Multi-Factor Authentication (2FA) with the Gateway KEE _ KEA

This document describes how to enable two factor authentication (2FA) between your client applications and backend servers using the Gateway.

Note: This document assumes that you are familiar with two factor authentication. For more information, see Multi-factor authentication, RSA SecurID, and RSA SecurID Hardware Token.

Kaazing 2FA provides the following benefits:

- Maintains security even if a username and password are compromised.
- No change to backend server configuration. Kaazing 2FA is a drop-in solution.
- Kaazing 2FA is consistent across different backend server platforms. It provides an agnostic 2FA solution.

This document contains the following sections:

- 2FA Topology and Components
  - Common Enterprise Topology
  - Topology Components
- Security in the 2FA Session
- Connection Process
- Deploying 2FA with the Gateway
  - Set Up the Authorization Gateway
  - Set Up the Discovery Service
  - Install and Configure the Kaazing Secure Connector
  - Test 2FA Deployment
- Discovery Service API/SPI
  - DiscoveryService.java

DiscoveryServiceSpi.java

DiscoveryServiceListener.java

DiscoveryServiceFactory.java

Endpoint.java

Overview

Single factor authentication follows a process where a user accesses a backend server in the WAN using client software:

1. The administrator enters the hostname or IP address of backend server to connect to the server.
2. The client software prompts for a username and password for authentication.
3. The user enters the username and password
4. The backend server authenticates the user.
5. The user accesses the backend server via the client software. For example, a database administrator (DBA) uses a SQL client to access a database server on the WAN.

Two factor authentication follows a more secure process where the user runs a local Kaazing client application, Kaazing Secure Connector, that connects to an Authorization Gateway protecting the backend server:

1. In Kaazing Secure Connector, the user supplies an URL for an Discover Service.
2. The Authorization Gateway requires that the user provide a username and password (first factor) *and* a RSA SecurID Hardware Token (second factor), as shown in the following picture.

3. The Authorization Gateway validates both factors.
4. Once the user has supplied both factors successfully, the user can use their previous client software to connect the the backend server, only this time the user will enter the URL of the Authorization Gateway instead of the backend server.

2FA Topology and Components

You can deploy 2FA in a number of ways, with clients on the same network or on different networks. For the purposes of this document, we will document an enterprise topology where a database administrator (DBA) is on the same WAN as the database server.

Common Enterprise Topology

The following diagram shows an enterprise 2FA topology using the Kaazing Secure Connector application, a Gateway acting as a Discovery Service, and an Authorization Gateway protecting the database server.

