Secure Tomcat  
  
application environment

Using Symantec Privileged Access Manager



This document is about Symantec Privileged Access Manager and how to secure some aspects of a Tomcat application server. It will show some advanced usages of the A2A client and how PAM can be used to provide secure access to applications in a semi-open environment.

The document covers how to setup Tomcat as a TLS server while still protecting the keystore with the private key without having a keystore password in a configuration file. The other topic covered is how to secure messages send from a client to the Tomcat server and how to avoid hardcoded passwords.

When using Tomcat as an application server with inbound TLS requirement, the Tomcat server uses a public/private key and a certificate to identify itself. This is to prevent someone from impersonating a valid Tomcat server and spoof the real one for potentially gaining access to secret information.

When configuring TLS in Tomcat, the private key is typically stored in a Java keystore and access to the keystore and private keys is controlled by a password. The documentation for Tomcat only reads that the password for the keystore is kept in a configuration file and that all access to the configuration file is restricted by file system access controls.

In some environments, this is sufficient and in others, it is not sufficient. The fear is that even an administrator may get rough and potentially compromise the system.

This document describes a mechanism where the PAM is used to control access to the keystore password and only releases it to a known, authenticated and authorized Tomcat server. It uses the PAM Application to Application (A2A) mechanism for controlling release of the keystore credentials and uses the PAM A2A security controls to verify the caller (Tomcat application server) is recognized, authentic and authorized to retrieve the password for the keystore. Only if the environment is trusted will the password be released, and the Tomcat server can setup itself as a TLS server and respond to inbound calls using the private key from the keystore.

Alternative approaches could be using a fine-grained access control system like PAM Server Control, and others. All of these will probably require new software with additional costs for installing and maintaining potentially complex security access rules in the server. This is not considered in this document.

In the document a sample client and sample server are described. The messages send to the Tomcat server are JSON Web Tokens, which provide message confidentiality and message integrity. The password or key used for message protection is stored in PAM and rotated frequently.

The final example shown in the document is about user access to Tomcat applications. It implements a CredentialHandler method, which is used by Tomcat to verify user credentials (i.e. the password). This is tight into the PAM access page and will permit automated login to a Tomcat application and still permit frequent rotation of the access password.

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date issued | Contributors | Description |
| 1.0 | 2018-10-25 | CR | Initial release |
| 1.1 | 2018-11-18 | CR | Rearranged chapters and sections. Added more security considerations and description of source code. |
| 2.0 | 2018-11-23 | CR | Added JWT message decode and sample applications. |
| 2.1 | 2018-11-28 | CR | Added CredentialHandler for UserDatabaseRealm protection using A2A |
| 2.1.1 | 2018-12-04 | CR | Added how to setup Tomcat Manager to accept calls from any server. Added section about using user password hash for user authentication to Tomcat. |

Table of Contents

[1 Overview 5](#_Toc531699178)

[1.1 Protecting keystore password 5](#_Toc531699179)

[1.2 Protecting client/server messages 6](#_Toc531699180)

[1.3 User access to Tomcat applications 6](#_Toc531699181)

[2 Environment 8](#_Toc531699182)

[2.1 PAM appliance 8](#_Toc531699183)

[2.2 Windows server 8](#_Toc531699184)

[2.3 Tomcat Application Server 8](#_Toc531699185)

[2.4 PAM A2A Client 8](#_Toc531699186)

[2.5 Java JDK 8](#_Toc531699187)

[2.6 Apache Ant 8](#_Toc531699188)

[3 PAM setup 9](#_Toc531699189)

[3.1 Catalina vault 9](#_Toc531699190)

[3.1.1 Vault device 9](#_Toc531699191)

[3.1.2 Password Composition Policy 9](#_Toc531699192)

[3.1.3 Vault application 11](#_Toc531699193)

[3.1.4 Vault account 11](#_Toc531699194)

[3.1.5 Target Group 13](#_Toc531699195)

[3.2 A2A Client 14](#_Toc531699196)

[3.2.1 Device with A2A client 14](#_Toc531699197)

[3.2.2 A2A script 15](#_Toc531699198)

[3.2.3 A2A request group 15](#_Toc531699199)

[3.2.4 A2A mapping 16](#_Toc531699200)

[4 Tomcat Application Server 18](#_Toc531699201)

[4.1 Mandatory configuration 18](#_Toc531699202)

[4.1.1 catalina.properties 18](#_Toc531699203)

[4.1.2 server.xml 18](#_Toc531699204)

[4.1.3 Library files 19](#_Toc531699205)

[4.1.4 secureTomcat.jar 19](#_Toc531699206)

[4.1.5 pam.keystore 19](#_Toc531699207)

[4.1.6 pam.filelist 20](#_Toc531699208)

[4.2 Optional configuration 20](#_Toc531699209)

[4.2.1 Disable automatic deployment of war files 20](#_Toc531699210)

[4.2.2 Disable port 8080 20](#_Toc531699211)

[4.2.3 Disable AJP port 8009 21](#_Toc531699212)

[4.2.4 Force redirect to port 8443 21](#_Toc531699213)

[4.2.5 Log level of secureTomcat 21](#_Toc531699214)

[4.3 Tomcat application 21](#_Toc531699215)

[4.4 Starting Tomcat 21](#_Toc531699216)

[4.4.1 Verification failed - callstack hash 21](#_Toc531699217)

[4.4.2 Verification failed - filelist hash 22](#_Toc531699218)

[4.4.3 Startup OK 22](#_Toc531699219)

[5 secureTomcat 23](#_Toc531699220)

[5.1 Source code 23](#_Toc531699221)

[5.1.1 PAM.java 23](#_Toc531699222)

[5.1.2 Message.java 24](#_Toc531699223)

[5.1.3 PAMCredentialHandler.java 24](#_Toc531699224)

[5.2 Building secureTomcat.jar 25](#_Toc531699225)

[6 Sample application 26](#_Toc531699226)

[6.1 Sample server - echoApp 26](#_Toc531699227)

[6.1.1 A2A script 26](#_Toc531699228)

[6.1.2 A2A mapping 26](#_Toc531699229)

[6.1.3 Sample run 26](#_Toc531699230)

[6.2 Sample client 27](#_Toc531699231)

[6.2.1 A2A script 27](#_Toc531699232)

[6.2.2 A2A mapping 27](#_Toc531699233)

[6.2.3 Sample run 27](#_Toc531699234)

[7 User access to Tomcat applications 28](#_Toc531699235)

[7.1 PAM setup 28](#_Toc531699236)

[7.1.1 PCP for user passwords 28](#_Toc531699237)

[7.1.2 Application for Tomcat users 28](#_Toc531699238)

[7.1.3 Account for Tomcat users 28](#_Toc531699239)

[7.1.4 TCP service for Tomcat application 29](#_Toc531699240)

[7.1.5 Policy for user access to Tomcat application 29](#_Toc531699241)

[7.2 Tomcat setup 30](#_Toc531699242)

[7.2.1 server.xml 30](#_Toc531699243)

[7.2.2 tomcat-user.xml 30](#_Toc531699244)

[7.2.3 context.xml 30](#_Toc531699245)

[7.3 Sample access 31](#_Toc531699246)

[7.4 User access using a password hash 32](#_Toc531699247)

[7.4.1 server.xml 32](#_Toc531699248)

[7.4.2 User’s password hash 32](#_Toc531699249)

[8 Security considerations 33](#_Toc531699250)

[8.1 Password for keystore is limited 33](#_Toc531699251)

[8.2 Callstack verification 33](#_Toc531699252)

[8.3 Filelist verification 33](#_Toc531699253)

[8.4 A2A script integrity 33](#_Toc531699254)

[8.5 Keystore password is not updated automatically 34](#_Toc531699255)

[8.6 Incorrect classes used 34](#_Toc531699256)

[8.7 Man in the Middle 34](#_Toc531699257)

[8.8 File system access 34](#_Toc531699258)

[8.8.1 Webapps protections 34](#_Toc531699259)

[8.9 Keystore password and private key in memory 34](#_Toc531699260)

[8.10 Firewall rules 35](#_Toc531699261)

[9 Troubleshooting 36](#_Toc531699262)

[9.1 Lots of exceptions when starting Tomcat 36](#_Toc531699263)

[9.2 401 - User not authorized 36](#_Toc531699264)

[9.3 402 - A2A Client not started 36](#_Toc531699265)

[9.4 405 - Alias not found 37](#_Toc531699266)

[9.5 409 - Unauthorized script name 37](#_Toc531699267)

[9.6 410 - Unauthorized execution path 37](#_Toc531699268)

[9.7 411 - Unauthorized execution user 37](#_Toc531699269)

[9.8 436 - Incorrect Script Integrity value 37](#_Toc531699270)

[9.9 Script name incorrect 38](#_Toc531699271)

# Overview

This document describes a mechanism where a Tomcat application server uses TLS for inbound connections and keeping access to the keystore secret from the administrator. Also included in this document is an example how secure messages can be send from a client to the Tomcat server without having to store passwords or keys on the Tomcat server.

## Tomcat application environment

A typical environment for an application server is outlined below. It is a Java based environment. The application server, e.g. Tomcat, is running in an OS and the server application is running inside the application server environment. Assuming that the only inbound connection permitted is HTTPS (TLS), the Tomcat server uses a Java Keystore during startup. If the keystore cannot be opened Tomcat will not listen to port 443 and will not accept incoming requests.



## Protecting keystore password

When using TLS connections for connections to a Tomcat server, the Tomcat server must have an X.509 certificate and corresponding private key. When a client (e.g. a browser) connects to the Tomcat server, the TLS hand-shake will authenticate the server. For this to work, the Tomcat server must have access to the private key matching the certificate issued to the Tomcat server. The certificate can be self-signed or signed by a trusted certificate authority.

Typically, the private key is stored in a keystore and the password or key needed to unlock the keystore is stored in a Tomcat configuration file. When looking at the Tomcat documentation, it is always recommended to limit access to the file system of the configuration files and keystore. However, if someone does have access, it is easy to create a clone of the Tomcat server and impersonate the Tomcat server for a real one. The client (e.g. browser) will not see a difference as the certificate and private key is available and will not easily detect that it is a clone or faked Tomcat server.

One way of preventing such malicious servers to be incorrectly authenticated, is to keep access to the keystore with the private key strictly controlled and only made possible to known and authorized Tomcat servers. The mechanism described here uses the PAM Application to Application (A2A) mechanism, where the password to unlock the keystore is kept in PAM and only released to a Tomcat server if the environment is authenticated and authorized.

The A2A mechanism can be setup to validate that the server running the A2A client and the script requesting credentials (the keystore password), are registered and only release the credentials if various checks are successfully passed. If any of these checks fail, the credentials are not released to the Tomcat application server and it will not have access to the keystore and will not be able to act as a trusted TLS server.



The mechanism used is to override the Java method getProperty and how this is called when defining the keystore password needed to setup the TLS server side. The method override is a standard way for Java to extend the behaviour of a parent class. In this case it is defined in the server.xml file in Tomcat when Tomcat uses the keystore password for accessing the keystore.

Before releasing credentials (the keystore password) to Tomcat various checks are done.

1. A2A authentication and authorization mappings.
2. Callstack verification, verify that the callstack matches the expected callstack.
3. File list check, verify a hash of files from a filelist.



## Protecting client/server messages

Protecting messages from clients to servers can be done in many ways, shapes, and forms. Main requirement is to provide data confidentiality and message integrity. Non-repudiation is not a prime concern.

The approach used here is JSON Web Tokens (short: jwt) with encrypted and integrity protected payloads. Encryption used is AES 128. Integrity protection is using HMAC with SHA-256. It was decided against using public keys and certificates. Main reasons are the necessity to use certificates and certification paths, including the storage and use of the Tomcat servers private key (for decrypting the payload). This is by itself not difficult to implement and use, but there will be an overhead to verify the client certificate whenever a message is received, and that public key operations typically are more CPU intensive than symmetric key operation.



