ARCWAY AG

Hinweise zu Sicherheitsmaßnahmen

English

16.12.2022

ARCWAY AG

Potsdamer Platz 10
10785 Berlin
GERMANY
+49 30 800 97 83 0 (Phone)
+49 30 800 97 83 111 (Fax)
info@arcway.com
Chairman: Karsten Wulfes
Chairman of the Supervisory Board: Uwe Barchmann
Document: AC.007.EN-1602*
16.12.2022
This document is created with ARCWAY Cockpit.

LIST OF CONTENTS

LIST OF FIGURES

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

1 INTRODUCTION

Preface

This document provides instructions for setting up a secure environment for Apache Tomcat 9. This guide considered version 9.0 and does not include changes against updates provided by Apache.

> *The performance can be negatively influenced by some of the described measures.*

Target group

This document is intended for system / application administrators and security experts who want to install an ARCWAY Cockpit Server on an Apache Tomcat 9 environment.

2 TOMCAT SSL CONFIGURATION

The data transfer between client and server can be done via SSL encrypted HTTP connections (HTTPS).

SSL support is not activated after the standard installation of an ARCWAY Cockpit server. Before SSL-encrypted HTTP connections can be used, configuration steps have to be carried out on both the Cockpit Server and the Cockpit clients. These steps are described in the following paragraphs using the standard installation of an ARCWAY Cockpit Server.

2.1 Server-side Configuration

Creation of a certificate/ asymmetric key pair

Prerequisite for the activation of SSL support is an asymmetric key pair usable for SSL encryption.

For further understanding the term certificate and the handling of such certificates by client Java VM's and the Tomcat server is important. Therefore, a short explanation of these relationships follows: A certificate is a data packet issued by a certification authority that serves to make the owner and other properties of a published key verifiable. Every Java VM installation includes a database with information about trusted certification authorities. Certificates issued by certification authorities for which no information is recorded in this database cannot be checked for authenticity. If the Tomcat server has a certificate for the public SSL key it uses, it will transmit this certificate to its clients when establishing SSL connections.

For an SSL connection to be established successfully, the client Java VM must trust the server's public SSL key. Trusted keys must meet one of the following conditions:

- The client Java VM has a certificate whose authenticity you can check and which assigns the public key of the server to an owner.

- The public key of the server was previously made known to the client VM as trusted.

Due to these interrelationships, it is recommended, even for smaller installations, to have a certificate issued for the server's public key, which can be verified by all Java VMs considered for client operation. This is recommended for security reasons. On the other hand, the configuration effort for setting up new clients can be reduced to a minimum if the appropriate certificate is stored on the server. In particular, no effort is then required to *transport* the public key of the server to all client VMs and to make it known there as trustworthy.

The creation of a key pair/certificate usable for the present use case practically always starts with the generation of a key pair from the Java SDK using the *keytool* command. After the execution of the command, the key pair is stored in an archive file, which is usually called **Keystore**.

Example:

```
keytool -genkey -alias tomcat -keyalg RSA -keysize 2048 -keystore
tomcat.keystore
```

The result of the above command is a file named **tomcat.keystore** which contains a key pair of the desired type.

If the use of a certificate is waived, the **tomcat.keystore** created in this first step already contains all contents relevant for use by the server. If a certificate is to be used, the following additional steps are required:

1. *Creation of a signing request* To apply for a certificate, the relevant data must be transmitted to the certification authority. The transmission of this data is usually done by sending a **signing request** file, which can be generated with the *keytool* command:

```
keytool -certreq -keystore tomcat.keystore -alias tomcat -file tomcat.csr
```

2. *Sending the Signing Request file to the certification authority*. After successful examination of the certification application, the certification body will send you the finished certificate in the form of a **.crt** file.

3. *Import certificate / deposit it on the server*. To *store the certificate on the server*, you have to import the certificate into the **tomcat.keystore**

created before. After that it will be made known to the server together with the key pair it contains. The import of the certificate into the keystore can be performed by a command of the following form:

```
keytool -import -trustcacerts -file tomcat.crt -alias tomcat -keystore
tomcat.keystore
```

Enable HTTPS / SSL support

The activation of SSL support in Tomcat is done by activating the *Connector* definition already present as a comment in the **server.xml** and adjusting the corresponding parameter values. The following *Connector* definition causes Tomcat to process HTTPS connection requests arriving on port *8443* and to use the first key pair/certificate contained in the keystore file **c:\tomcat.keystore** for SSL encryption The password *changeit* is used as decryption password for the keystore file.

```
<-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
<Connector
        port="8443"
        maxHttpHeaderSize="8192"
        maxThreads="150"
        minSpareThreads="25"
        maxSpareThreads="75"
        enableLookups="false"
        disableUploadTimeout="true"
        acceptCount="100"
        scheme="https"
        secure="true"
        clientAuth="false"
        sslProtocol="TLS"
        keystoreFile="c:\tomcat.keystore"
        keystorePass="changeit"
    />
```

This means that SSL support is already activated. For more information on configuring SSL support, see the Apache Tomcat online documentation *SSL Configuration HOW-TO*:

http://tomcat.apache.org/tomcat-9.0-doc/ssl-howto.html

The references of the officially recognised certification bodies are sometimes somewhat confusing and there may actually be other ways of generating the key pair. If a different procedure for generating the certificate is proposed, one should make sure that the certificate generated in the described manner is actually suitable for the present application.

