Securing MobilFirst server

Overview

Below are several methods you can follow in order to secure your MobileFirst Server instance.

Jump to

- Configuring App Transport Security (ATS)
- LDAP configuration for containers

Configuring App Transport Security (ATS)

ATS configuration does not impact applications connecting from other, non-iOS, mobile operating systems. Other mobile operating systems do not mandate that servers communicate on the ATS level of security but can still communicate with ATS-configured servers. Before configuring your server, have the generated certificates ready. The following steps assume that the keystore file **ssl_cert.p12** has the personal certificate and **ca.crt** is the signing certificate.

1. Copy the **ssl_cert.p12** file to the **mfpf-server-libertyapp/usr/security/** folder.
2. Modify the **mfpf-server-libertyapp/usr/config/keystore.xml** file similar to the following example configuration:

```
<server>
   <featureManager>
      <feature>ssl-1.0</feature>
   </featureManager>
   <ssl id="defaultSSLConfig" sslProtocol="TLSv1.2" keyStoreRef="defaultKeyStore" enabledCiphers="TLS_ECDHE_ECD
SA_WITH_AES_256_GCM_SHA384" />
   <keyStore id="defaultKeyStore" location="ssl_cert.p12" password="******" type="PKCS12"/>
</server>
```

   - **ssl-1.0** is added as a feature in the feature manager to enable the server to work with SSL communication.
   - **sslProtocol="TLSv1.2"** is added in the ssl tag to mandate that the server communicates only on Transport Layer Security (TLS) version 1.2 protocol. More than one protocol can be added. For example, adding **sslProtocol="TLSv1+TLSv1.1+TLSv1.2"** would ensure that the server could communicate on TLS V1, V1.1, and V1.2. (TLS V1.2 is required for iOS 9 apps.)
   - **enabledCiphers="TLS*ECDHE*ECDSA*WITH*AES*256*GCM_SHA384"** is added in the ssl tag so that the server enforces communication using only that cipher.
   - The **keyStore** tag tells the server to use the new certificates that are created as per the above requirements.

The following specific ciphers require Java Cryptography Extension (JCE) policy settings and an additional JVM option:

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

The the policy files are already installed in the Liberty for Java runtime and you dont have to add tham to the package again.Just add the following JVM option to the **mfpf-server-libertyapp/usr/env/jvm.options file**: `Dcom.ibm.security.jurisdictionPolicyDir=/opt/ibm/wlp/usr/servers/worklight/resources/security/`.

For development-stage purposes only, you can disable ATS by adding following property to the info.plist file:

```
<key>NSAppTransportSecurity</key>
<dict>
   <key>NSAllowsArbitraryLoads</key>
      <true/>
</dict>
```

Security configuration for IBM MobileFirst Foundation

Your IBM MobileFirst Foundation instance security configuration should include encrypting passwords, enabling application authenticity checking, and securing access to the consoles.

Encrypting passwords

Store the passwords for MobileFirst Server users in an encrypted format. You can use the securityUtility command available in the Liberty profile to encode passwords with either XOR or AES encryption. Encrypted passwords can then be copied into the /usr/env/server.env file. See Encrypting passwords for user roles configured in MobileFirst Server for instructions.

Application-authenticity validation

To keep unauthorized mobile applications from accessing the MobileFirst Server, enable the application-authenticity security check. Learn more...

Securing a connection to the back end

If you need a secure connection between your container and an on-premise back-end system, you can use the Bluemix® Secure Gateway service. Configuration details are provided in this article: Connecting Securely to On-Premise Backends from MobileFirst on IBM Bluemix containers.

Encrypting passwords for user roles configured in MobileFirst Server

The passwords for user roles that are configured for the MobileFirst Server can be encrypted.
Passwords are configured in the **server.env** files in the **package_root/mfpf-server-liberty-app/usr/env** . Passwords should be stored in an encrypted format.

1. You can use the `securityUtility` command in the Liberty profile to encode the password. Choose either XOR or AES encryption to encode the password.
2. Copy the encrypted password to the **server.env** file. Example: `MFPF_ADMIN_PASSWORD={xor}PjsyNjE=`
3. If you are using AES encryption and used your own encryption key instead of the default key, you must create a configuration file that contains your encryption key and add it to the **usr/config** directory. The Liberty server accesses the file to decrypt the password during runtime. The configuration file must have the .xml file extension and resemble the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
   <variable name="wlp.password.encryption.key" value="yourKey" />
</server>
```

Restricting access to the consoles running on containers

You can restrict access to the MobileFirst Operations Console and the MobileFirst Analytics Console in production environments by creating and deploying a Trust Association Interceptor (TAI) to intercept requests to the consoles.

The TAI can implement user-specific filtering logic that decides if a request is forwarded to the console or if an approval is required. This method of filtering provides the flexibility for you to add your own authentication mechanism if needed.

See also: Developing a custom TAI for the Liberty profile (https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_dev_custom_tai.html?view=embed)

1. Create a custom TAI that implements your security mechanism to control access to the MobileFirst Operations Console. The following example of a custom TAI uses the IP Address of the incoming request to validate whether to provide access to the MobileFirst Operations Console or not.