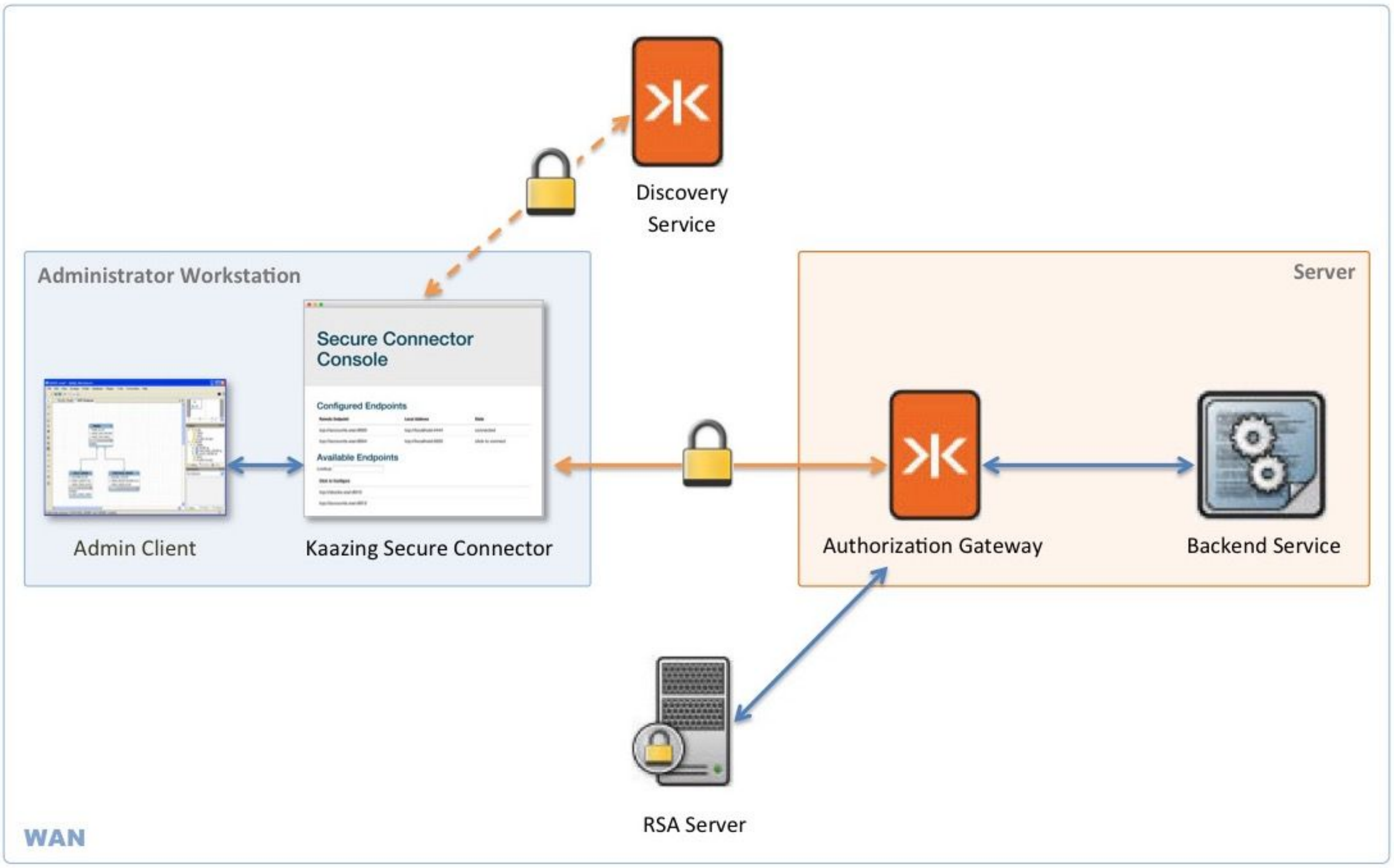

Figure 1: Enterprise 2FA Deployment using the Gateway

Topology Components

The following table lists the components involved in an enterprise 2FA deployment.

Component	**Description**
Administrator application	The application the administrator uses to access the backend server and its resources. For example, MySQL Workbench.
Backend server	The server running the protected resource the administrator wants to access.
Kaazing Secure Connector (KSC)	A desktop application running on the administrator's workstation that enables the administrator to authenticate using 2FA and, once successful, KSC acts

	as a proxy to connect the administrator application to the backend server.
Discovery Service	A Gateway hosting a directory service that maps user friendly Authorization Gateway names to URIs. It responds to requests for Authorization Gateway URIs from the KSC.
Authorization Gateway	The Gateway that performs 2FA and proxies connections from the KSC to the backend server.
Login Modules	A Java interface that provides authentication. Login modules authenticate the administrator by verifying credentials such as usernames and password, and RSA SecureIDs.
First factor authentication	Authentication of the first credentials provided by the administrator. For example, authenticating a username and password against a file or database record of user accounts.
Second factor authentication	Authentication of the second credentials provided by the administrator. For example, authenticating a RSA SecureID against an RSA server.
RSA Authentication Manager	The platform behind RSA SecurID that allows for centralized management of the RSA SecurID environment including authentication methods, users, applications, and agents across multiple physical sites. For information on setting up Authentication Agents, see the RSA documentation: RSA Authentication Manager, and RSA Solutions Training videos.