Prevent access via HTTP

Enabling HTTPS/SSL support means that HTTPS can be used as a communication protocol in addition to the standard support for HTTP. It is no problem for the Cockpit Server if both types of connection are allowed and used simultaneously. However, if you only want to use HTTPS connections, we recommend that you explicitly switch off support for HTTP.

To completely disable HTTP support, you can delete or comment out the definition of the HTTPS "connector" in the "server.xml". However, this option must not be used in the standard installation of ARCWAY Cockpit, because in addition to the actual Cockpit Server web application, the Cockpit License Server web application is also operated on the Tomcat Servlet Container. If HTTP support were to be completely switched off, communication between cockpit server and cockpit license server would no longer be possible and the "concurrent licenses" possibly existing on the cockpit license server would no longer be usable. Therefore, we recommend that you do not switch off HTTP support completely, but only restrict access via HTTP.

In the past, it has proven useful in this context to restrict access via HTTP so that HTTP connections are only allowed on the loopback interface. Since the loopback interface can only be accessed from the local computer, access from other computers is then impossible. To achieve this restriction, the "Connector" definition of the HTTP connector must be added to the parameter address="127.0.0.1". The "Connector" definition of the HTTP connector could look like this:

```
<Connector
       port="8080"
       address="127.0.0.1"
       redirectPort="8443"
```

```
      minSpareThreads="25"
      connectionTimeout="20000"
      maxSpareThreads="75"
      maxThreads="150">
   </Connector>
```

2.2 Client-side Configuration

On the client side, the use of HTTPS / SSL encryption instead of HTTP results in the following differences:

URLs

The server URLs now start with "https", and typically the server is also accessed via a different port (8443 instead of 8080). Here is an example of an HTTPS URL:

https://192.168.35.12:8443/CockpitServer/

Making the public server key of the client VM known

Before a connection can be established via HTTPS, the client Java VM must classify the server's public SSL key as trustworthy. This is done on the basis of the criteria mentioned at the beginning.

Therefore, the configuration steps described in this section only need to be carried out if no certificate has been stored on the server.

The server's public key is made known to the client by specifying a "Trust Store" file containing this key and a password for decrypting the contents of this file. The location of the "Trust Store" file and the associated password can be set in the dialogue for **User Defaults** under the item **ARCWAY Cockpit**. This dialogue can be opened via the menu item **Windows » User Defaults**:

Such a "Trust Store" file has the same format as the "tomcat.keystore" file created during the server setup. In fact, you could even use the "tomcat.keystore" file and the associated password directly as a "Trust Store" file and "Trust Store" password. However, this is not recommended, since this file contains the public key and the corresponding private key of the server, which should be kept secret for security reasons.

To create a "Trust Store" file that contains the public key of the server but not the private key of the server, you can:

4. Extract the public key contained in the tomcat.keystore file using the following command

```
keytool -export -alias tomcat -file tomcat.cert -keystore tomcat.keystore
```

5. Create a new keystore where the public key of the server can be "stored" afterwards. Unfortunately you cannot create empty keystores with the "keytool" commands. Therefore a new, filled keystore must be created first. From this keystore, the key pair stored during the creation must be deleted. Here are the appropriate commands:

```
keytool -genkey -alias ungenutzt -keystore client.truststore
keytool -delete -alias ungenutzt -keystore client.truststore
```

6. Then the public key of the server can be imported into the so prepared keystore:

```
keytool -import -keystore client.truststore -file tomcat.cert
```

The screenshot below shows a log that was recorded during the exemplary execution of the above steps:

During the logged execution of the steps a "Trust Store" file with the name "client.truststore" was created, which is protected with the password "michelangelo”.

Proxy Configuration

If access is not to take place directly, but via a proxy, the SSL proxy settings must also be configured in addition to an already configured HTTP proxy.

The proxy settings are to be made in the dialogue **User Preferences** under the item **General » Network Connections**. This dialogue can be opened via the menu item **Windows » User Preferences**.

3 REMOVE EXTERNAL RESOURCES

Description

The Apache Tomcat 9 installation may contain examples of applications, documentation and other directories that should not be used for production.

By default, the subdirectories **docs, examples, ROOT, manager and host-manager** are present in **TOMCAT_HOME/webapps**.

Recommendation for action

You should delete all standard applications from the directory that you do not use.

If you are using the **manager and host-manager**, you do not need to remove these applications, this document will not discuss any security aspects of these applications.

Reference

Default_web_applications/General

4 SERVER INFORMATION LEAKS

A lot of information makes it easier for an attacker to determine which vulnerabilities exist on a server.

Therefore it is recommended to restrict some information.

Note that, if you patch your Tomcat server regularly, the following steps from this chapter must be performed repeatedly.

4.1 Limiting information about the Tomcat

The *server.info* attribute contains information about the version and build of the Tomcat server.