The client (sender) will fetch a password from PAM using A2A. This passphrase is converted into a symmetric key and used as a key encryption key for the message. A random data encryption key is generated and used to create a HMAC and to encrypt the message. The random data encryption key is encrypted using key encryption key fetched from PAM via an A2A call. Everything is packaged into a JWT message and send to the Tomcat web application.

On the Tomcat server the process is reversed.

The server (receiver) will fetch a password from PAM using A2A. This passphrase is converted into a symmetric key and used as a key encryption key for the message. With that in hand, the data encryption key is decrypted, then the message and finally the integrity of the message is validated, and the clear text payload is available.

Key elements in this setup are:

1. The passphrase for the key encryption key is only released to known and authorized scripts via the A2A client.
2. The filelist integrity verification when releasing the keystore password should include all the files for the web application receiving JWT messages.
3. The Tomcat server automatic unpacking and deployment of war files should be turned off.

The rest of this chapter will discuss and describe the additions to the keystore password protection needed to accommodate receiving and decoding the JWT messages. There is a sample client which can be used to generate and send JWT messages, but this is only discussed briefly.

## User access to Tomcat applications

When users connect to PAM, they can authenticate using local username/password hardcoded in the tomcat-users.xml file. There are mechanisms available to not store the user password in clear text but to keep a hashed value of the password in the configuration file. For many administrators this is a perfect solution.

For scenarios where connection to the Web application is established from PAM as a web application, the password can be injected directly at login. This can either be the HTML form or HTTP access. Regardless of which protocol is used, PAM is the master of the password and can automatically generate new random passwords and use them when needed. When a user (via PAM) tries to login to an application running in the Tomcat application server, the secureTomcat module will reach back to PAM and fetch the current password. If this is correct, the user session is authenticated.



Alternatives exists for user authentication to Tomcat applications. These includes, but are not limited to, a mechanism where password hashes are created in PAM and stored directly in the tomcat-users.xml file, user authentication using LDAP/AD or a JAAS module for authenticating users.

# Environment

This chapter describes the environment used for the keystore access used in the decimation. The version of PAM, the OS and other components used here can be different. If they differ from what is described here, the GUI layout and operation may differ.

The environment used is:

* PAM appliance  
  PAM version 3.2.2 in a 2-node cluster setup. The cluster operation is not important here.  
  The license applied to the PAM instance must include Credential Management and A2A type devices. An Access type device is not required to demonstrate the keystore access mechanism.
* Windows server 2008R2 for Tomcat and A2A client
* Tomcat application server version 9.0.12
* PAM A2A Client version 4.12.1
* Java SDK version 1.8 update 181 with keytool
* Apache ant version 1.9.4

## PAM appliance

The PAM appliance used here is version 3.2.2 running in VMware. It is not critical which version nor platform for the appliance is used.

## Windows server

The Windows server used to host the Tomcat application server is a Windows 2008R2 server. It could be any version of Windows and even a UNIX server. The Tomcat application server and PAM A2A client is needed on this server.

## Tomcat Application Server

The approach outlined here has been tested using Tomcat version 7.0.79, 8.5.34 and 9.0.12. There are differences from version 7 to 8 and 9 and the Java code must be compiled using a different classpath.

The application server requires a Java run-time to be installed on the server. Compiling the Java source code to a class file will require a JDK to be available and Java JDK 1.8 is installed and used.

Define the environment variable CATALINA\_HOME=C:\apache-tomcat-9.0.12

## PAM A2A Client

The PAM client used here is version 4.12.1.

Two environment variables are needed for the PAM A2A Client.

CSPM\_CLIENT\_BIT\_TYPE=64

CSPM\_CLIENT\_HOME=c:\cspm\cloakware

Update the Path environment variable to include %CSPM\_CLIENT\_HOME%\cspmclient\bin;%CSPM\_CLIENT\_HOME%\cspmclient\lib

## Java JDK

The JDK used here is version 1.8 update 181. Different versions compatible with the Tomcat server can be used.

Define the environment variable JAVA\_HOME=C:\Program Files\Java\jdk1.8.0\_181

Update the Path environment variable to include %JAVA\_HOME%\bin

## Apache Ant

The ant tool version 1.9.4 is used to build the necessary jar and class files from the sources.

# PAM setup

This chapter details how PAM is configured to accommodate secure access to a keystore.

Three “passwords” are stored in a PAM vault and fetched via A2A calls.

1. Keystore password
2. Callstack hash
3. Filelist hash

The keystore password is a real password used by Tomcat to gain access to the keystore.

The callstack hash is a hash value of the expected callstack.

The filelist hash is a hash value of one or more files which integrity is to be validated.

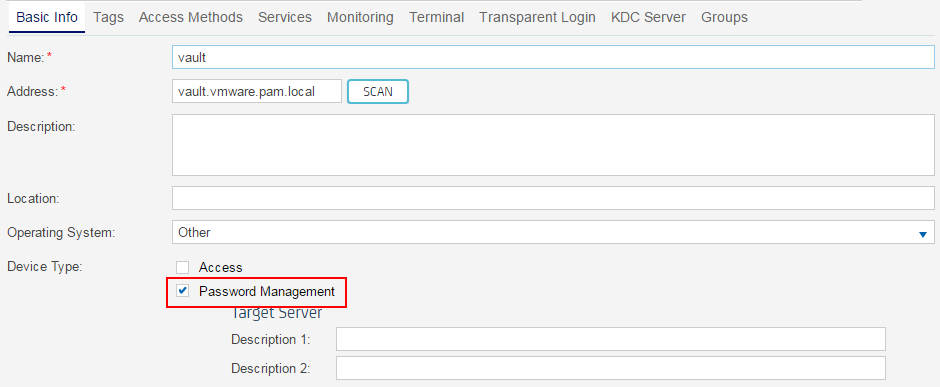
## Catalina vault

The Catalina vault is used to store the password for accessing the keystore, the callstack hash and the filelist hash. The account type used is Generic and as always with accounts in PAM, it requires a triplet to be defined. The device, an application and the account.

### Vault device

Out-of-the-box PAM does not support rotating of passwords in a keystore and PAM is used only to store the password needed to generate the password for the keystore. Future releases of PAM may behave differently, and a different device may be used.

A dummy device as anchor for passwords is created. The device must have Password Management device type enabled. The address of the device is a dummy device and does not have to be a real server.



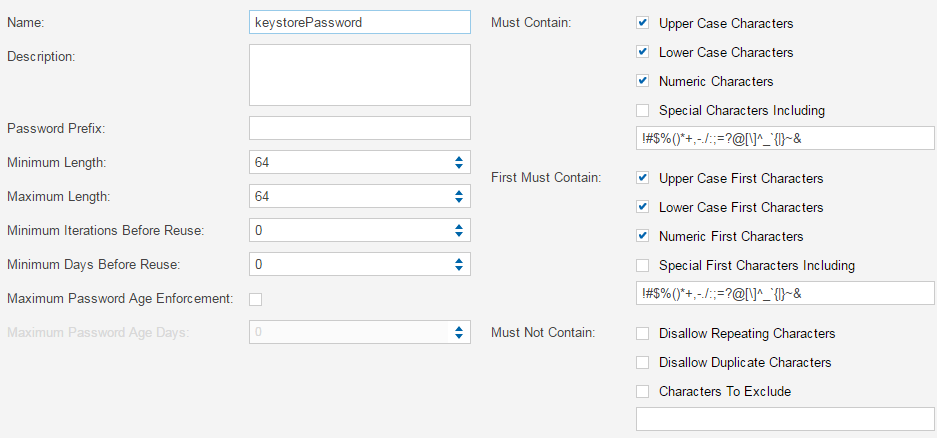
### Password Composition Policy

Two Password Composition Policies are needed. One for the keystore password and one for hash values.

#### PCP for keystore password

Even though the password for the keystore is kept in memory only, it should still strong with sufficient length. In PAM a password composition policy is created for generating passwords used for protecting the keystore.

The minimum/maximum length is set to 64 characters without any special characters.

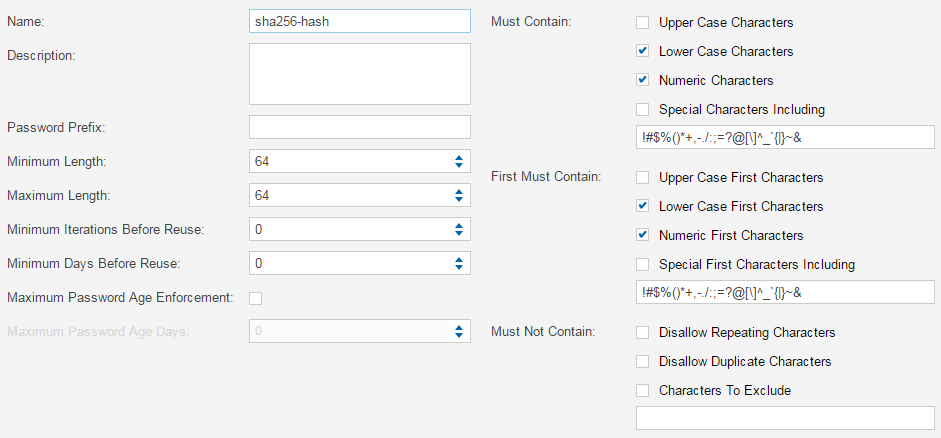


The keystore PCP is for a real password and should include upper case, lower case and numeric characters. As the example shown in this document uses a script to create the keystore, special characters are omitted as these may need to be escaped when used in a script.

The maximum length can be at most 254 characters.

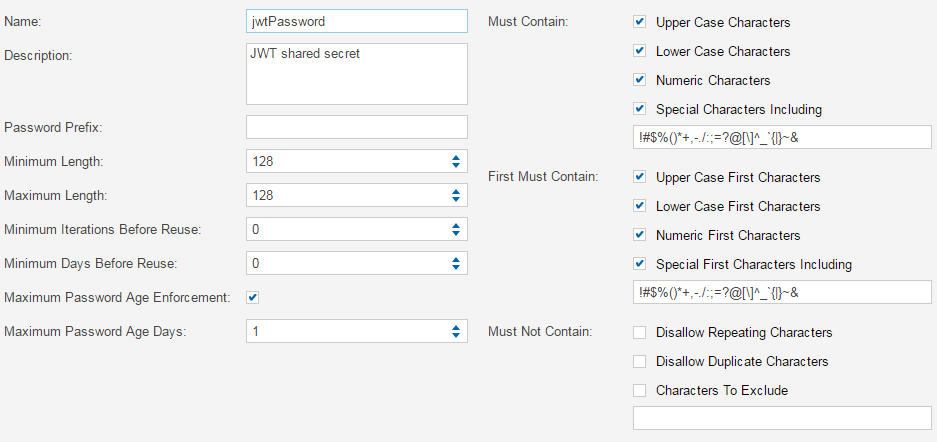
#### PCP for hashes

The hash PCP is for storing a sha256 hash. The hashes used in this example all uses only lower case and numeric characters a dedicated PCP for hashes is needed. The hash used is exactly 64 characters (256 bits). If the hash method used is changed, the minimum/maximum length must be changed too.



#### PCP for JWT messages

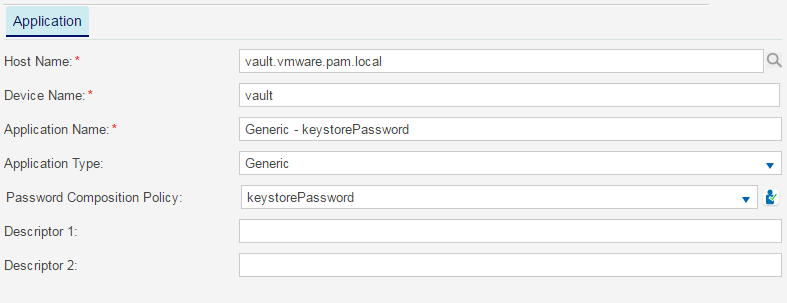
The PCP used for keys for the JWT messages are long, complex, and most important, they have a defined maximum age. The reason for the age limit is that the key should be rotated frequently.