Security in the 2FA Session

Security and authentication takes place at a number of points in the 2FA deployment:

- All connections made by the Kaazing Secure Connector are end-to-end encrypted using TLS. The connection between the Kaazing Secure Connector and the Discovery service is encrypted using TLS, either by HTTPS or WSS. The connection between the Kaazing Secure Connector and the Authorization Gateway is encrypted using TLS via WSS.

- An encrypted connection between any DBA client application and the backend server is required by the Kaazing 2FA deployment.
- Authorization Gateway requires 2 factor authentication from the Kaazing Secure Connector:
  - Username and password.
  - pin/password such as RSA SecureID verified against a RSA SecureID server.
- Any backend TCP connectivity through the Authorization Gateway to the protected server must pass an IP whitelist.

Connection Process

The following process describes how a database administrator (DBA) using MySQL Workbench to connect to a database server is authenticated using Kaazing 2FA:

1. The DBA opens the Kaazing Secure Connector (KSC).
2. The Kaazing Secure Connector automatically creates a secure connection with the Discovery Service Gateway and receives the list of available endpoints (such as databases) to which the user can connect.
3. In KSC, the DBA selects an Authorization Gateway as a remote endpoint.
4. KSC connects to the Authorization Gateway via TLS, requesting that the Authorization Gateway present a valid TLS certificate for its hostname.
5. The Authorization Gateway sends its TLS certificate to KSC.
6. KSC validates the TLS certificate and requests access to the database from the database server. A local address, such as `localhost:4444`, is requested at this step. This local address is used to communicate with the Authorization Gateway for that specific database.
7. The Authorization Gateway challenges the KSC for the first authentication factor, a username and password.
8. The DBA enters the username and password into KSC and presses **Enter**.

9. The Authorization Gateway authenticates the username and password and then challenges the KSC with the second authentication factor, the RSA SecureID. The challenge is displayed in KSC.
10. The DBA uses his RSA SecurID device to obtain an RSA code, enters the code in KSC, and presses **Enter**.
11. The Authorization Gateway verifies the RSA code against the RSA server in the WAN.
12. The Authorization Gateway proxies the connection from Kaazing Secure Connector to the backend server. (Note, the connection between Kaazing Secure Connector and the Authorization Gateway remains encrypted by TLS.)
13. The DBA opens MySQL Workbench and opens the screen to connect to a remote database server.
14. The DBA enters localhost IP address:port `127.0.0.1:4444` for KSC, and clicks **Connect**.
15. MySQL Workbench connects to KSC via the localhost IP address:port, the KSC proxies that connection to the Authorization Gateway via its valid TLS and 2FA secured connection, and then the Authorization Gateway proxies the connection to the database server.
16. The DBA uses MySQL Workbench with the remote database server.

Deploying 2FA with the Gateway

Deploying 2FA involves minimal configuration of the Kaazing Secure Connector or the Gateways used for discovery and authentication. The Kaazing 2FA packaging includes login modules for various authentication sources, including RSA SecureID.

The following deployment steps deploy the server components first, followed by the client components.

To deploy 2FA, follow this checklist:

#	Step	Section
1	Set up the Authorization Gateway.	Set Up the Authorization Gateway
2	Set up the Discovery service on a Gateway.	Set Up the Discovery Service
3	Install Kaazing Secure Connector on the	Install Kaazing Secure Connector

	administrator’s computer.	
4	Test your 2FA deployment by authenticating using Kaazing Secure Connector, and then by logging into the backend server using the 3rd party client software.	Test 2FA Deployment

Set Up the Authorization Gateway

The Authorization Gateway performs the 2FA for the user via Kaazing Secure Connector. Once authentication is successful, the Authorization Gateway proxies incoming connections from the Kaazing Secure Connector to the backend server. Configuring the Authorization Gateway involves setting up its 2FA and proxy service.

Notes:

- Setting up the Authorization Gateway is very simple. The Gateway distribution comes with YAML configuration files that are easy to update, and several login modules to accommodate different authentication services.
- The 2FA Gateway distribution ships with a configuration and files for testing purposes. The examples used in the following procedure include hostnames and file names used in testing. When you deploy Kaazing 2FA you will replace these names with live, production names.