This value is transmitted when clients connect to the Tomcat server.

> *By modifying the **ServerInfo.properties** file, the complexity for attackers to determine which vulnerabilities could affect the server can be increased.*

If you want to change the *server.info* attribute, proceed as follows. 1.

1. extract the **catalina.jar**

The attribute *server.info* is part of the **catalina.jar**.

To adapt this attribute, it is necessary to decompile the **catalina.jar** and then compile it again.

```
$cd $CATALINA_HOME/lib
$jar xf catalina.jar org/apache/catalina/util/ServerInfo.properties
```

2. navigate to the created util folder

/org/apache/cataline/util

3. edit **ServerInfo.properties** in the editor.

The default value for the *server.info* attribute is **Apache Tomcat /ServerVersion**

Change this value as follows and then save your changes:

|[[code server.info=Arcway Cockpit Server

4. adjustment of the attribute *server.build*

The default value for the *server.build* attribute is the date and time of the build.

> *Example: Jul 8 2008 11:40:35*

```
server.build=Jan 1 2000 01:01:01
```

5. update **catalina.jar** with the modified attributes of *server.info*.

```
$cd $CATALINA_HOME/lib
$jar to catalina.jar org/apache/catalina/util/ServerInfo.properties
```

4.2 Modify Server Header

The server attribute controls the value of the server's HTTP header.

The default value of this header is for Tomcat 4.1.x to 8.0.x *Apache-Coyote/1.1*.

As of 8.5.x this header is no longer the default. This header can provide both clients and attackers with information about the type of server.

In the file **$CATALINA_HOME/conf/server.xml** in the section [[STYLE#EMPHASIS|<Connector />]] make the following change:

```
<Connector port="8443" server="I am a Cockpit Server"
redirectionPort="8080" />
```

4.3 Tomcat shutdown port

By default, the Tomcat Server listens on the TCP port **8005** and accepts shutdown requests.

Connecting to this port and transmitting the command **SHUTDOWN** (in the default configuration) will cause the Tomcat Server to shut down.

It is recommended to replace the default value **SHUTDOWN** with an arbitrary string or even better to disable the port for the SHUTDOWN sequence.

In the file **$CATALINA_HOME/conf/server.xml** in the section *<Connector />* make the following change:

```
<Connector port="8443" server="I am a Cockpit Server"
redirectionPort="8080" />
```

5 LINUX

5.1 Tomcat user

For security reasons, the Tomcat server should not be run under the **root** user.

For this purpose, a new system user and a new group should be created with the Tomcat's home directory **TOMCAT_HOME**, which will run the Tomcat service.

```
sudo useradd -r -m -U -d /TOMCAT_HOME -s /bin/false tomcat
```

> TOMCAT_HOME *im Codeblock muss durch das Verzeichnis des Tomcats ersetzt werden*

5.2 Directory and file rights

As we mentioned in the previous section, Tomcat will run under the Tomcat user. This user must have access to the **TOMCAT_HOME** directory.

> ***TOMCAT_HOME** in the codeblock must be replaced by the directory of the Tomcat.*

The following command changes the directory ownership to the user and group tomcat:

```
sudo chown -RH tomcat: /TOMCAT_HOME
```

The scripts in the bin directory must have the executable flag:

```
sudo sh -c 'chmod +x /TOMCAT_HOME/bin/*.sh'
```

5.3 Tomcat SYSTEMD Deamon

To run Tomcat as a service, you must create a new file for **systemd**.

1. Open your text editor and create a file called **tomcat.service** in the **/etc/systemd/system/** directory:

```
sudo nano /etc/systemd/system/tomcat.service
```

2. Paste the following configuration:

```
[Unit]
Description=Tomcat 9 - Arcway Cockpit Server
After=network.target
[Service]
Type=forking
User=tomcat
Group=tomcat
Environment="JAVA_HOME=/usr/lib/jvm/default-java"
Environment="JAVA_OPTS=-Djava.security.egd=file:///dev/urandom -
Djava.awt.headless=true"
Environment="CATALINA_BASE=/TOMCAT_HOME"
Environment="CATALINA_HOME=/TOMCAT_HOME"
Environment="CATALINA_PID=/TOMCAT_HOME/temp/tomcat.pid"
Environment="CATALINA_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC"
ExecStart=/TOMCAT_HOME/bin/startup.sh
ExecStop=/TOMCAT_HOME/bin/shutdown.sh
[Install]
WantedBy=multi-user.target
```

> ***TOMCAT_HOME*** *in the codeblock must be replaced by the directory of the Tomcat.*
>
> *Change the value of* ***JAVA_HOME*** *if the path to your Java installation is different.*

5. Check the status of the service with the following command:

```
sudo systemctl status tomcat
```

> *Check the output for errors and that the Tomcat is accessible.*

3. Save and close the file and inform the system that a new systemd file is available:

```
sudo systemctl daemon-reload
```

4. Start the Tomcat service by running:

```
sudo systemctl start tomcat
```

6. Set up the autostart of the server

```
sudo systemctl enable tomcat
```