this level of documentation.
- an up-to-date FAQ

6.1.6 Conservative Commit policy

Projects like Linux maintain their quality through a restricted change process, whereby changes are submitted as patches, then reviewed by the maintainer and included, or modified. JPam follows the same process.

6.1.7 Full public information on the history of every bug

Through the SourceForge project bug tracker, the full history of all bugs are shown, including current status. We take this for granted in an open source project, as this is typically a feature that all open source projects have, but this transparency makes it possible to gauge the quality and riskiness of a library, something not usually possible in commercial products.

6.1.8 Responsiveness to serious bugs

The JPam team is serious about quality. If one user is having a problem, it probably means others are too, or will have. The JPam team use JPam themselves in production. Every effort will be made to provide fixes for serious production problems as soon as possible. These will be committed to trunk. From there an affected user can apply the fix to their own branch.

6.2 Open Source Licensing

6.2.1 Apache 2.0 license

JPam’s original Apache1.1 copyright and licensing was reviewed and approved by the Apache Software Foundation, making JPam suitable for use in Apache projects. JPam 1.0 is released under the updated Apache 2.0 license.

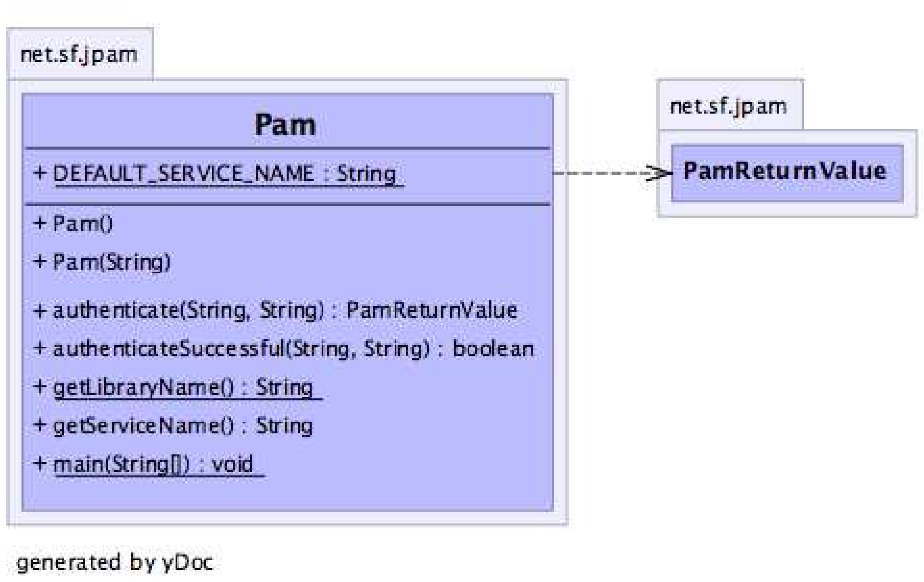
The Apache license is also friendly one, making it safe and easy to include JPam in other open source projects or commercial products.