To configure the Authorization Gateway, do the following:

1. Download and install the Kaazing WebSocket Gateway - 2FA Edition on the server.
2. In the distribution, open the Authorization Gateway YAML configuration file, *GATEWAY_HOME/conf/authorization-gateway-config.yml*. In the file you will see hostnames and IP addresses used for testing that you can use or replace with hostnames and IP addresses from your WAN. To use the testing hostnames and IP addresses, you will need to update the DNS service used by the hosts in your WAN. Here is what the YAML file looks like:

```
# The proxy service that accepts connections from the Kaazing Secure Connector
# and proxies the connections to the desired endpoints
wsProxy:
    # The URI entered in the Kaazing Secure Connector to connect to
    # the Authorization Gateway
    - accept: auth1.2fa.kaazing.test:8443
      # Bind the accept URI to the local server address and port
      bind: 127.0.0.1:8443
      # Connect Kaazing Secure Connector connections to the desired endpoint
      connect:
          - localhost:3306
          - 192.168.99.100:8000

# The proxy service that connects to the backend server
tcpProxy:
    # The URI the tcpProxy service listens on.
    # It contains its local hostname and the port number of the service
    - accept: db1-app.2fa.kaazing.test:3306
      # Bind the connection to the local server IP address
      bind: 127.0.0.1:4321
      # IP addresses of hosts that may connect via the proxy service
      ipWhitelist:
          - 127.0.0.1
      # The IP address of the backend server and port for the backend service
      connect:
          - 192.168.99.100:8000
    # Additional accept if the Authorization Gateway is being used
    # by another application server
    - accept: db2-app.2fa.kaazing.test:3307
      # IP addresses of hosts that may connect to the proxy service
      ipWhitelist:
          - 192.168.99.0/24
      # The IP address of the backend server and port for the backend service
      connect:
          - 192.168.99.100:9000
# keystore containing the TLS certificate sent to the Kaazing Secure Connector
keystore:
    type: JCEKS
    # The file containing the certificates
    file: keystore.db
    # Password file for the keystore
    passwordFile: keystore.pw
```

Note: The connect URI for a host name in a service (`wsProxy` or `tcpProxy`) must match the `backend` URI for the same host name in the mappings.yml file used by the Discovery Service. This is discussed in Set Up the Discovery Service below.

3. For production, modify the WebSocket proxy service. The WebSocket proxy service, `wsProxy`, accepts connections from the Kaazing Secure Connector, applies 2FA, and proxies traffic to the back-end server. `wsProxy` uses WebSocket Secure (WSS) and for the connection from Kaazing Secure Connector and therefore it provides a TLS certificate from its keystore to Kaazing Secure Connector to establish the TLS connection. The Kaazing 2FA Gateway distribution contains certificates you can use for testing under the hostname `2fa.kaazing.t est`.

   When you move to production you will need to update the hostnames and IP addresses in `wsProxy` for your production hosts. In addition, you will need to add a TLS certificate to the keystore for the production hostname that the Authorization Gateway uses in its `accept`. For more information, see Secure the Gateway Using Trusted Certificates.
4. For production, modify the TCP proxy service. The TCP proxy service, `tcpProxy`, proxies connections to the backend server. When you move to production you will need to update the hostnames and IP addresses in `tcpProxy` for your production hosts.
5. Save the YAML file.
6. Modify the login modules and related files used to perform 2FA. The `wsProxy` service will use the login modules located and related files located in `*GATEWAY_HOME*/lib/ext`. For example, a login module to authenticate the username and password is located in that folder along with a corresponding file containing usernames and passwords. A login module named RsaLoginModule.java is used for the RSASecureID authentication and calls a corresponding file, **rsa.property.file**, located in the same folder. You must update **rsa.property.file** with the RSA server information, such as the IP address.

7. Start the Authorization Gateway by running the startup script, **authorization-gateway.start**, in the `GATEWAY_HOME/bin` folder.