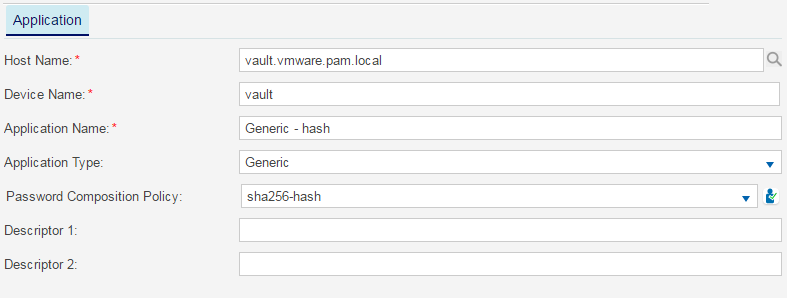
### Vault application

Two target applications are created. One for the keystore password and one for hashes. The two applications are both of type Generic and uses different Password Composition Policy.

#### Keystore application

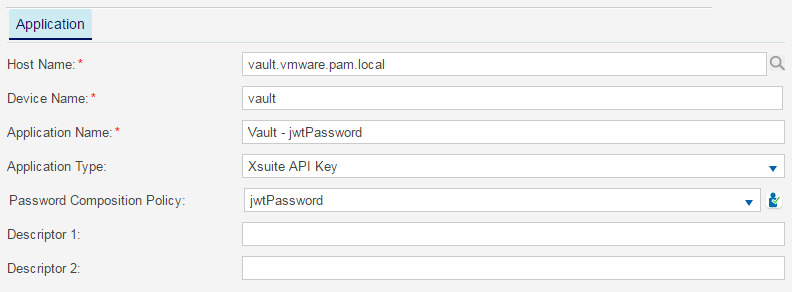


#### Hashes application



#### JWT application

The passwords for the JWT messages are not changed on an end-point, but only internally in PAM. This calls for the application type Xsuit Api Key, which can be setup to rotate the account password automatically. The PCP for the application is the one just created with the password age limit set to 1 day.

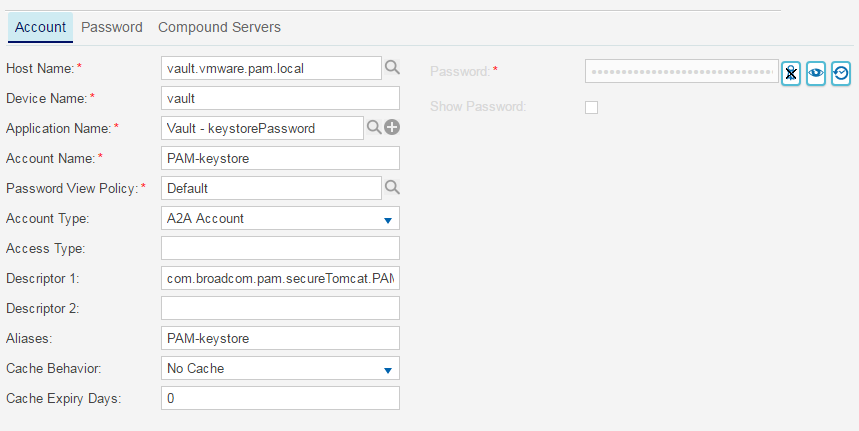


### Vault account

Three accounts are defined in PAM. One for the keystore password and two for storing the expected callstack hash and filelist hash.

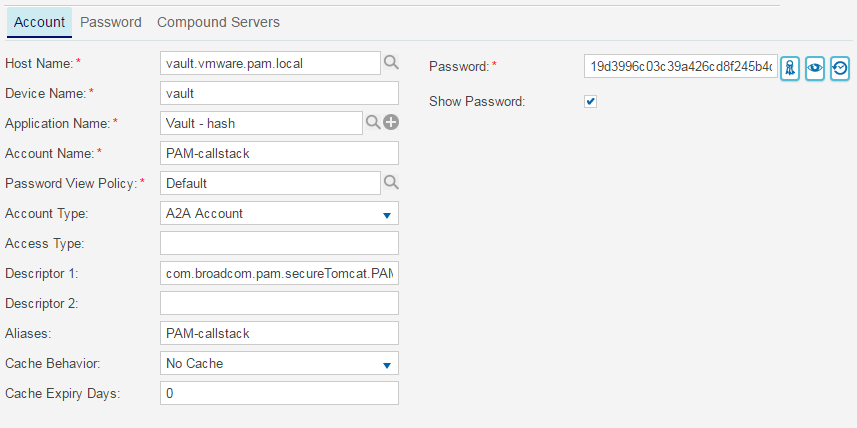
#### Keystore account

A target account is created for keeping the password used to calculate the keystore password. The account type is A2A with same alias as account name. The account name and alias can be different. The cache behavior is set to “No Cache”. The cache setting is to avoid the necessity of handling stale cache in the Catalina application server. A new random password is created for this account. The descriptor1 field is set to com.broadcom.pam.secureTomcat.PAM and is used to filter accounts in a target group.



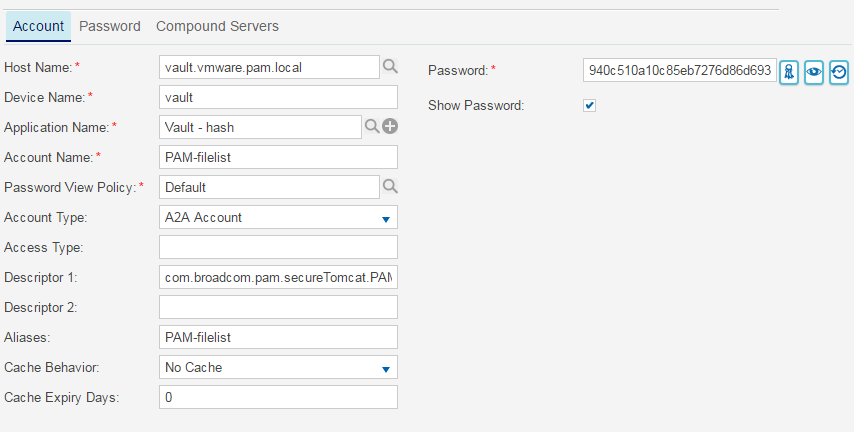
#### Callstack account

A target account is created for storing the expected hash of the callstack. The account type is A2A with same alias as account name. The account name and alias can be different. The cache behaviour is set to “No Cache”. The cache setting is to avoid the necessity of handling stale cache in the Catalina application server. The password is set to the hash of the callstack. If it is set to an incorrect value, the Catalina log will show the expected hash value. The hash can be copied from there during setup of the Tomcat server. The descriptor1 field is set to com.broadcom.pam.secureTomcat.PAM and is used to filter accounts in a target group.



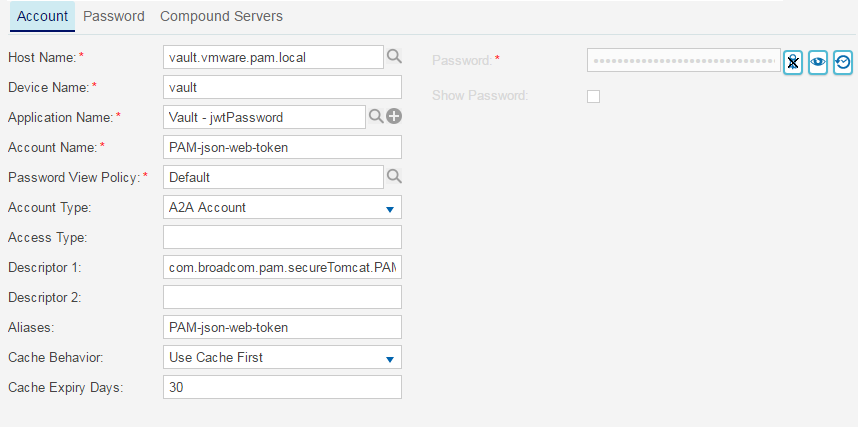
#### Filelist account

A target account is created for storing the expected hash of a filelist. The account type is A2A with same alias as account name. The account name and alias can be different. The cache behaviour is set to “No Cache”. The cache setting is to avoid the necessity of handling stale cache in the Catalina application server. The password is set to the hash of the callstack. If it is set to an incorrect value, the Catalina log will show the expected hash value. The hash can be copied from there during setup of the Tomcat server. The descriptor1 field is set to com.broadcom.pam.secureTomcat.PAM and is used to filter accounts in a target group.



#### JWT account

The account for JWT is the password used to generate the keys for the JWT messages. The password is created randomly and should be changed frequently. This can be done using scheduled jobs or by using the password age. When decoding JWT messages it is important not always to contact PAM and to use the local cache. The code will handle stale cache; thus, the cache behaviour is set to Use Cache First. The descriptor1 field is set to com.broadcom.pam.secureTomcat.PAM and is used to filter accounts in a target group.

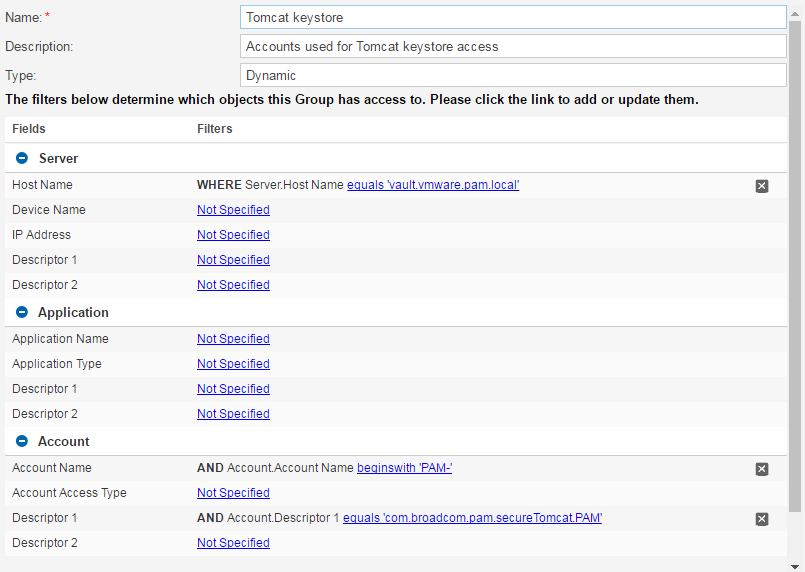


### Target Group

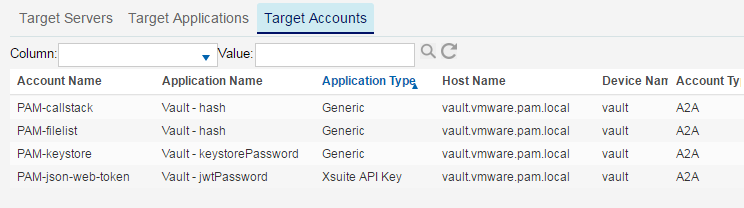
A target group is created. It is used later to setup the A2A mappings to cover all A2A accounts used

The filters are set to use the hostname, account name and account descriptor1. Different filters can be used to list only the accounts used to hold the keystore password, the callstack hash and filelist hash.

Especially if multiple Tomcat servers are used in a cluster setup, the use of a target group is very useful.



Verify that the list of accounts filtered in the target group is as expected.