```
package com.ibm.mfpconsole.interceptor;
import java.util.Properties;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.websphere.security.WebTrustAssociationException;
import com.ibm.websphere.security.WebTrustAssociationFailedException;
import com.ibm.wsspi.security.tai.TAIResult;
import com.ibm.wsspi.security.tai.TrustAssociationInterceptor;

public class MFPConsoleTAI implements TrustAssociationInterceptor {
```

```
public class MFPConsoleTAI implements TrustAssociationInterceptor {

  String allowedIP =null;

  public MFPConsoleTAI() {
    super();
  }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#isTargetInterceptor
 * (javax.servlet.http.HttpServletRequest)
 */
  public boolean isTargetInterceptor(HttpServletRequest req)
            throws WebTrustAssociationException {
    //Add logic to determine whether to intercept this request

    boolean interceptMFPConsoleRequest = false;
    String requestURI = req.getRequestURI();

    if(requestURI.contains("mfpconsole")) {
      interceptMFPConsoleRequest = true;
    }

    return interceptMFPConsoleRequest;
  }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#negotiateValidateandEstablishTrust
 * (javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
 */
  public TAIResult negotiateValidateandEstablishTrust(HttpServletRequest request,
              HttpServletResponse resp) throws WebTrustAssociationFailedException {
     // Add logic to authenticate a request and return a TAI result.
     String tai_user = "MFPConsoleCheck";

     if(allowedIP != null) {

        String ipAddress = request.getHeader("X-FORWARDED-FOR");
        if (ipAddress == null) {
          ipAddress = request.getRemoteAddr();
        }

        if(checkIPMatch(ipAddress, allowedIP)) {
          TAIResult.create(HttpServletResponse.SC_OK, tai_user);
        }
        else {
          TAIResult.create(HttpServletResponse.SC_FORBIDDEN, tai_user);
        }

     }
     return TAIResult.create(HttpServletResponse.SC_OK, tai_user);
  }

  private static boolean checkIPMatch(String ipAddress, String pattern) {

    if (pattern.equals("*.*.*.*") || pattern.equals("*"))
        return true;

    String[] mask = pattern.split("\\.");
    String[] ip_address = ipAddress.split("\\.");

    for (int i = 0; i < mask.length; i++)
    {
      if (mask[i].equals("*") || mask[i].equals(ip_address[i]))
        continue;
      else
        return false;
    }
    return true;
```

```
  }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#initialize(java.util.Properties)
 */
  public int initialize(Properties properties)
            throws WebTrustAssociationFailedException {

    if(properties != null) {
      if(properties.containsKey("allowedIPs")) {
        allowedIP = properties.getProperty("allowedIPs");
      }
    }
    return 0;
  }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#getVersion()
 */
  public String getVersion() {
    return "1.0";
  }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#getType()
 */
  public String getType() {
    return this.getClass().getName();
  }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#cleanup()
 */
  public void cleanup()

  {}
}
```

2. Export the custom TAI Implementation into a .jar file and place it in the applicable **env** folder (**mfpf-server-libertyapp**/**usr**/**env**).
3. Create an XML configuration file that contains the details of the TAI interceptor (see the TAI configuration example code provided in step 1) and then add your .xml file to the applicable folder (**mfpf-server-libertyapp**/**usr**/**config**). Your .xml file should resemble the following example. **Tip:** Be sure to update the class name and properties to reflect your implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">
<featureManager>
  <feature>appSecurity-2.0</feature>
</featureManager>

<trustAssociation id="MFPConsoleTAI" invokeForUnprotectedURI="true"
            failOverToAppAuthType="false">
  <interceptors id="MFPConsoleTAI" enabled="true"
            className="com.ibm.mfpconsole.interceptor.MFPConsoleTAI"
            invokeBeforeSSO="true" invokeAfterSSO="false" libraryRef="MFPConsoleTAI">
    <properties allowedIPs="9.182.149.*"/>
  </interceptors>
</trustAssociation>

<library id="MFPConsoleTAI">
  <fileset dir="${server.config.dir}" includes="MFPConsoleTAI.jar"/>
</library>

</server>
```

4. Re-deploy the server. The MobileFirst Operations Console is now accessible only when the configured TAI security mechanism is satisfied.

LDAP configuration for containers

You can configure an IBM MobileFirst Foundation to securely connect to an external LDAP repository.

The external LDAP registry can be used for the following purposes:

- To configure the MobileFirst administration security with an external LDAP registry.
- To configure the MobileFirst mobile applications to work with an external LDAP registry.

Configuring administration security with LDAP

Configure the MobileFirst administration security with an external LDAP registry.
The configuration process includes the following steps:

- Setup and configuration of an LDAP repository
- Changes to the registry file (registry.xml)
- Configuration of a secure gateway to connect to a local LDAP repository and the container. (You need an existing app on Bluemix® for this step.)

LDAP repository

Create users and groups in the LDAP repository. For groups, authorization is enforced based on user membership.