Set Up the Discovery Service

The Discovery Service locates the Authorization Gateways according to the backend servers they proxy. When the Kaazing Secure Connector is used for the first time, the administrator will specify the Discovery Service URL to access, and the Kaazing Secure Connector will contact the Discovery Service to locate the Authorization Gateway for a backend server. To discover Authorization Gateways, the Discovery Service uses a local discovery file that maps user-friendly names to backend server addresses. The Discovery Service then returns the Authorization Gateway URI to the Kaazing Secure Connector.

Configuring the Gateway for the Discovery Service involves:

- Configuring a directory service on the Gateway.
- Configuring a keystore for the TLS certificate the Discovery Service will present to the Kaazing Secure Connector.
- Populating a mapping file for the backend servers.

Note: The Discovery Service includes an API to allow you to configure your own database storage for the backend server mappings data. It is also an SPI that you can extend. For more information, see Discovery Service API/SPI.

To configure the Discovery Service, do the following:

1. Download and install the Kaazing WebSocket Gateway - 2FA Edition on the server.
2. In the distribution, open the Discovery Service YAML configuration file, *GATEWAY_HOME/conf/discovery-service-config.yml*.
3. Modify the configuration file. The configuration file contains a directory service and a keystore configuration.

```
directory:
    # the URI the Kaazing Secure Connector uses to connect to the
    # Discovery Service.
    acceptURL: discovery.2fa.kaazing.test:8443
    # The local IP address of the Gateway
    bind: 127.0.0.1:8443
# keystore containing the TLS certificate sent to the Kaazing Secure Connector
keystore:
    type: JCEKS
    # The file containing the certificates
    file: keystore.db
    # Password file for the keystore
    passwordFile: keystore.pw
```

The Discovery Service provides a TLS certificate from its keystore to Kaazing Secure Connector to establish the TLS connection. The Kaazing 2FA Gateway distribution contains certificates you can use for testing under the hostname `2fa.kaazing.test`.

When you move to production you will need to update the hostnames and IP addresses in `directory` for your production hosts. In addition, you will need to add a TLS certificate to the keystore for the production hostname that the Discovery Service Gateway uses. For more information, see Secure the Gateway Using Trusted Certificates.

4. Save the configuration file.
5. Open the YAML mappings file, **mappings.yml**, that maps user-friendly names to Authorization Gateway URIs.

```
endpoints:
- name: Auth 1
  proxy: 'auth1.2fa.kaazing.test:8443'
  backend: 'localhost:3306'
- proxy: 'auth1.2fa.kaazing.test:8443'
  backend: 'localhost:3307'
- proxy: 'auth1.2fa.kaazing.test:8443'
  backend: 'localhost:3308'
- name: Auth 2
  proxy: 'auth2.2fa.kaazing.test:8443'
  backend: 'localhost:3309'
- proxy: 'auth2.2fa.kaazing.test:8443'
  backend: 'localhost:3310'
```

Notes:

- The `name` value is the user-friendly name that will appear in the Kaazing Secure Connector, such as **Stock Database**.
- The `proxy` value is the URI that the Kaazing Secure Connector will use to locate the Authorization Gateway. It is the same URI as the `accept` value in the Authorization Gateway's configuration YAML file. The `backend` value is the IP address and port of the backend server. The value must match the `connect` value for the same hostname in the Authorization Gateway config file. For example:

  authorization-gateway-config.yml:
  ```
  wsProxy:
      - accept: auth1.2fa.kaazing.test:8443
        bind: 127.0.0.1:8443
        connect:
            - 192.168.99.100:8000
  ```

  mappings.yml:
  ```
  - name: Auth 1
    proxy: 'auth1.2fa.kaazing.test:8443'
    backend: '192.168.99.100:8000'
  ```

- When you move to production you will need to update these values.
- Always synchronize the `proxy` and `backend` values in the **mappings.yml** file of the Discovery Service with the `accept` and `backend` values for the same hostname in the Authorization Gateway config file.

6. Save the configuration file.
7. Start the Discovery Service by running the startup script, **discovery-gateway.start**, in the `GATEWAY_HOME/bin` folder.

