Quarkus - Built-In Authentication Support

This document describes the Quarkus built-in authentication mechanisms for HTTP based FORM, BASIC and Mutual TLS authentication as well as the proactive authentication.

Basic Authentication

To enable basic authentication set `quarkus.http.auth.basic=true`. You must also have at least one extension installed that provides a username/password based `IdentityProvider`, such as Elytron JDBC.

Please see Security Identity Providers for more information.

Form Based Authentication

Quarkus provides form based authentication that works in a similar manner to traditional Servlet form based auth. Unlike traditional form authentication, the authenticated user is not stored in an HTTP session, as Quarkus does not provide clustered HTTP session support. Instead the authentication information is stored in an encrypted cookie, which can be read by all members of the cluster (provided they all share the same encryption key).

The encryption key can be set using the `quarkus.http.auth.session.encryption-key` property, and it must be at least 16 characters long. This key is hashed using SHA-256 and the resulting digest is used as a key for AES-256 encryption of the cookie value. This cookie contains an expiry time as part of the encrypted value, so all nodes in the cluster must have their clocks synchronized. At one minute intervals a new cookie will be generated with an updated expiry time if the session is in use.

The following properties can be used to configure form based auth:

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
🔒 `quarkus.http.auth.form.enabled` If form authentication is enabled	boolean	`false`
🔒 `quarkus.http.auth.form.login-page` The login page	string	`/login.html`

🔒 `quarkus.http.auth.form.error-page` The error page	string	`/error .html`
🔒 `quarkus.http.auth.form.landing-page` The landing page to redirect to if there is no saved page to redirect back to	string	`/index .html`
🔒 `quarkus.http.auth.form.redirect-after-login` Option to disable redirect to landingPage if there is no saved page to redirect back to. Form Auth POST is followed by redirect to landingPage by default.	boolean	`true`
🔒 `quarkus.http.auth.form.timeout` The inactivity (idle) timeout When inactivity timeout is reached, cookie is not renewed and a new login is enforced.	Duration ❓	`PT30M`
🔒 `quarkus.http.auth.form.new-cookie-interval` How old a cookie can get before it will be replaced with a new cookie with an updated timeout, also referred to as "renewal-timeout". Note that smaller values will result in slightly more server load (as new encrypted cookies will be generated more often), however larger values affect the inactivity timeout as the timeout is set when a cookie is generated. For example if this is set to 10 minutes, and the inactivity timeout is 30m, if a users last request is when the cookie is 9m old then the actual timeout will happen 21m after the last request, as the timeout is only refreshed when a new cookie is generated. In other words no timeout is tracked on the server side; the timestamp is encoded and encrypted in the cookie itself and it is decrypted and parsed with each request.	Duration ❓	`PT1M`
🔒 `quarkus.http.auth.form.cookie-name` The cookie that is used to store the persistent session	string	`quarku s- creden tial`

ℹ️ *About the Duration format*

The format for durations uses the standard `java.time.Duration` format. You can learn more about it in the Duration#parse() javadoc.

You can also provide duration values starting with a number. In this case, if the value consists only of a number, the converter treats the value as seconds. Otherwise, `PT` is implicitly prepended to the value to obtain a standard `java.time.Duration` format.

Mutual TLS Authentication

Quarkus provides mTLS authentication so that you can authenticate users based on their X.509 certificates.

To use this authentication method, you should first enable SSL for your application. For more details, check the Supporting secure connections with SSL guide.

Once your application is accepting secure connections, the next step is to configure a `quarkus.http.ssl.certificate.trust-store-file` holding all the certificates that your application should trust as well as how your application should ask for certificates when a client (e.g.: browser or another service) tries to access one of its protected resources.

```
quarkus.http.ssl.certificate.key-store-file=server-keystore.jks
①
quarkus.http.ssl.certificate.key-store-
password=the_key_store_secret
quarkus.http.ssl.certificate.trust-store-file=server-truststore.jks
②
quarkus.http.ssl.certificate.trust-store-
password=the_trust_store_secret
quarkus.http.ssl.client-auth=required
③

quarkus.http.auth.permission.default.paths=/*
④
quarkus.http.auth.permission.default.policy=authenticated
```

① Configures a key store where the server's private key is located.

② Configures a trust store from where the trusted certificates are going to be loaded from.

③ Defines that the server should **always** ask certificates from clients. You can relax this behavior by using `REQUEST` so that the server should still accept requests without a certificate. Useful when you are also supporting authentication methods other than mTLS.

④ Defines a policy where only authenticated users should have access to resources from your application.

Once the incoming request matches a valid certificate in the truststore, your application should be able to obtain the subject by just injecting a `SecurityIdentity` as follows:

Obtaining the subject

```
@Inject
SecurityIdentity identity;

@GET
@Produces(MediaType.TEXT_PLAIN)
public String hello() {
    return String.format("Hello, %s",
identity.getPrincipal().getName());
}
```

You should also be able to get the certificate as follows:

Obtaining the certificate

```
import java.security.cert.X509Certificate;
import io.quarkus.security.credential.CertificateCredential;

CertificateCredential credential =
identity.getCredential(CertificateCredential.class);
X509Certificate certificate = credential.getCertificate();
```

Authorization

The information from the client certificate can be used to enhance Quarkus `SecurityIdentity`. For example, one can add new roles after checking a client certificate subject name, etc. Please see the SecurityIdentity Customization section for more information about customizing Quarkus `SecurityIdentity`.

Proactive Authentication

By default Quarkus does what we call proactive authentication. This means that if an incoming request has a credential then that request will always be authenticated (even if the target page does not require authentication).

This means that requests with an invalid credential will always be rejected, even for public pages. You can change this behavior and only authenticate when required by setting `quarkus.http.auth.proactive=false`.

References

- Quarkus Security