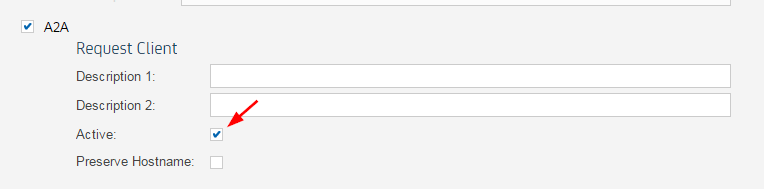


## A2A Client

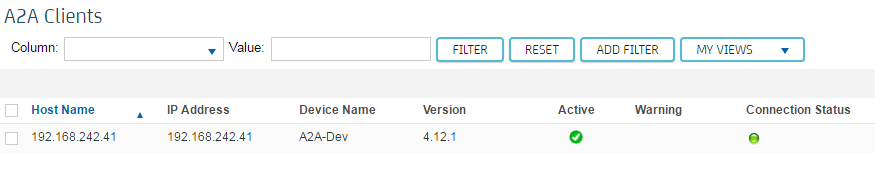
A PAM A2A Client is installed on the same server where the Tomcat application server is used. When the A2A client is started first time, it is registered in PAM and marked as inactive. An administrator must identify the device and change the A2A client from inactive to active.

### Device with A2A client

If the device was not previously created, a new device is added to PAM. It is marked as an A2A device type without being active.



Navigate to “Credentials > Manage A2A > Clients”. The A2A client should be visible and marked as active.

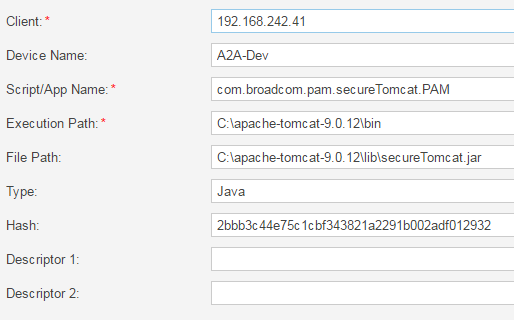


### A2A script

An A2A script describes the program which is using A2A to fetch managed credentials. The script in a Java context is the class calling PAM A2A client.

The script belongs to an A2A Client. The script name is the calling class, here the class name com.broadcom.pam.secureTomcat.PAM is used. Execution path is the bin directory of the Tomcat server and the file path where the class (script) is found. Here it is the Tomcat server lib directory and the jar file secureTomcat.jar. The script type is Java. The script hash is calculated when first used or explicitly requested through the GUI or CLI command.

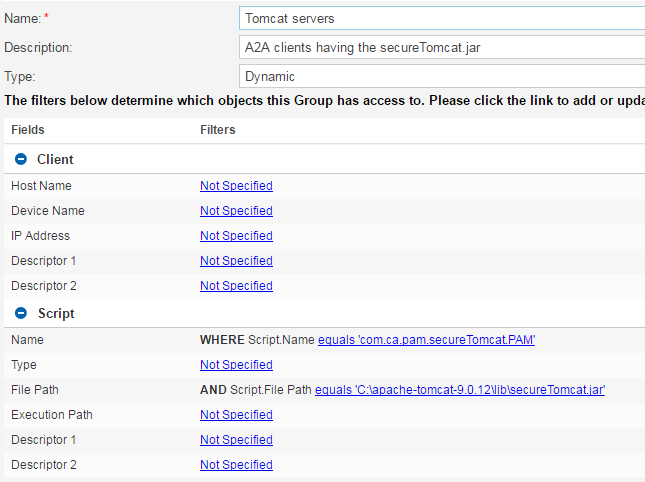
If a different version of Tomcat or different platform is used, the execution and file paths will be different.



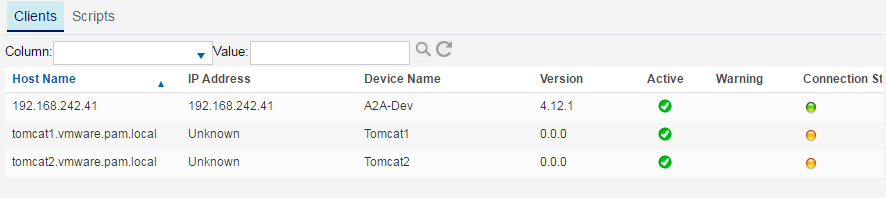
### A2A request group

An A2A request group is created. It filters the A2A clients (request servers) having the script for the Java method getProperty override defined. Here the filter is on the script name and file path.

Using a request group for A2A clients is useful when using multiple Tomcat servers.



Verify that only the defined Tomcat servers (A2A clients) are included in the list. In this example three Tomcat servers are defined.



### A2A mapping

The A2A mapping is an authorization for a script running on a specific server to fetch a specific credential.

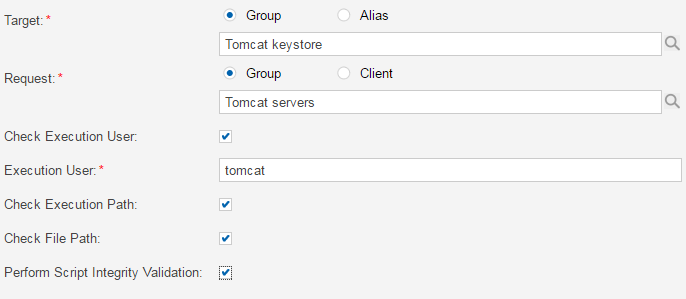
The A2A mapping can be defined for a single alias or use a target group. The target group defined earlier is used to cover all three accounts in one policy. It can also use a single request client or a request group. If multiple Tomcat servers are used, using a request group is the better approach.

Other security check options can be enabled if needed. For production usage, these should all be set to verify that only the authorized and authenticated script will receive the credentials.

#### Multiple Tomcat servers

An A2A mapping using a request group is useful when having multiple Tomcat servers in a cluster or fail-over mechanism.

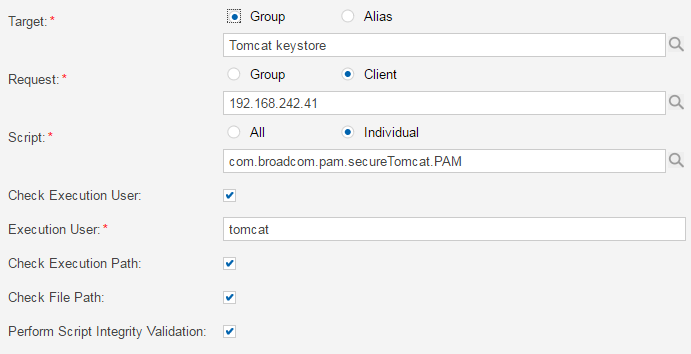
There is an unexpected behaviour with PAM 3.2.2.121 and using request groups in A2A mappings. The mapping with a request group seems to require that at least one individual A2A client definition is created. When one A2A mapping for a target group to a single A2A Client is defined, then will the request group mapping works as expected.



#### Single Tomcat server

If only a single Tomcat server is used it is not necessary to define a request group and the mapping can be created for the single client only.

In such a setup the mapping must also define the individual script permitted to retrieve the credentials.



# Tomcat Application Server

The Tomcat application server used here is version 9.0.12 for Windows and is installed in C:\apache-tomcat-9.0.12. The approach and method described in this document has been tested on Tomcat7 and Tomcat 8.

This chapter includes several sections. The first section is about how-to setup and configure Tomcat to meet the requirements for the secure access to the keystore password. The second section is describing additional configuration options to secure Tomcat. These are not required but should be considered for a production setup.

## Mandatory configuration

This section describes the required configuration of Tomcat for accessing the password for the keystore are the described in this section.

### catalina.properties

Edit the file %CATALINA\_HOME%\conf\catalina.properties.

Add these lines at the end of the file.

# PAM Access Tomcat Keystore

org.apache.tomcat.util.digester.PROPERTY\_SOURCE=com.broadcom.pam.secureTomcat.PAM

pam.keystore.alias=PAM-keystore

pam.callstack.alias=PAM-callstack

pam.filelist.alias=PAM-filelist

pam.filelist.name=c:/apache-tomcat-9.0.12/conf/pam.filelist

pam.jwt.alias=PAM-json-web-token

The org.apache.tomcat... is a reference to the method used to override the default Tomcat getProperty method. The method is implemented in the library file secureTomcat.jar.

The pam.keystore.alias is the alias for the PAM account holding the password for the keystore.

The pam.callstack.alias is the alias for the PAM account holding the callstack hash.

The pam.filelist.alias is the alias for the PAM account holding the filelist hash.

The pam.filelist.name is the filename where the filelist is found. If using “\” they must be written as “\\”.

The pam.jwt.alias is the alias for the PAM account holding the JWT password.

### server.xml

Edit the file %CATALINA\_HOME%\conf\server.xml.

Add a new connector.

<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->

<Connector

protocol="org.apache.coyote.http11.Http11NioProtocol"

port="8443"

maxThreads="200"

scheme="https"

secure="true"

SSLEnabled="true"

keystoreFile="${catalina.home}/conf/pam.keystore"

keystorePass="${pam.keystore.alias}"

clientAuth="false"

sslProtocol="TLS"/>

When Tomcat tries to setup itself with a TLS server, it needs to gain access to the keystore. To do so, it reads the property ${pam.keystore.alias} from catalina.properties. The Tomcat getProperty method is overwritten and will talk back to PAM via the A2A Client to fetch the password returned to Tomcat to unlock the keystore.

### Library files

Several jar files must be deployed in the Tomcat lib directory.

The getProperty override method and the PAM A2A java libraries are to be copied into the %CATALINA\_HOME%\lib directory.

Copy the A2A Client library files found in %CSPM\_CLIENT\_HOME%\lib to %CATALINA\_HOME%\lib:

* cspmclient.jar
* cwjcafips.jar
* cpaspiffadaptor64.dll
* cspminterface64.dll
* cwjcafips.dll

The A2A Client library files are A2A Client version specific. The filenames mentioned here are for A2A Client version 4.12.1. If a different A2A Client version is used, the library files for that version of A2A Client must be used.

The secureTomcat library is:

* secureTomcat.jar

### secureTomcat.jar

The hearth of this exercise is the Java method override where Tomcat calls the method getProperty to fetch the keystore password, and this overwritten method will fetch the details from PAM via an A2A call and calculate the password for the keystore.

Copy the file secureTomcat.jar to the %CATALINA\_HOME%\lib directory.

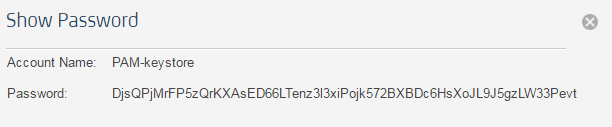
### pam.keystore

The keystore file is either created and uses a self-signed certificate or a keypair is generated and the public keys is signed by a certification authority. Some PKI environments will generate the key pair centrally and distribute the private key and certificate. It should be places in %CATALINA\_HOME%\conf or if placed somewhere else, change the server.xml to reflect the correct location and filename.

The approach followed here is to generate a new RSA keypair and create a self-signed certificate for the Tomcat server.

**First step** is to get the keystore password from PAM.

View the password generated by PAM for the account PAM-keystore.



**Second step** is to create a keystore from scratch. Run the following commands from a command prompt (one line each).

set KEYSTORE=pam.keystore

set PASSWORD=DjsQPjMrFP5zQrKXAsED66LTenz3l3xiPojk572BXBDc6HsXoJL9J5gzLW33Pevt

keytool -genkey -alias pam -keyalg RSA -keysize 2048 -dname "CN=keystore, OU=PAM demo, O=CA, L=Zurich, ST=Zurich, C=CH" -keypass %PASSWORD% -storepass %PASSWORD% -keystore %KEYSTORE% -validity 3650

keytool -importkeystore -srckeystore %KEYSTORE% -srcstorepass %PASSWORD% -destkeystore %KEYSTORE% -deststorepass %PASSWORD% -deststoretype pkcs12

keytool -exportcert -alias pam -keystore %KEYSTORE% -keypass %PASSWORD% -storepass %PASSWORD% -rfc -file secureTomcat.pem

Different approaches can be used to create the keystore and to set the password.

