RAKSHA.IO

At Raksha.io we develop applications for web security environments.

This includes the basics of PKI and that is X.509 authentications.

The most common use of X.509 certificate authentication is in verifying the identity of a server when using **SSL**/TLS 2.0, most commonly when using HTTPS from a browser. The browser will automatically check that the certificate presented by a server has been issued (ie digitally signed) by one of a list of trusted certificate authorities which it maintains.

Using TLS / SSL with “mutual authentication”; the server will then request a valid certificate from the client as part of the SSL handshake. The server will authenticate the client by checking that its certificate is signed by an

acceptable authority. If a valid certificate has been provided, it can be obtained through the callbacks in an application. Raksha.io Security X.509 module extracts the certificate using a filter. It maps the certificate to an application user and loads that user's set of granted authorities for use with the standard Security infrastructure.

Raksha.io has CA and SUBCA engines that creates master certificates such as :

1. Domain Validated (DV)
2. Organization Validated (OV)
3. Extended Validation (EV)

on request from our clients. Clients than use TLS/SSL certificates to establish an encrypted connection between a website/server and a browser with what's known as an "SSL handshake." For visitors to client website, the process is invisible—and instantaneous.

Raksha.io has OAuth, or open authorization token converter engine.

OAuth is about authorization and not authentication. Authentication is proving your identity so that you can gain entry to an application or system. Authorization is asking for and receiving permission to access specific data, features, or areas of that application or system. The "auth" part of "OAuth" stands for authorization, not authentication. It does not pass credentials between users and service providers and instead only authorizes secure access in the form of temporary tokens. OAuth uses access tokens to grant temporary access to third parties. The tokens are typically used short term, but some can grant recurring access.

OAuth authorizes access process assumes that a user has already signed in to a website or service, and now the user wants to initiate a transaction that needs access to a third-party site or service. The authorization process follows the steps below:

1. The application requests authorization to access a protected service provider.
2. The user authorizes the request.
3. The application provides proof of user authorization to the service provider in exchange for an access token.
4. The user is redirected to the service provider to provide permission.
5. Once approved by the user, the application obtains the access token.
6. The application requests access to the protected resources from the service provider.

The greatest benefit of OAuth for a website, such as a news, community, or e-commerce site, is that authenticated website access can be extended to an unlimited number of additional users without those users creating new accounts requiring an email address and a new password. Open authorization reduces friction for both parties. Websites can scale, and users do not have to create yet another online account.

OAuth artifacts:

Resource Server: The server hosting user-owned resources that are protected by OAuth2. The resource server validates the access-token and serves the protected resources.

ii) Resource Owner: Typically, the user of the application is the resource owner. The resource owner has the ability to grant or deny access to their own data hosted on the resource server.

iii) Authorization Server: The authorization server gets consent from the resource owner and issues access tokens to clients for accessing protected resources hosted by a resource server.

iv) Client: An application making API requests to perform actions on protected resources on behalf of the resource owner. Before it may do so, it must be authorized by the resource owner, and the authorization must be validated by resource server/authorization

server. OAuth2 defines two client types, based on their ability to authenticate securely with the authorization server (i.e., ability to maintain the confidentiality of their client credentials):

a)Confidential: Clients capable of maintaining the confidentiality of their credentials. Confidential clients are implemented on a secure server with restricted access to the client credentials (e.g., a web application running on a web server).

b) Public: Clients incapable of maintaining the confidentiality of their credentials (e.g. an installed native application or a web browser-based application), and incapable of secure client authentication via any other means.

OAuth access tokens:

Access tokens are used in token-based authentication to allow an application to access an API. The application receives an

access token after a user successfully authenticates and authorizes access, then passes the access token as a credential when it calls the target API. The passed token informs the API that the bearer of the token has been authorized to access the API and perform specific actions specified by the **scope** that was granted during authorization.

In addition, if you have chosen to allow users to log in through an Identity Provider (IdP), such as Facebook, the IdP will issue its own access token to allow your application to call the IDP's API. For example, if your user authenticates using Facebook, the access token issued by Facebook can be used to call the Facebook Graph API. These tokens are controlled by the IdP and can be issued in any format. See Identity Provider Access Tokens for details.