Chapter 7

Code Samples

- #Using the JPAM API
- #Using the JAAS API

7.1 Using the JPAM API

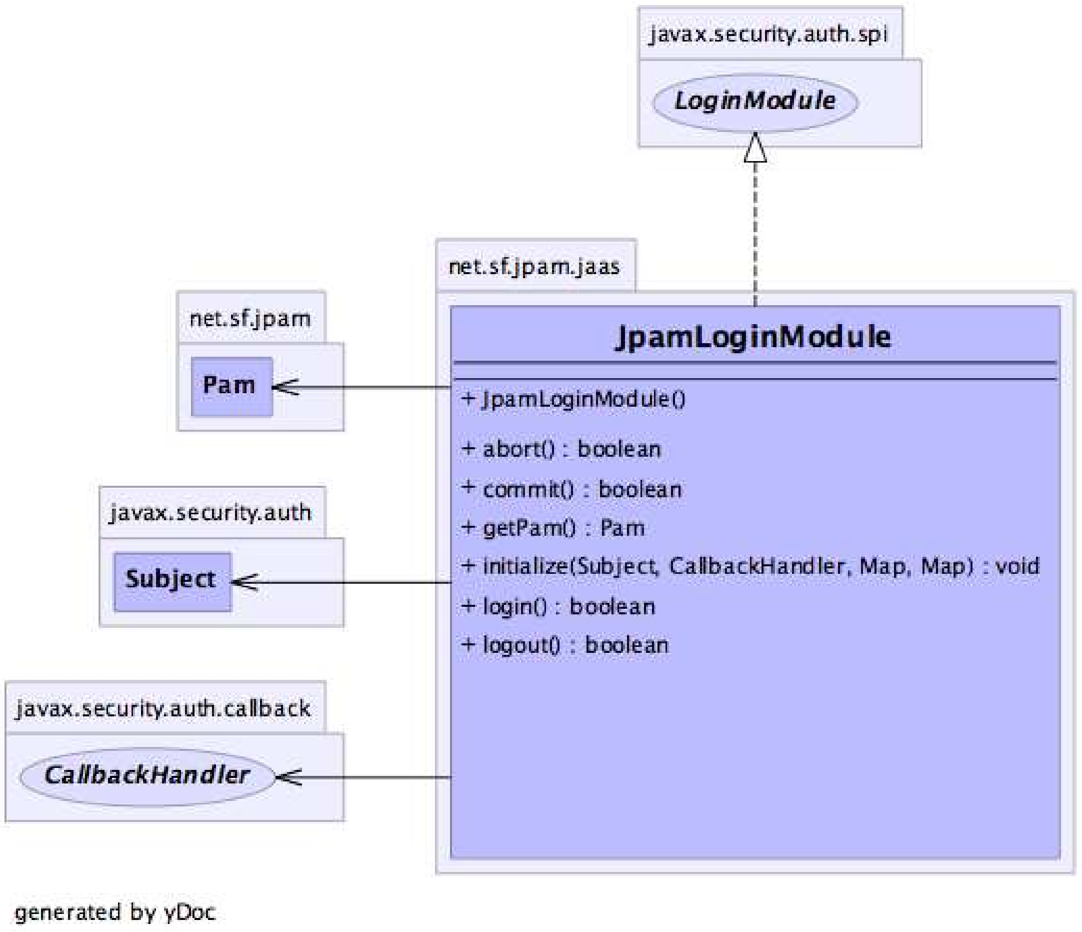

net.sf.jpam
Pam
+ DEFAULT_SERVICE_NAME : String
+ Pam()
+ Pam(String)
+ authenticate(String, String) : PamReturnValue
+ authenticateSuccessful(String, String) : boolean
+ getLibraryName() : String
+ getServiceName() : String
+ main(String[]) : void
net.sf.jpam
PamReturnValue
generated by yDoc

Pam Class Diagram

Attempt to authenticate a username and password

```
String user1Name = "test";
String user1Credentials = "testPassword";
Pam pam = new Pam();
boolean authenticated = pam.authenticateSuccessful(user1Name, user1Credentials));
```

7.2 Using the JAAS API



JPamLoginModule Class Diagram

Shows how to use the JAAS API together with a CallbackHandler.

```
    LoginContext loginContext = new LoginContext("net-sf-jpam", new JpamCallbackHandler());
    loginContext.login();
    loginContext.login();


    /**
     * The application must implement the CallbackHandler.
     * <p/>
     * <p> This application is text-based.  Therefore it displays information
     * to the user using the OutputStreams System.out and System.err,
     * and gathers input from the user using the InputStream, System.in.
     */
    class JpamCallbackHandler implements CallbackHandler {

        /**
     * Invoke an array of Callbacks.
     * <p/>
     * <p/>
     *
     * @param callbacks an array of <code>Callback</code> objects which contain
     *                  the information requested by an underlying security
     *                  service to be retrieved or displayed.
     * @throws java.io.IOException          if an input or output error occurs. <p>
     * @throws UnsupportedCallbackException if the implementation of this
     *                                      method does not support one or more of the Callba
     *                                      specified in the <code>callbacks</code> parameter
     */
    public void handle(Callback[] callbacks)
            throws IOException, UnsupportedCallbackException {

        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof TextOutputCallback) {

                // display the message according to the specified type
                TextOutputCallback toc = (TextOutputCallback) callbacks[i];
                switch (toc.getMessageType()) {
                    case TextOutputCallback.INFORMATION:
                        System.out.println(toc.getMessage());
                        break;
                    case TextOutputCallback.ERROR:
                        System.out.println("ERROR: " + toc.getMessage());
                        break;
                    case TextOutputCallback.WARNING:
                        System.out.println("WARNING: " + toc.getMessage());
                        break;
                    default:
                        throw new IOException("Unsupported message type: "
                                + toc.getMessageType());
                }

            } else if (callbacks[i] instanceof NameCallback) {

                // prompt the user for a username
                NameCallback nc = (NameCallback) callbacks[i];
                nc.setName(user1Name);

            } else if (callbacks[i] instanceof PasswordCallback) {

                // prompt the user for sensitive information
                PasswordCallback pc = (PasswordCallback) callbacks[i];
                pc.setPassword(callbackCredentials.toCharArray());

            } else {
                throw new UnsupportedCallbackException
                        (callbacks[i], "Unrecognized Callback");
            }
        }
    }

}
```