**Third step** is to copy the generated keystore file pam.keystore to %CATALINA\_HOME%\conf.

**Forth step** is to give the self-signed certificate file secureTomcat.pem to the clients connecting to Tomcat. Such clients should verify that the TLS server matches the expected certificate.

### pam.filelist

A filelist file %CATALINA\_HOME%\conf\pam.filelist is created and contains a list of fully qualified filenames. A combined SHA-256 value is calculated of the content of all files in the filelist. There is one condition for the filename in the filelist. The filename of the filelist itself must be included in the list. If it is not included in the filelist, the keystore password is not released.

The expected hash value is retrieved from PAM using A2A call and if the calculated hash matches the retrieved and expected hash, attempt to retrieve the password for the keystore is done.

It is not required to place the files in any particular order.

Example filelist

**C:\apache-tomcat-9.0.12\conf\pam.filelist**

C:\apache-tomcat-9.0.12\bin\tomcat9w.exe

C:\apache-tomcat-9.0.12\lib\annotations-api.jar

C:\apache-tomcat-9.0.12\lib\catalina-ant.jar

C:\apache-tomcat-9.0.12\bin\bootstrap.jar

C:\apache-tomcat-9.0.12\bin\commons-daemon.jar

C:\apache-tomcat-9.0.12\lib\catalina-ha.jar

C:\apache-tomcat-9.0.12\bin\tcnative-1.dll

C:\apache-tomcat-9.0.12\bin\tomcat-juli.jar

C:\apache-tomcat-9.0.12\bin\tomcat9.exe

C:\apache-tomcat-9.0.12\lib\catalina-tribes.jar

C:\apache-tomcat-9.0.12\lib\catalina.jar

C:\apache-tomcat-9.0.12\lib\servlet-api.jar

C:\apache-tomcat-9.0.12\webapps\sampleApp\WEB-  
 INF\classes\com\broadcom\pam\secureTomcat\sampleApp\echoApp.class

C:\apache-tomcat-9.0.12\conf\catalina.properties

C:\apache-tomcat-9.0.12\conf\server.xml

C:\apache-tomcat-9.0.12\conf\web.xml

C:\apache-tomcat-9.0.12\bin\catalina.bat

## Optional configuration

After TLS on port 8443 has been enabled, other ports can be disabled and/or redirected to port 8443. Applications not needed should be removed from the %CATALINA\_HOME%\webapps directory. If filelist validation is enabled, the Tomcat configuration files must should be included in the filelist.

The sections below may not be a complete list of mitigations to be done to lock down a Tomcat server deployment.

### Disable automatic deployment of war files

Depending on your Tomcat setup and usage, it may be desirable not to automatically update deployed war files. One reason can be that you include the war-file and expanded war-file in the filelist integrity verification. This is verified when Tomcat starts, but not if a new war-file is dropped into the webapps directory and automatically deployed. That will allow a malicious program to be deployed. The automatic deployment of updated war-files can be prevented by editing the file %CATALINA\_HOME%\conf\server.xml. Locate the Engine section, e.g. “<Engine name="Catalina" defaultHost="localhost">” and change both the unpackWARs and autoDeploy option to false.

<Host name="localhost" appBase="webapps"

unpackWARs="**false**" autoDeploy="**false**">

### Disable port 8080

If this Tomcat server shall only accept inbound TLS connections on port 8443, disable the connector for port 8080 (plain HTTP). Edit the file %CATALINA\_HOME%\conf\server.xml and comment out the connector for port 8080.

**<!--**

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443" />

**-->**

### Disable AJP port 8009

If this Tomcat server is not talking to other Apache or Tomcat servers, consider disabling port 8009. Edit the file %CATALINA\_HOME%\conf\server.xml and comment out the connector for port 8009.

**<!--**

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

**-->**

### Force redirect to port 8443

It is possible to automatically redirect unsecure connection to a protected context. This may not be needed if Tomcat only listen on port 8443. If port 8080 is already disabled, this configuration is not needed. Connection attempts at 8080 will be rejected.

Edit the file %CATALINA\_HOME%\conf\web.xml and add this section before the final </web-app>.

<security-constraint>

<web-resource-collection>

<web-resource-name>Protected Context</web-resource-name>

<url-pattern>/\*</url-pattern>

</web-resource-collection>

<user-data-constraint>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>

### Log level of secureTomcat

The secureTomcat and sample programs uses the java.util.logger class to write entries to the catalina.log. It is possible to change the log level and provide more details about the operation of the secureTomcat methods.

Edit the file %CATALINA\_HOME%\conf\logging.properties and add these lines to the end of the file.

# To see debug messages for com.broadcom.pam.secureTomcat handling,

# uncomment the following lines:

#com.broadcom.pam.secureTomcat.level= FINE

#com.broadcom.pam.secureTomcat.handlers= java.util.logging.ConsoleHandler

## Tomcat application

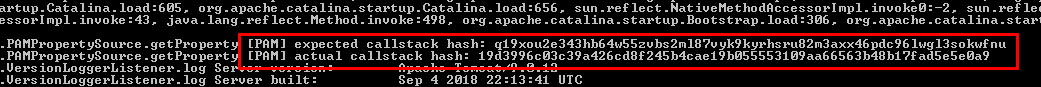
The sample application echoApp (see chapter 6.1 - “Sample server - echoApp”) should be deployed to the Tomcat server and the war-file unpacked manually, if automatic unpacking is set to false.

## Starting Tomcat

In the %CATALINA\_HOME%\bin directory, run the startup command. This will start a new command Window with log information in the console window. Tomcat can also be started as a service or daemon process. This is not covered here.

### Verification failed - callstack hash

Initially when using the overridden getProperty method, the callstack defined in PAM may be incorrect or unknown. Start Tomcat and observe the catalina.log file. It will show the error and the actual hash.

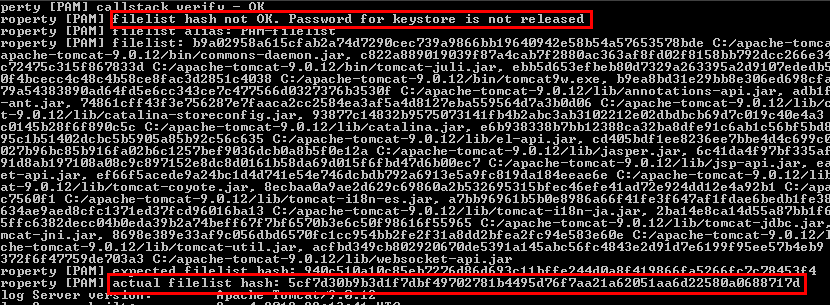


Copy the actual hash from the catalina.log and use it as the password for the account PAM-callstack.

**Note**: Only update the callstack hash if it is known that Tomcat is updated, or a new version is used.

### Verification failed - filelist hash

Initially during setup of Tomcat with the overridden getProperty method, the filelist hash may be incorrect. This is expected, and the catalina.log will show an error and the actual hash created.

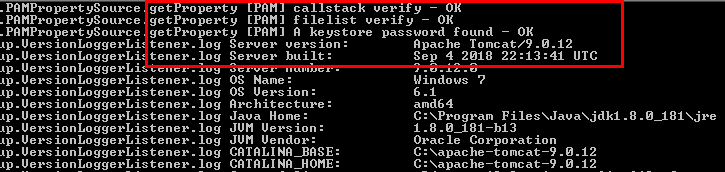


Copy the actual hash from the log and update the password for the account PAM-filelist.

**Note**: Only update the filelist hash if it is known that files have changed, or files are added/removed from the filelist.

### Startup OK

If everything goes as expected, Tomcat is started and will respond to port 8443 using the TLS server private key and certificate found in the keystore.



# secureTomcat

The heart in the method described is the Java package com.broadcom.pam.secureTomcat and the class PAM. This chapter describes some of the key elements of the source code, how to fine tune it and how to build the secureTomcat.jar file.

## Source code

The zip file with secureTomcat.jar should also include the source code and the ant build files.

...\accessKeystore

\build

build.xml

build.properties

\docs

disclaimer.txt

PAM - Tomcat Keystore.pdf (this document)

\lib

secureTomcat.jar

\src

\com\broadcom\pam\secureTomcat

PAM.java

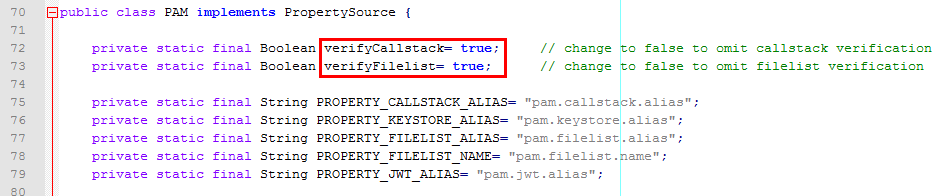
Message.java

PAMCredentialHandler.java

### PAM.java

Most important is how the class integrate with the PAM A2A Client. In PAM the script (this module) is registered and is recognized when it tries to retrieve credentials. The A2A mapping can verify the integrity of the script and it is necessary to have all the code used to verify the callstack and filelist as well as retrieving the keystore password in a single class file.

Callstack and filelist checking can be disabled by changing the variables verifyCallstack and/or verifyFilelist to false. If the property names in the catalina.properties file changes, change the constants accordingly.



The method getProperty is called by Tomcat when it tries to retrieve a keystore password.

It will verify that a hash of the callstack (i.e. how Tomcat came down to calling this method) is what is expected. The expected hash is stored in PAM and retrieved via an A2A call back to PAM. If the expected hash does not match the calculated hash the keystore password is not released.

After verifying the callstack, integrity of a filelist is done. The filelist integrity checking is a filelist having a list of filenames (path + filename) and a hash value is calculated for each file and a final hash over all hashes is calculated. It is verified that the filelist itself is a member of the filelist and that the calculated filelist hash matches the expected filelist hash. The expected filelist hash is stored in PAM and retrieved via an A2A call. If the filelist filename is not part of the filelist or expected hash does not match the calculated hash the keystore password is not released.

Finally, when both checks are passed, the keystore password is retrieved from PAM and returned to Tomcat.

Methods included in the PAM class.

* public PAM()  
  This is the constructor for the class. When called, it will verify that the class instantiating the PAM class is known and originate from a known jar file. If not, releasing passwords is not possible with this instance of the PAM class.
* public String getProperty( String arg0 )  
  This method is already discussed.
* public String getPassword( String alias )  
  This method calls the getPassword with parameter bpc=false
* public String getPassword( String alias, Boolean bpc )  
  This method is where the PAM A2A Client is called. The A2A Client is a service/daemon process running on the server where the A2A client is installed. The bypass cache (the A2A client cache) is can be either true or false. If it is set to false, the calling program or method must counter for stale cache information. Such error handling is not available when Tomcat retrieves a keystore password and when used to fetch the callstack and filelist hashes and the keystore password, the use of cache for these accounts is disabled in PAM. The same method (i.e. getPassword) can be used by other applications in Tomcat and these may have error handling for stale cache incorporated.
* private String sha256String( String source )  
  This calculate a sha256 value of a string. The hash is returned as a hex string.
* private String sha256File( String filename )   
  This calculate a sha256 value of a files content. The hash is returned as a hex string.
* private String convertToHex( byte[] bytes )  
  This converts a byte[]to a hex string.
* private Boolean checkCallstack(String expectedHash)  
  Calls checkCallstack with strict=true.
* private Boolean checkCallstack(String expectedHash,Boolean strict)  
  The is retrieves the callstack, calculate the actual hash and compare it to the expected hash. Returns false if there is a difference and strict=true. Returns true if there is no difference or strict=false.
* private Boolean checkFilelist( String filename, String expectedHash )  
  Calls checkFilelist with strict=true
* private Boolean checkFilelist(String filename,String expectedHash,Boolean strict )  
  This method reads the list of filenames from the file and calculates a hash of every single file listed in the filelist. A long list of hash + filename is created, and a combined hash value is calculated. Returns false if there is a difference and strict=true. Returns true if there is no difference or strict=false.