Raksha.io implements JWKS & JWT token engines with endpoints.

JSON Web Token (JWT) access tokens conform to the JWT standard and contain information about an entity in the form of claims. They are self-contained therefore it is not necessary for the recipient to call a server to validate the token.

Access tokens issued for the Management API and access tokens issued for any custom API that you have registered with Auth0 follow the JWT standard, which means that their basic structure conforms to the typical JWT structure, and they contain standard JWT claims asserted about the token itself.

JWT sample:


```
{
 "iss": "https://my-domain.auth0.com/",
 "sub": "auth0|123456",
 "aud": [
  "https://example.com/health-api",
  "https://my-domain.auth0.com/userinfo"
 ],
 "azp": "my_client_id",
 "exp": 1311281970,
 "iat": 1311280970,
```

```
  "scope": "openid profile read:patients read:admin"
}
```

The token does not contain any information about the user except for the user ID (located in the `sub` claim). In many cases, you may find it useful to retrieve additional user information. You can do this by calling the userinfo API endpoint with the Access Token. Be sure that the API for which the Access Token is issued uses the **RS256** or **EC256** as signing algorithm.

Raksha.io implements SAML 2.0 assert conversions:

The OASIS Security Assertion Markup Language (SAML) standard defines an XML-based framework for describing and exchanging security information between on-line business partners. This security information is expressed in the form of portable SAML assertions that applications working across security domain boundaries

can trust. The OASIS SAML standard defines precise syntax and rules for requesting, creating, communicating, and using these SAML assertions.

The OASIS Security Services Technical Committee (SSTC) develops and maintains the SAML standard. The SSTC has produced this technical overview to assist those wanting to know more about SAML by explaining the business use cases it addresses, the high-level technical components that make up a SAML deployment, details of message exchanges for common use cases, and where to go for additional information.

UseCases of SAML:

1. Single Sign-On
2. Federated identity
3. Web services and other industry standards

Raksha.io demo is based on the above mentioned PKI, web security technologies and a brief on what demo did is :

1. Creating users, the details of this are given in Register Or Create User document.
2. After you have created the user the first time or reused a user you have already created.
3. The demo user logs in by signing in “Create a Demo User”. All fields must be filled with correct info.
4. After the “Register” button is activated, demo user is signed in and is shown in the “Generate Random User”

   Tab.
5. All the underlaying apps will have registered the demo user after the above action.
6. The first app on the site is : Token Exchange For User: (demo user id)

a. In this app there are 6 token exchange services and are self explanatory.

b. Each button is a service endpoint and shows the results in the pans below the buttons.

7. The second app SAML Assert For User: (demo user id)

a. On button activation shows the SAML 2.0 assert

8. The third app is JWKSets For User: (demo user id)

a. The JSON Web Key (JWKS) is a JavaScript Object Notation (JSON) data structure that represents a cryptographic key. Raksha.io has JWKS non-rotating type sets and implements two types including the x5c (X509

certificate chain) type and standard type and has the following attributes:

JSON Web Key (JWK) Format

b. "kty" (Key Type) Parameter
c. "use" (Public Key Use) Parameter
d. "key_ops" (Key Operations) Parameter
e. "alg" (Algorithm) Parameter
f. "kid" (Key ID) Parameter
g. "x5u" (X.509 URL) Parameter
h. "x5c" (X.509 Certificate Chain) Parameter

i. "x5t" (X.509 Certificate SHA-1 Thumbprint) Parameter
j. "x5t#S256" (X.509 Certificate SHA-256 Thumbprint) Parameter

9. The fourth app is JWT(standard token) and JWE (Encrypted token)
   a. The JWT includes x5c (X509 certificate chain)
   b. JWE includes x5c (X509 certificate chain)
10. The fifth app is OCSP for X509 certificate chain verification and validation.
11. The sixth app is a Enterprise usage app for money transactions documents transfer between Customers and their Banks.

The app is used by many EU enterprises.