7.3 Browse the JUnit Tests

JPam comes with a comprehensive JUnit test suite, which not only tests the code, but shows you how to use JPam.

A link to browsable unit test source code for the major JPam classes is given per section. The unit tests are also in the src.zip in the JPam tarball.

Chapter 8

Logging And Debugging

8.1 Commons Logging

JPam uses the Apache Commons Logging library for logging.

It acts as a thin bridge between logging statements in the code and logging infrastructure detected in the classpath. It will use in order of preference:

- log4j
- JDK1.4 logging
- and then its own `SimpleLog`

  This enables JPam to use logging infrastructures.

  It does create a dependency on Apache Commons Logging, however many projects share the same dependency.

For normal production use, use the `WARN` level in log4J and the `WARNING` level for JDK1.4 logging.

8.2 libjpam.so Debugging

If the DEBUG logging level is enabled, JPam will instruct libjpam.so to log messages to the console. This is very useful for identifying errors.

8.3 syslogd logging

It can be useful to turn on syslogd for PAM logging. Library problems with PAM modules will then be logged.

Add "auth.notice" to the /var/log/messages line in /etc/syslog.conf.

e.g.

```
*.info;mail.none;authpriv.none;cron.none;auth.notice /var/log/messages
```

Then simply tail /var/log/messages to see PAM logging.

Chapter 9

Frequently Asked Questions

9.1 Does JPam run on JDK1.2?

No, it is not supported.

9.2 JPam does not have a configuration for my OS/architecture. What can I do?

It is easy to make the changes to the build.xml and makefile to support other architectures. Do it and submit a patch! Both Solaris and HP-UX support come from patches.

Chapter 10

Limitations

Jpam presently does not support advanced PAM conversations such as:

- change password on first login
- change password as expired
  These may be added into a future version.

Chapter 11

Building JPam from Source

11.1 Building from source

To build JPam from source:

1. Check the source out from the subversion repository.
2. As root, install the pam-devel-0.77-66.2 package or similar is installed (Linux systems only)
3. Create the following users on your machine:
   - user test password test01
   - user test2 password test02
4. As root, copy src/config/*architecture*/net-sf-jpam* to /etc/pam.d
5. Copy src/config/*architecture*/.java.login.config to your home directory
6. Ensure you have a valid JAVA_HOME and ANT_HOME configured with binaries in your PATH
7. From within the pam directory, type ant

11.2 RPM packaging (Optional)

The source download and the CVS source contain a src/rpm directory containg an RPM spec file created in collaboration with developers at Red Hat. You can use this spec.file to build an rpm installable package for your architecture.

11.3 Building the Site

(These instructions are for project maintainers)

You need the following unix utilities installed:

- latex or tetex
- ghostscript

- pdftk
- aptconvert
- netpbm

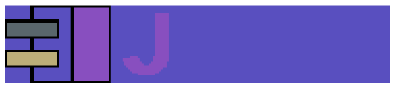
  You also need a yDoc license.

  With all that, build the site as below:

  ```
  mvn site:site
  ```

Chapter 12

About the JPam name and logo



Logo

The JPam logo has a blue rectangle representing a Unix OS (blue is the Solaris colour), Java (the purple colour) and PAM (the small rectangles), which is what JPam is all about.

Index