Install and Configure the Kaazing Secure Connector

The Kaazing Secure Connector is the administrator’s interface to Kaazing 2FA. The administrator uses Kaazing Secure Connector to discover Authorization Gateways for the backend servers he wants to connect to, and he uses the Kaazing Secure Connector to enter in his credentials for 2FA.

Configuring the Kaazing Secure Connector involves:

- Specifying a remote endpoint for the connection. The remote endpoint is the user-friendly name for an Authorization Gateway. The Discovery Service maps the name to the URI of an Authorization Gateway.
- Configuring an HTTPS/WSS connection from Kaazing Secure Connector to the Discovery Service.
- Saving the administrator’s configuration after the initial run of Kaazing Secure Connector.

To configure the Kaazing Secure Connector, do the following:

1. Ensure that the workstation is connected to the company WAN. If you are working remotely, ensure that you have a VPN connection to the WAN.
2. Download and install the Kaazing WebSocket Gateway - 2FA Edition on the administrator's workstation. To install, run the KSC installer inside the distribution. On Windows, the Kaazing Secure Connector is installed in `C:\Users\<USER_NAME>\AppData\Local\KSC\` . On Macintosh, it is installed in `Applications`.

3. Run the KSC app to connect to the Authorization Gateway. The app is displayed.

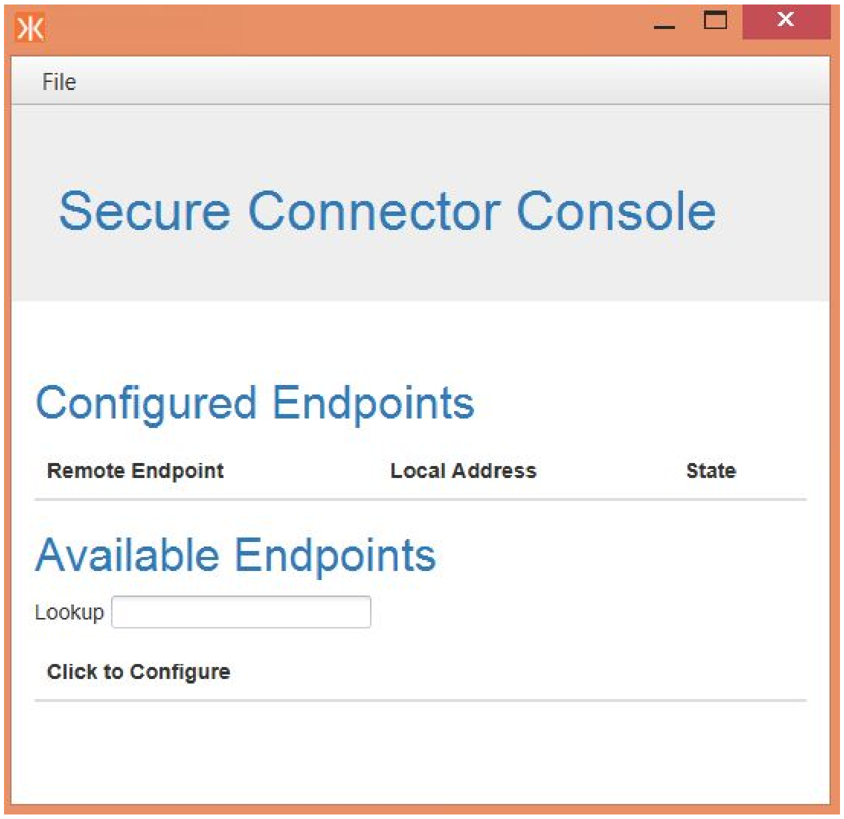

Figure 2: Kaazing Secure Connector

4. Click **File**, and then click **Preferences**. The Configuration Menu appears.
5. In **Truststore**, click **Browse**. Locate the truststore for the KSC. By default, the truststore is located in the same folder as the app, and you can simply enter **truststore.db**.
6. In Discovery Service URL, enter `https://` followed by the `acceptURL` in the Discovery Service configuration file's `directory` service. For example, `https://discovery.2fa.kaazing.test:8443` .
7. In **Time to poll the Discovery Service**, enter the number of minutes you would like KSC to poll the Discovery Service for Authorization Gateways.
8. Click **Export configuration to file**, and then click **Save Configuration**. The configuration is saved in the app folder.

