**Compare and contrast the authentication types (BASIC, DIGEST, FORM, and CLIENT-CERT); describe how the type works; and given a scenario, select an appropriate type.**

A web client can authenticate a user to a web server using one of the following mechanisms:

* HTTP Basic Authentication
* HTTP Digest Authentication
* HTTPS Client Authentication
* Form Based Authentication

**HTTP Basic Authentication.**

HTTP Basic Authentication, which is based on a username and password, is the authentication mechanism defined in the HTTP/1.0 specification. A web server requests a web client to authenticate the user. As part of the request, the web server passes the realm (a string) in which the user is to be authenticated. The realm string of Basic Authentication does not have to reflect any particular security policy domain (confusingly also referred to as a realm). The web client obtains the username and the password from the user and transmits them to the web server. The web server then authenticates the user in the specified realm.

Basic Authentication is not a secure authentication protocol. User passwords are sent in simple base64 ENCODING (not ENCRYPTED !), and the target server is not authenticated. Additional protection can alleviate some of these concerns: a secure transport mechanism (HTTPS), or security at the network level (such as the IPSEC protocol or VPN strategies) is applied in some deployment scenarios.

<web-app>

<security-constraint>

<web-resource-collection>

<web-resource-name>User Auth</web-resource-name>

<url-pattern>/auth/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

<role-name>manager</role-name>

</auth-constraint>

</security-constraint>

<login-config>

<auth-method>BASIC</auth-method>

<realm-name>User Auth</realm-name>

</login-config>

<security-role>

<role-name>admin</role-name>

</security-role>

<security-role>

<role-name>manager</role-name>

</security-role>

</web-app>

**HTTP Digest Authentication.**

Like HTTP Basic Authentication, HTTP Digest Authentication authenticates a user based on a username and a password. However the authentication is performed by transmitting the password in an ENCRYPTED form which is much MORE SECURE than the simple base64 encoding used by Basic Authentication, e.g. HTTPS Client Authentication. As Digest Authentication is not currently in widespread use, servlet containers are encouraged but NOT REQUIRED to support it.

The advantage of this method is that the cleartext password is protected in transmission, it cannot be determined from the digest that is submitted by the client to the server.

Digested password authentication supports the concept of digesting user passwords. This causes the stored version of the passwords to be encoded in a form that is not easily reversible, but that the Web server can still utilize for authentication. From a user perspective, digest authentication acts almost identically to basic authentication in that it triggers a login dialog. The difference between basic and digest authentication is that on the network connection between the browser and the server, the password is encrypted, even on a non-SSL connection. In the server, the password can be stored in clear text or encrypted text, which is true for all login methods and is independent of the choice that the application deployer makes.

<web-app>

<security-constraint>

<web-resource-collection>

<web-resource-name>User Auth</web-resource-name>

<url-pattern>/auth/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

<role-name>manager</role-name>

</auth-constraint>

</security-constraint>

<login-config>

<auth-method>DIGEST</auth-method>

<realm-name>User Auth</realm-name>

</login-config>

<security-role>

<role-name>admin</role-name>

</security-role>

<security-role>

<role-name>manager</role-name>

</security-role>

</web-app>

**HTTPS Client Authentication.**

End user authentication using HTTPS (HTTP over SSL) is a strong authentication mechanism. This mechanism requires the user to possess a Public Key Certificate (PKC). Currently, PKCs are useful in e-commerce applications and also for a single-sign-on from within the browser. Servlet containers that are not J2EE technology compliant are not required to support the HTTPS protocol.

Client-certificate authentication is a more secure method of authentication than either BASIC or FORM authentication. It uses HTTP over SSL, in which the server and, optionally, the client authenticate one another with Public Key Certificates. Secure Sockets Layer (SSL) provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection. You can think of a public key certificate as the digital equivalent of a passport. It is issued by a trusted organization, which is called a certificate authority (CA), and provides identification for the bearer. If you specify client-certificate authentication, the Web server will authenticate the client using the client's X.509 certificate, a public key certificate that conforms to a standard that is defined by X.509 Public Key Infrastructure (PKI). Prior to running an application that uses SSL, you must configure SSL support on the server and set up the public key certificate.

<web-app>

<security-constraint>

<web-resource-collection>

<web-resource-name>User Auth</web-resource-name>

<url-pattern>/auth/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

<role-name>manager</role-name>

</auth-constraint>

</security-constraint>

<login-config>

<auth-method>CLIENT-CERT</auth-method>

<realm-name>User Auth</realm-name>

</login-config>

<security-role>

<role-name>admin</role-name>

</security-role>

<security-role>

<role-name>manager</role-name>

</security-role>

</web-app>

**Form Based Authentication.**

The look and feel of the 'login screen' cannot be varied using the web browser's built-in authentication mechanisms. This specification introduces a required form based authentication mechanism which allows a Developer to CONTROL the LOOK and FEEL of the login screens.

The web application deployment descriptor contains entries for a login form and error page. The login form must contain fields for entering a username and a password. These fields must be named j\_username andj\_password, respectively.

When a user attempts to access a protected web resource, the container checks the user's authentication. If the user is authenticated and possesses authority to access the resource, the requested web resource is activated and a reference to it is returned. If the user is not authenticated, all of the following steps occur:

1. The login form associated with the security constraint is sent to the client and the URL path triggering the authentication is stored by the container.
2. The user is asked to fill out the form, including the username and password fields.
3. The client posts the form back to the server.
4. The container attempts to authenticate the user using the information from the form.
5. If authentication fails, the error page is returned using either a forward or a redirect, and the status code of the response is set to 200.
6. If authentication succeeds, the authenticated user's principal is checked to see if it is in an authorized role for accessing the resource.
7. If the user is authorized, the client is redirected to the resource using the stored URL path.

The error page sent to a user that is not authenticated contains information about the failure.

Form Based Authentication has the same lack of security as Basic Authentication since the user password is transmitted as plain text and the target server is not authenticated. Again additional protection can alleviate some of these concerns: a secure transport mechanism (HTTPS), or security at the network level (such as the IPSEC protocol or VPN strategies) is applied in some deployment scenarios.

Form based login and URL based session tracking can be problematic to implement. Form based login should be used only when sessions are being maintained by cookies or by SSL session information.

In order for the authentication to proceed appropriately, the action of the login form must always be j\_security\_check. This restriction is made so that the login form will work no matter which resource it is for, and to avoid requiring the server to specify the action field of the outbound form.

Here is an example showing how the form should be coded into the HTML page:

<form method='post' action='j\_security\_check'>

<input type='text' name='j\_username'>

<input type='password' name='j\_password'>

</form>

<web-app>

<security-constraint>

<web-resource-collection>

<web-resource-name>User Auth</web-resource-name>

<url-pattern>/auth/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

<role-name>manager</role-name>

</auth-constraint>

</security-constraint>

<login-config>

<auth-method>FORM</auth-method>

<realm-name>User Auth</realm-name>

<form-login-config>

<form-login-page>login.jsp</form-login-page>

<form-error-page>error.jsp</form-error-page>

</form-login-config>

</login-config>

<security-role>

<role-name>admin</role-name>

</security-role>

<security-role>

<role-name>manager</role-name>

</security-role>

</web-app>

<https://www.youtube.com/watch?v=6uQ44YxUqUs>

<https://www.youtube.com/watch?v=CPbvxxslDTU>

<https://stackoverflow.com/questions/4113934/how-is-oauth-2-different-from-oauth-1>

<https://api2cart.com/api-technology/choosing-oauth-type-api/>