Registry file

1. Open the **registry.xml** and find the `basicRegistry` element. Replace the `basicRegistry` element with code that is similar to the following snippet:

```
<ldapRegistry
   id="ldap"
   host="1.234.567.8910" port="1234" ignoreCase="true"
   baseDN="dc=worklight,dc=com"
   ldapType="Custom"
   sslEnabled="false"
   bindDN="uid=admin,ou=system"
   bindPassword="secret">
   <customFilters userFilter="(&amp;(uid=%v)(objectclass=inetOrgPerson))"
   groupFilter="(&amp;(member=uid=%v)(objectclass=groupOfNames))"
   userIdMap="*:uid"
   groupIdMap="*:cn"
   groupMemberIdMap="groupOfNames:member"/>
</ldapRegistry>
```

Entry	Description
`host` and `port`	Host name (IP address) and port number of your local LDAP server.
`baseDN`	The domain name (DN) in LDAP that captures all details about a specific organization.
`bindDN="uid=admin,ou=system"`	Binding details of the LDAP server. For example, the default values for an Apache Directory Service would be `uid=admin,ou=system`.
`bindPassword="secret"`	Binding password for the LDAP server. For example, the default value for an Apache Directory Service is `secret`.
`<customFilters userFilter="(&(uid=%v)(objectclass=inetOrgPerson))" groupFilter="(&(member=uid=%v)(objectclass=groupOfNames))" userIdMap="*:uid" groupIdMap="*:cn" groupMemberIdMap="groupOfNames:member"/>`	The custom filters that are used for querying the directory service (such as Apache) during authentication and authorization.

2. Ensure that the following features are enabled for `appSecurity-2.0` and `ldapRegistry-3.0`:

```
<featureManager>
   <feature>appSecurity-2.0</feature>
   <feature>ldapRegistry-3.0</feature>
</featureManager>
```

   For details about configuring various LDAP server repositories, see the WebSphere Application Server Liberty Knowledge Center (http://www-01.ibm.com/support/knowledgecenter/was_beta_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_sec_ldap.html).

Secure gateway

To configure a secure gateway connection to your LDAP server, you must create an instance of the Secure Gateway service on Bluemix and then obtain the IP information for the LDAP registry. You need your local LDAP host name and port number for this task.

1. Log on to Bluemix and navigate to **Catalog, Category > Integration**, and then click **Secure Gateway**.
2. Under Add Service, select an app and then click **Create**. Now the service is bound to your app.
3. Go to the Bluemix dashboard for the app, click on the **Secure Gateway** service instance, and then click **Add Gateway**.
4. Name the gateway and click **Add Destinations** and enter the name, IP address, and port for your local LDAP server.
5. Follow the prompts to complete the connection. To see the destination initialized, navigate to the Destination screen of the LDAP gateway service.
6. To obtain the host and port information that you need, click the Information icon on the LDAP gateway service instance (located on the Secure Gateway dashboard). The details displayed are an alias to your local LDAP server.
7. Capture the **Destination ID** and **Cloud Host : Port** values. Go to the registry.xml file and add these values, replacing any existing values. See the following example of an updated code snippet in the registry.xml file:

```
<ldapRegistry
   id="ldap"
   host="cap-sg-prd-5.integration.ibmcloud.com" port="15163" ignoreCase="true"
   baseDN="dc=worklight,dc=com"
   ldapType="Custom"
   sslEnabled="false"
   bindDN="uid=admin,ou=system"
   bindPassword="secret">
   <customFilters userFilter="(&amp;(uid=%v)(objectclass=inetOrgPerson))"
   groupFilter="(&amp;(member=uid=%v)(objectclass=groupOfNames))"
   userIdMap="*:uid"
   groupIdMap="*:cn"
   groupMemberIdMap="groupOfNames:member"/>
</ldapRegistry>
```

Configuring apps to work with LDAP

Configure MobileFirst mobile apps to work with an external LDAP registry.
The configuration process includes the following step: Configuring a secure gateway to connect to a local LDAP repository and the container. (You need an existing app on Bluemix for this step.)

To configure a secure gateway connection to your LDAP server, you must create an instance of the Secure Gateway service on Bluemix and then obtain the IP information for the LDAP registry. You need your local LDAP host name and port number for this step.

1. Log on to Bluemix and navigate to **Catalog, Category > Integration**, and then click **Secure Gateway**.
2. Under Add Service, select an app and then click **Create**. Now the service is bound to your app.
3. Go to the Bluemix dashboard for the app, click on the **Secure Gateway** service instance, and then click **Add Gateway**.
4. Name the gateway and click **Add Destinations** and enter the name, IP address, and port for your local LDAP server.
5. Follow the prompts to complete the connection. To see the destination initialized, navigate to the Destination screen of the LDAP gateway service.
6. To obtain the host and port information that you need, click the Information icon on the LDAP gateway service instance (located on the Secure Gateway dashboard). The details displayed are an alias to your local LDAP server.
7. Capture the **Destination ID** and **Cloud Host : Port** values. Provide these values for the LDAP login module. Results The

communication between the MobileFirst app on Bluemix with your local LDAP server is established. The authentication and authorization from the Bluemix app is validated against your local LDAP server.