### Message.java

This source implements the message decoding of JWT messages received from a client. The method is used in the sampleServer/echoApp application.

Method included in the com.broadcom.pam.secureTomcat.Message class is:

* public String jwtDecode(String jweString)  
  This method is used to decode a JWT message from a client and return the plain text userData payload.

Not included in the code is a secure wiping of memory after it has been used.

### PAMCredentialHandler.java

This source implements the CredentialHandler used in the Tomcat server.xml when authenticating users in the UserDatabaseRealm. Methods included in the com.broadcom.pam.secureTomcat.PCPCredentialHandler class are:

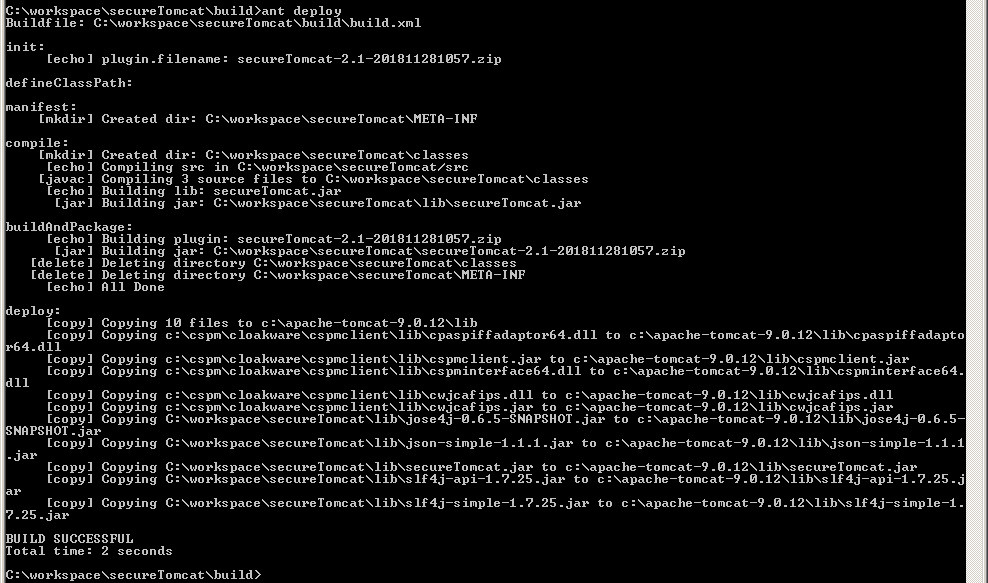
* public boolean matches (String inputCredentials, String alias)  
  This method receives the user entered password and the password stored in the user database. The second parameter is the value found in the tomcat-users.xml file. The password for the alias is requested and if it matches the inputCredentials password, the user is authenticated.
* public String mutate( String inputCredentials )  
  The method is required for the CredentialHandler class.

Not included in the code is a secure wiping of memory after it has been used.

## Building secureTomcat.jar

Verify that the directories mentioned in the build.properties file meets the installation of PAM A2A Client and Apache Tomcat.

From the build directory, run ant or ant deploy. This will compile the source code and create a secureTomcat.jar file in the lib directory. It will also create a new zip file including the directories used to build the file.



If everything goes as planned, a new zip-file is created which includes the compiled jar file, the build files and sources, and this document. The compiled jar is also found in the lib directory.

If ant is used without the deploy option, copy the lib\secureTomcat.jar and other library files to %CATALINA\_HOME%\lib directory.

When using ant deploy the necessary PAM A2A library files and the JWT library files are also copied.

# Sample application

There are two sample applications available. One is the Tomcat server application to received and decode JWT messages and one is a client application creating JWT messages and sending them to the Tomcat application.

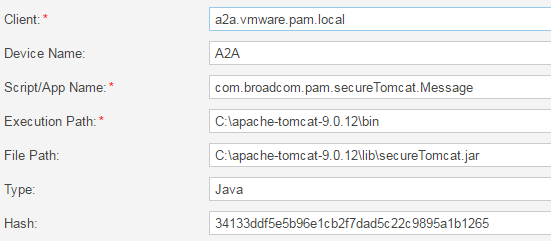
## Sample server - echoApp

The sample server web application is a simple echo application. It received a JWT message, decrypt the message and echo back the payload to the caller.

It must be deployed to the %CATALINA\_HOME%\webapps directory and unpacked (if automatic unpacking is false).

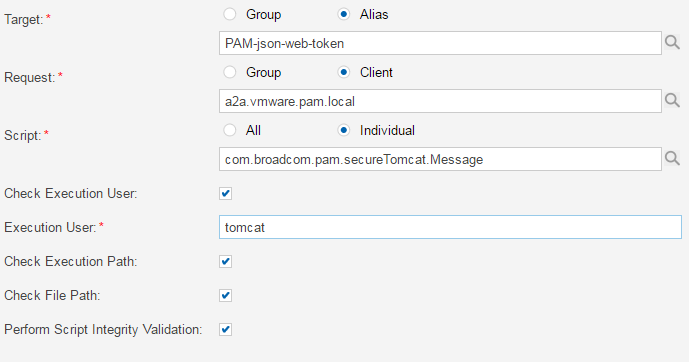
### A2A script

The sample server will retrieve the JWT password from PAM using A2A and the client must be registered as a valid script. Different PAM A2A clients may find different scripts for the JWT message decoding.



### A2A mapping

In PAM an A2A mapping is defined permitting the sampleServer to retrieve the JWT password.



### Sample run

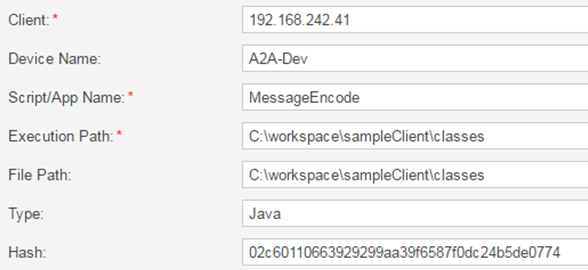


## Sample client

The sample client uses the same JWT mechanisms as expected by the jwtDecode method. By choice there is no jwtEncode method included in the secureTomcat.jar file.

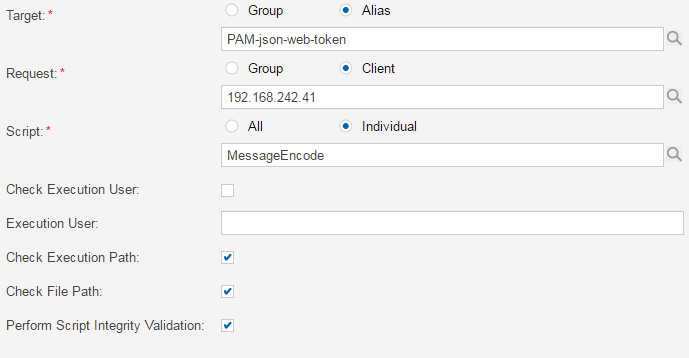
### A2A script

The sample client will retrieve the JWT password from PAM using A2A and the client must be registered as a valid script. Depending on where the client code resides, the execution and file paths may differ.

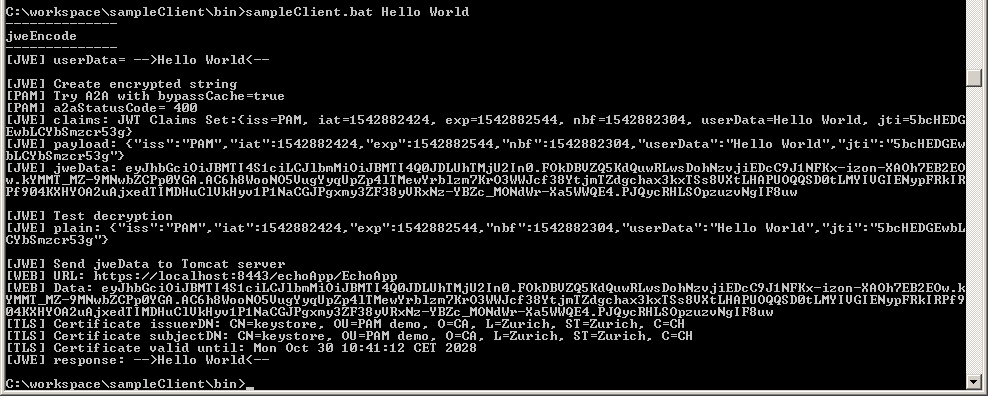


### A2A mapping

In PAM an A2A mapping is defined permitting the sampleClient to retrieve the JWT password.



### Sample run



# User access to Tomcat applications

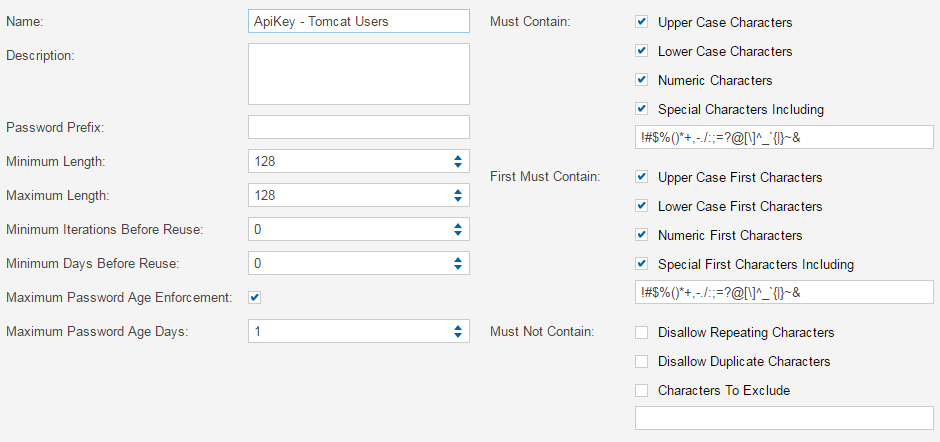
The example shown in this chapter is using username/password authentication for users found in the standard UserDatabase stored in %CATALINA\_HOME%\conf\tomcat-users.xml. The file includes the roles and users defined. Traditionally, the password is hardcoded in this file. It is fairly easy to setup Tomcat to store hashes of the user passwords and only compare the value provided by the user with the hashed value. This is far better than having the clear password stored.

The approach used in this example is to use the A2A client to fetch the current password for a user and compare it with what the user enters. In reality, it is not the human user who enters the password, but PAM is doing an automated login to the application.

## PAM setup

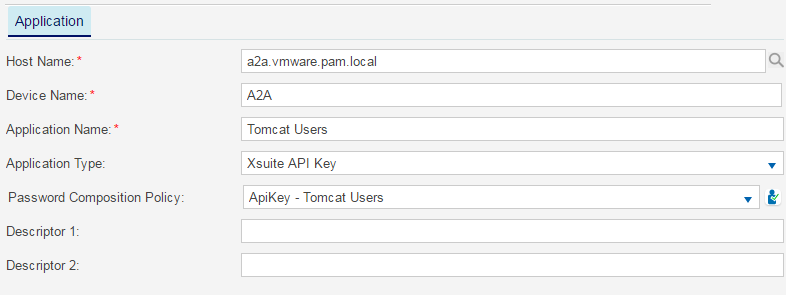
### PCP for user passwords

Passwords are generated randomly and should never be viewed or known to users. They are set to 128 characters (maximum 254 characters) and uses all character types. Most important is the password age set to 1 day, thus passwords are changed once a day.