9. Modify the truststore. The truststore contains certificates that the Kaazing Secure Connector trusts. The truststore should contain the certificates for the hostname of the Discovery Service the administrator wants to connect with. The default distribution of Kaazing 2FA includes a self-signed certificate for the name `*.2fa.kaazing.test`. You can use this certificate for your testing phase, but when you move to production, a new, trusted certificate is required. For more information on modifying the truststore, see Secure Gateway-to-Server Connections.
10. In KSC, under **Click to Configure**, click the name of an Authorization Gateway.
11. Follow the authorization prompts to perform 2FA.

Test 2FA Deployment

To test your 2FA deployment, do the following:

1. Ensure that the following components are running and their hostnames can be resolved using DNS:
    a. RSA Server. Also, ensure it is running and is configured to verify codes.
    b. Backend server.
    c. Authorization Gateway.
    d. Discovery Service Gateway.
2. Run the Kaazing Secure Connector and follow the authorization prompts to perform 2FA.
3. On the administrator workstation, open the client you typically use to connect to the backend server, for example MySQL Workbench.
4. In the client, create a connection using the localhost address, 127.0.0.1:4444. The client connects through the KSC to the backend server.

Discovery Service API/SPI

You can build your own program to modify, update or extend the Discovery Service. The Discovery Service API/SPI consists of 3 interfaces and 2 classes.

DiscoveryService.java

```
/**
 * Describes a service that can discover end-points that an application can use to
connect to.
 */
public interface DiscoveryService {

    /**
     * Gets the end-points it discovered.
     *
     * @return the discovered end-points
     */
    List<Endpoint> getEndpoints();

    /**
     * Registers a new {@code DiscoveryServiceListener} to listen for end-point events.
     *
     * @param listener the instance to register
     */
    void addListener(DiscoveryServiceListener listener);

}
```

DiscoveryServiceSpi.java

```
/**
 * The Service Provider Interface for the {@link DiscoveryService} class.
 *
 * <p>Providers wanting to implement a new discovery service type must implement this
interface.
```

```
 */
public interface DiscoveryServiceSpi {

    /**
     * Returns the type of discovery service it can create.
     *
     * @return the {@code DicoveryService} type
     */
    String getDiscoveryServiceType();

    /**
     * Creates a new {@code DiscoveryService} instance, using the provided {@code
options}.
     *
     * @param options the information needed to properly configure the service of this
type
     * @return a new {@code DiscoveryService} instance
     */
    DiscoveryService createDiscoveryService(Map<String, ?> options);
}
```

DiscoveryServiceListener.java

```
/**
 * Describes a listener that can react to end-point related events.
 */
public interface DiscoveryServiceListener {

    /**
     * Defines the behavior of the listener when the {@code endpoint} provided as input
is removed from the list of known end-points.
     *
     * @param endpoint the removed end-point
     */
    public void endpointRemoved(Endpoint endpoint);

    /**
     * Defines the behavior of the listener when the a new {@code endpoint} is found by
the {@code DiscoveryService}.
```

```
     *
     * @param endpoint the newly discovered end-point
     */
    public void endpointAdded(Endpoint endpoint);
}
```

DiscoveryServiceFactory.java

```
/**

* Creates a new {@code DiscoveryService} instance of the specified {@code type},

configured using the {@code options} map.

*

* @param type the type of {@code DiscoveryService} to create

* @param options a {@code Map} containing the information needed by the service to

start properly

* @return a new {@code DiscoveryService} instance

*/

public static DiscoveryService createDiscoveryService(String type, Map<String, ?>

options) {...}
```

Endpoint.java

This is a POJO describing an communication endpoint. It contains three fields:

- Name
- Proxy address
- Backend address