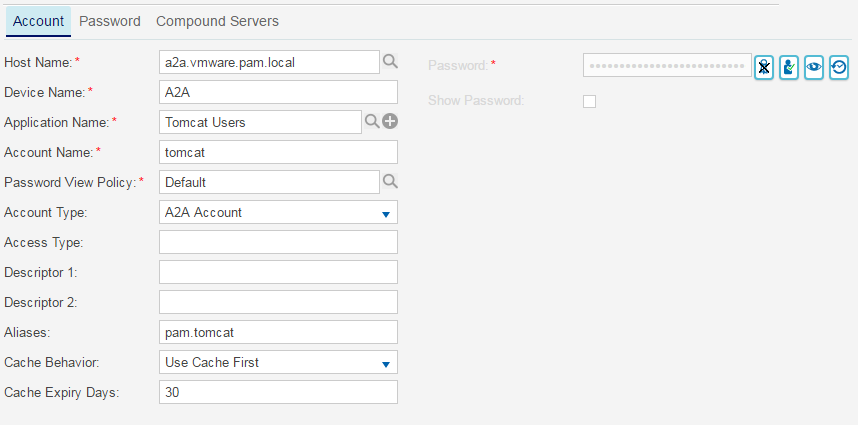
### Application for Tomcat users

The application is defined on the device where the Tomcat server is running.



### Account for Tomcat users

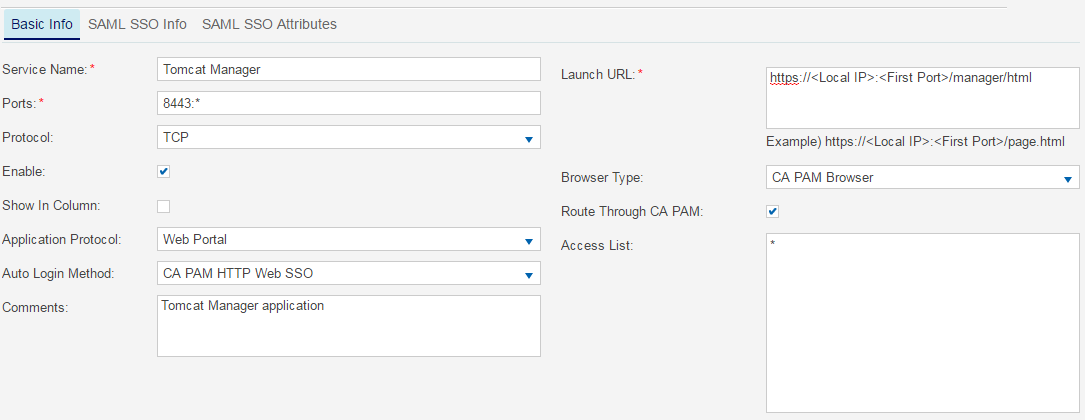
The account is having an account name matching the account name in the tomcat-users.xml file. The alias for the account in PAM must also match what is configured as the user password in the tomcat-users.xml file. The password can be defined as a new random value.



### TCP service for Tomcat application

The TCP application defined is specific for the application used. The example here is the Tomcat Manager application installed by default.

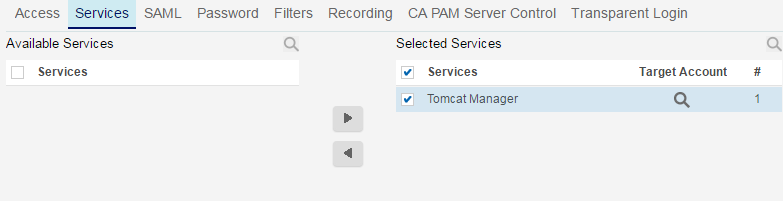
It uses automatic login at the HTTP level and uses the CA PAM Browser. The access list can probably be limited to less than the allow all used in the example here.



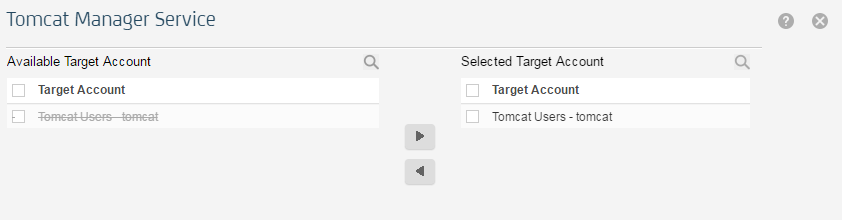
Not shown: This service is assigned to the device having Tomcat installed.

### Policy for user access to Tomcat application

A policy is created for a PAM user and the Tomcat device. User groups and device groups can be used.



The account used for automated login is the tomcat account having the pam.tomcat alias.



## Tomcat setup

### server.xml

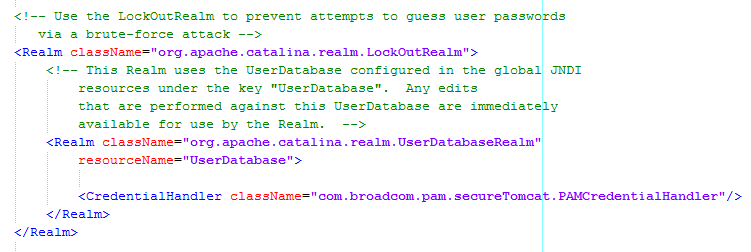
Find the section with <Engine name=”Catalina” ....

Add/replace the CredentialHandler for the UserDatabaseRealm.

<Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase">

<CredentialHandler className="com.broadcom.pam.secureTomcat.PAMCredentialHandler"/>

</Realm>



The PAMCredentialHandler is included in the secureTomcat.jar.

### tomcat-user.xml

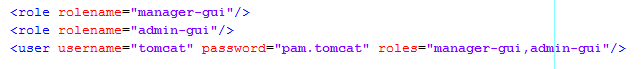
Edit the file tomcat-users.xml and add/change the roles and users.

The definition used here is:

<role rolename="manager-gui"/>

<role rolename="admin-gui"/>

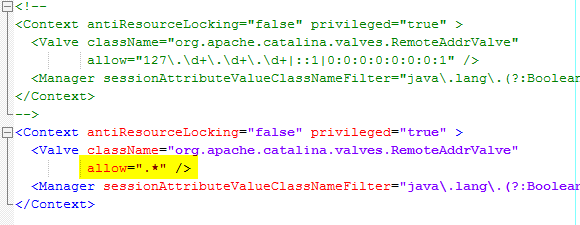
<user username="tomcat" password="pam.tomcat" roles="manager-gui,admin-gui"/>



Instead of a real password, the value used is a PAM account alias, which will be retrieved via the A2A client installed on the Tomcat server.

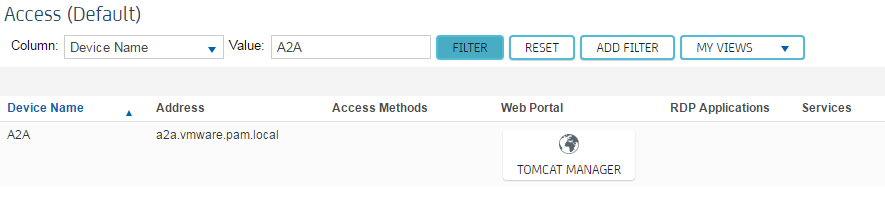
### context.xml

Default setting when installing Tomcat is that the manager application can only be called from localhost. To change that, edit the file %CATALINA\_HOME%\webapps\manager\META-INF\context.xml. Change the allowed addresses to .\* or a more restrictive address match.

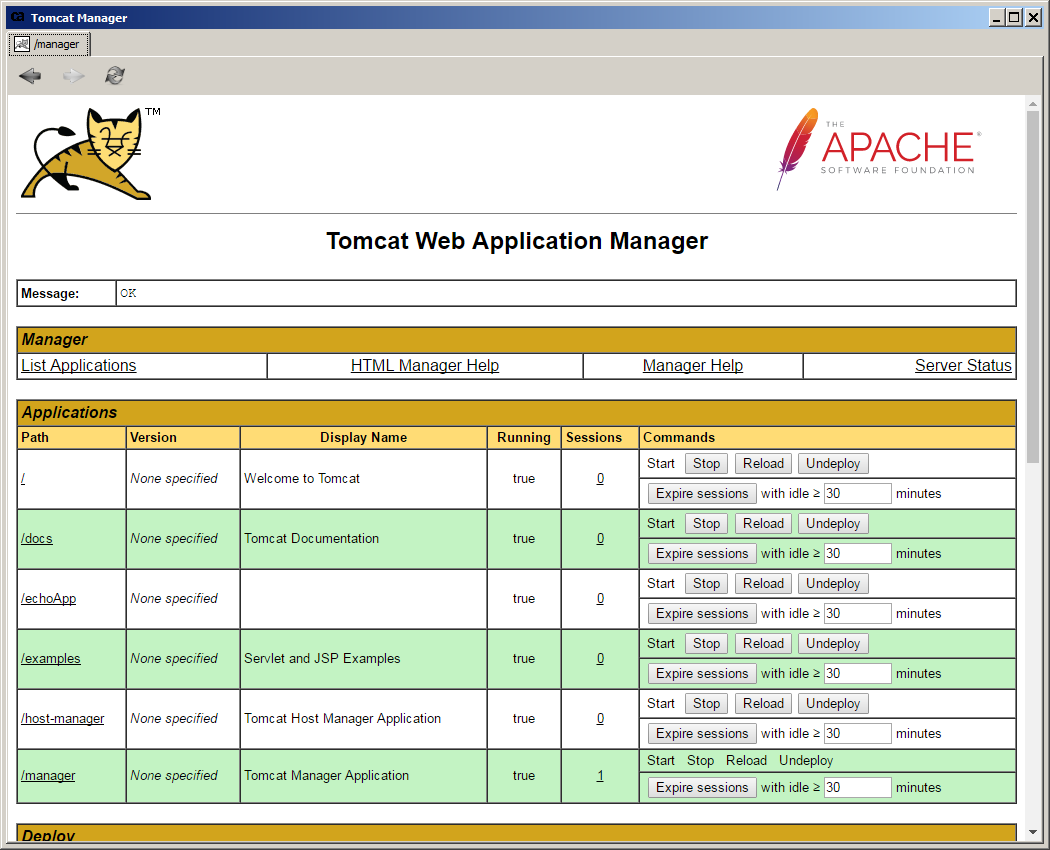


## Sample access

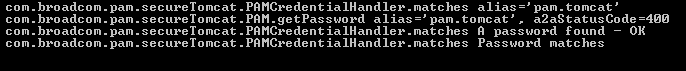
The user’s access page has a Web Portal for the TOMCAT MANAGER. Selecting this will open the Tomcat manager application.



If everything works, the web application is opened without prompting the user for a password.



The catalina.out log will show that the password for alias pam.tomcat is fetched and that it matches the password used by PAM when the web session is opened.



## User access using a password hash

An alternative to using A2A client for the user access is to store a hash of the user password in the tomcat-users.xml file. The challenge is that PAM cannot change the password automatically and the account password in PAM is static and can only be changed manually in both PAM and the tomcat-users.xml file.

If the setup described here is used, the account password is stored in PAM as a Generic account and used for automated login.

### server.xml

Edit the %CATALINA\_HOME%\conf\server.xml and change/update the CredentialHandler.

<Realm className="org.apache.catalina.realm.UserDatabaseRealm"

resourceName="UserDatabase">

**<CredentialHandler className="org.apache.catalina.realm.SecretKeyCredentialHandler"**

**algorithm="PBKDF2WithHmacSHA512"/>**

</Realm>

### User’s password hash

To calculate the user’s password hash, use the %CATALINA\_HOME%\bin\digest.bat script. It will show the hash to be stored in the tomcat-users.xml file as the password.

digest -a "PBKDF2WithHmacSHA512" -i 100000 -s 16 -k 256 -h "org.apache.catalina.realm.SecretKeyCredentialHandler" **MySecretPassword**

The result may look like this

MySecretPassword:**ecbbe17f4837502f348ad2537fa16ce2$100000$5c5e033dcb33343ccbf524b60abf0812d78ef2396144ba2426ad270c2a0dcc26**

The user’s password hash is the text after the :

In PAM store the user’s password (here MySecretPassword) for the account used in the policy for the Tomcat Manager web application.

# Security considerations

## Password for keystore is limited

The length and complexity of the password used to access the keystore is controlled by the PAM Password Composition Policy. In the example used in this document, it is 64-characters long and without special characters.

The length of the password can be changed to a maximum of 254 characters.

Special characters are omitted here. Reason is that the example used will create the keystore using a command in a script. Some special characters have special meaning in Windows (and UNIX) shell and if used in a password, the script to generate the keystore may fail.

If the password is entered manually special characters can be used without any concern.

## Callstack verification

Before the password is released the callstack is verified. A sha256 hash of the callstack is created and verified with an expected value. If the callstack is different from the expected, the password for the keystore is not retrieved.

If the Tomcat server is updated or patched, the callstack may change and the callstack hash value must be recreated and changed in the source code.

Verification of the callstack can be disabled. This will require a recompilation of the Java code.

## Filelist verification

Before the password is released integrity of files on the Tomcat server can be done. A filelist in %CATALINA\_HOME%\conf\pam.filelist is read and a combined hash of all files in the list is calculated. The expected hash is retrieved from PAM and verified to be identical.

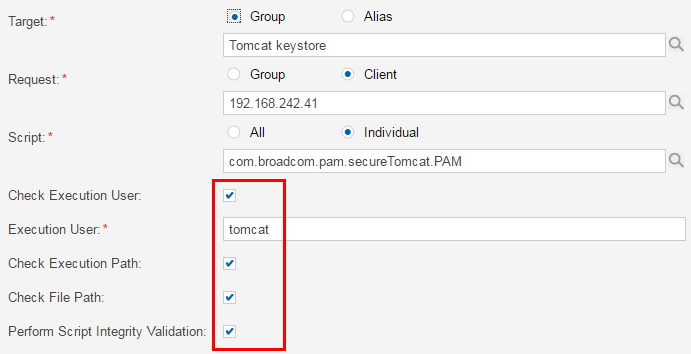
The filename of the filelist itself must be included in the list of files. If the filelist filename is not found in the filelist, the keystore password is not released.

If the calculated hash differs from the expected hash retrieved from PAM, the keystore password is not retrieved.

Verification of files from a filelist can be disabled. This will require a recompilation of the Java code.

## A2A script integrity

A built-in functionality in PAM is to verify the integrity of a script before releasing credentials to the calling application. For production setup, it is recommended to enable all the checks in the A2A mapping.



If one of the checks is failing, credentials will not be released to the Java class.

The class com.ca.pam.secureTomcat.PAM is self-contained and uses only known standard libraries. These libraries should be part of the filelist integrity checking.

## Keystore password is not updated automatically

Today with PAM 3.2.2 there is no easy mechanism to rotate a password used for a Java keystore. It means that any update of the password used to unlock the keystore is a manual process and when changed, the account in PAM must be updated with a new password.

A future version of PAM may include a mechanism to update a password of a keystore. Should this become available, the password for the keystore should be rotated frequently and automatically. It is possible that a UNIX script can accomplish the same, but this has not been considered further.

## Incorrect classes used

When using different classes for different parts of the secureTomcat library, there is always a risk that someone will create a bogus class with the same name. If that class is loaded first, this will be used by Tomcat. PAM is registering for the class calling the A2A client as “the script”. Ideally, if you place everything in one class, the integrity mechanism in A2A verifies that the correct code is executed before passwords are released.

Reality shows that you cannot always use a single class. Sometimes, you must extend existing classes (override) and if you need more than one class, you cannot have everything in one class.

Also, it has been found that the class registered as script in PAM is not always the class calling A2A. Depending on the version of the A2A client it may or may not be so, or it is a different class which is registered as script.

In the example code, the class constructor for the class PAM will verify which class is instantiating the PAM class. If the class is known the jar file is verified to be secureTomcat.jar. If it is a different jar-file, releasing passwords will not work.

There may be ways to bypass these additional checks and other ways to guarantee system integrity of the running Tomcat environment should also be considered.

## Man in the Middle

The mechanism described here does allow validation of the server certificate (of the Tomcat server) when establishing the TLS connection. If the client connecting to the Tomcat server is validating the server certificate or not is outside the scope of this document.

## File system access

The mechanism described here does integrity validation of files listed in the pam.filelist file. Regardless of this, it is best practise to limit access to the file system of production servers.

### Webapps protections

An obvious choice is to include one or more Tomcat applications in the list of files verified. If using a standard Tomcat installation, new or updated war files in the webapps directory are automatically deployed and loaded when the classes are used. The integrity verification of files is only done when Tomcat starts and retrieve the keystore password, thus any later updates of war files may compromise the system integrity. One possible mitigation is to disable automatic unpacking of war files and to disable automatic deployment of updated war files.

The additional security of integrity verification of the (manually) deployed application is an extra operational step. It is recommended to do so for production environments, where updates to applications should be strictly controlled. See section “4.2 - Optional configuration” for more details.

## Keystore password and private key in memory

When Tomcat fetches the password to access the password to unlock the private key from the keystore, information may be kept in memory and may be accessible. It has not been investigated how or if the memory for the password to the keystore or the private key itself is kept in Tomcat.

## Firewall rules

The secure access mechanism to the keystore described here will require some firewall rules when communicating with PAM.

| **Source** | **Destination** | **Port** | **Protocol** | **Description** |
| --- | --- | --- | --- | --- |
| PAM servers | Tomcat server | 28888 | TCP | Notifications and script integrity requests from PAM to A2A client. |
| Tomcat server | PAM servers | 443 | TCP/HTTPS | A2A client calling back to PAM |

# Troubleshooting

## Lots of exceptions when starting Tomcat

Problem:  
The A2A call retrieves the password (return code 400), but there are lots of exceptions being thrown when starting Tomcat.

For example:

01-Nov-2018 14:59:46.063 INFO [main] org.apache.coyote.AbstractProtocol.init Initializing ProtocolHandler ["https-openssl-nio-8443"]

01-Nov-2018 14:59:46.401 SEVERE [main] org.apache.catalina.util.LifecycleBase.handleSubClassException Failed to initialize component [Connector[HTTP/1.1-8443]]

org.apache.catalina.LifecycleException: Protocol handler initialization failed

at org.apache.catalina.connector.Connector.initInternal(Connector.java:935)

at org.apache.catalina.util.LifecycleBase.init(LifecycleBase.java:136)

at org.apache.catalina.core.StandardService.initInternal(StandardService.java:533)

at org.apache.catalina.util.LifecycleBase.init(LifecycleBase.java:136)

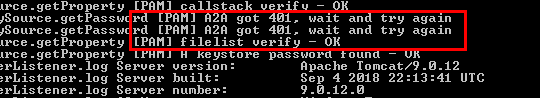
at org.apache.catalina.core.StandardServer.initInternal(StandardServer.java:852)

Reason:  
Possible reason is that the password returned, and the password needed for decrypting the keystore are different.

Solution:  
Rebuild the keystore or update the account password in PAM with the password used for the keystore.

## 401 - User not authorized

Problem:  
Occasionally when calling the A2A Client to retrieve credentials, the A2A Client returns “401”. This is a user not authorized for A2A calls.



Reason:  
Unknown what the reason for this error message is. There is no dedicated A2A client user.

Solution:  
Wait a bit and try retrieving the credentials again. The wait/retry mechanism is built into the class com.broadcom.secureTomcat.PAM and will in most cases be sufficient to recover from this error. If it is still failing after 10 attempts, then something different is wrong and the credentials are not retrieved.

## 402 - A2A Client not started

Problem:  
The A2A call returns an error code 402.

Reason:  
The A2A Client is not started.

Solution:  
Start the A2A client.

## 405 - Alias not found

Problem:  
The A2A call returns an error code 405.

Reason:  
The account alias is not found.

Solution:  
Verify/correct the pam.keystore.alias entry in catalina.properties to match the account alias in PAM. The alias is case sensitive.

## 409 - Unauthorized script name

Problem:  
The A2A call returns an error code 409.

Reason:  
The A2A mapping uses an incorrect script name.

Solution:  
Correct the A2A script to match the mapping and script program.

## 410 - Unauthorized execution path

Problem:  
The A2A call returns an error code 410.

Reason:.  
The A2A script contain a wrong execution path.

Solution:  
Change the A2A script to use the correct execution path.

## 411 - Unauthorized execution user

Problem:  
The A2A call returns an error code 411.

Reason:  
The user running the Tomcat server is not authorized to retrieve credentials.

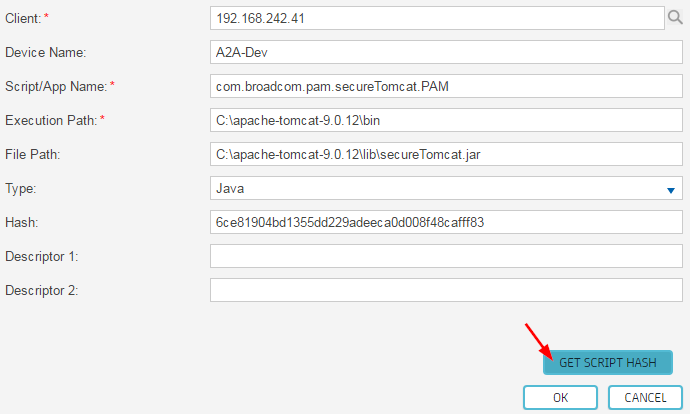
Solution:  
Verify that the A2A mapping includes the username of the Tomcat user.

## 436 - Incorrect Script Integrity value

Problem:  
The A2A call returns an error code 436.

Reason:  
The A2A Script for the program Java class has an incorrect hash value. Either it is configured incorrectly in PAM or someone modified a changed version of the secureTomcat.jar file on the Tomcat server.

Solution:  
If it is known that a new or updated version of the secureTomcat.jar is deployed, use the PAM GUI to update the script integrity value.

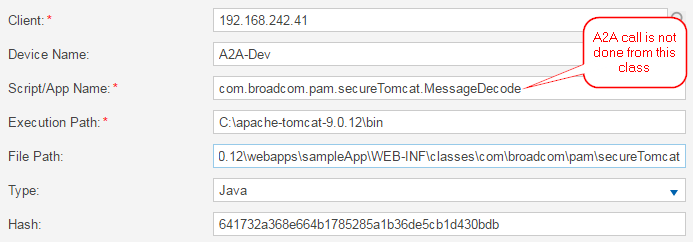


This will send a request to the A2A client and refresh the script integrity value.

Alternative is to uncheck the “Perform Script Integrity Validation” on the A2A mapping.   
This is not recommended for a production environment.

## Script name incorrect

Problem:  
The script registered when the A2A client contacts PAM is incorrect.



Reason:  
Different versions of the A2A Client behaves differently. The Java class registered in PAM should be the class making the A2A retrieveCredentials call. Version 4.12.1 (and before) will register the calling class if this is the only class being used. If the call to the A2A retrieveCredentials is done via other classes, one of these is what will be registered. If for example a class in a Tomcat application (webapps) is calling a method using retrieveCredentials is different from the application class, it is the webapps class that is registered. If you use A2A Client version 4.13 or newer, it is the class calling requestCredentials.

Solution:  
No good solution exists. The secureTomcat.jar example used in this document will register correct for different A2A client versions 4.12.1 and 4.15.0 (the ones tested). The webapps sample application will register differently depending on the client version. For the secureTomcat.jar usage the A2A mapping with script integrity validation can be used to verify the integrity of the class. It includes the ability to verify the integrity of other files (using a filelist), and all the files in the application should be included in the list. I.e. even though the script registered for the webapps may be incorrect, the filelist verification done before releasing the keystore password can be used to verify the integrity of applications.