Security Mechanisms

- There are four security mechanisms detailed by the servlet specification: authentication, authorization (access to controlled resources), data integrity, and confidentiality (data privacy).
- Authentication is the process of proving you are who (or what) you claim to be.
- Authentication can be achieved through basic means (user IDs and passwords) versus complex means (digital certifi cates).
- Authorization is the process of ensuring that an authenticated party gains access only to the resources it is entitled to.
- In servlet spec terms, this process means identifying resources by their URLs (and the HTTP methods used to access them), and associating these with logical roles.
- It's the web server's job to supply some means or other for relating these logical roles to specifi c users or groups of users. There's no standard way to achieve this—it is server specifi c.
- Data integrity is the process of ensuring that any messages passed through a network have not been tampered with in transit.
- Data integrity very often involves encrypting the contents of a message. Integrity comes about because any tampering with an encrypted message will render the message impossible to decrypt.
- Confidentiality (data privacy) goes one step further than data integrity, by promising that the information in a message is available only to users authorized to see that information.

- The encryption process usually involves public and private keys

Deployment Descriptor Security Declarations

- Inside the root element `<web-app>`, there are three top-level elements devoted to security: `<security-constraint>`, `<login-config>`, and `<security-role>`.
- `<security-constraint>` is the biggest and most complex of these three.
- Its purpose is to associate resources (and HTTP methods executed on those resources) with logical roles for authorization, and also with guarantees on resource security in transit over a network.
- `<security-constraint>` has three main subelements—`<web-resource-collection>`, `<auth-constraint>`, and `<user-data-constraint>`.
- The first of these main subelements, `<web-resource-collection>`, defines the resources to be secured.
  - The first element inside `<web-resource-collection>` must be `<web-resource-name>`, whose value is a logical name to describe the group of resources protected.
  - Next, `<web-resource-collection>` defines the URL patterns to protect using one or more `<url-pattern>` subelements.
  - The value of a URL pattern is a path to a resource (or resources) within the web application.
  - Valid values for URL patterns are the same as for servlet mappings and filter mappings: exact path (/exactmatch), path prefix (longest match first) (/partial/*), extension matching (*.jsp), and default servlet (/). As for servlet mappings, web resource collection URL patterns types are processed in that order.

- Any resource can be protected—static or dynamic.
- `<web-resource-collection>` optionally contains `<http-method>` elements (zero to many).
- Use `<http-method>` to associate given HTTP methods with the resource protected. You may want to associate specific protection on certain resources just for the HTTP POST method, for example.

- The second subelement of `<security-constraint>` is `<auth-costraint>`.
- `<auth-constraint>` lists the named roles authorized to the resources defined in the web resource collection.
  - Each named role is placed in a subelement called `<role-name>`.
- An authority constraint with no value (e.g., `<auth-constraint />`) denotes that there should be no permitted access whatsoever to the resource.
- Identical URL patterns are protected, for the identical HTTP methods may appear in separate web resource collections—effectively protecting the same resource. In this case, all the role names specified in all the authority constraints are "added together"—a user in any one of those roles can access the resource with the given HTTP method.
- There's an exception to this authority constraint addition rule: If the no-value authority constraint (e.g., `<auth-constraint />`) is one of several authority constraints protecting the same resource, it *overrides everything else*—no access is allowed.
- The third and final subelement of `<security-constraint>` is `<user-data-constraint>`.
- `<user-data-constraint>` serves to define guarantees on the network used to transmit resources to clients.
  - `<user-data-constraint>` contains the element `<transport-guarantee>` for this purpose.
  - `<transport-guarantee>` has three valid values: NONE, INTEGRAL, and CONFIDENTIAL.
  - A value of NONE means that no guarantee is offered on the network traffic—and is equivalent to omitting `<user-data-constraint>` altogether (the default).
  - A value of INTEGRAL means that the web server must be able to detect any tampering with HTTP requests and responses for protected resources.

- A value of CONFIDENTIAL means that the web server must ensure the content of HTTP requests and responses so that protected resources remain secret to all but authorized parties.
- It is common practice for a web server to employ SSL (secure sockets layer) as the network transport layer to fulfill INTEGRAL and CONFIDENTIAL guarantees.

- Neither `<auth-constraint>` nor `<user-data-constraint>` is a mandatory element of `<security-constraint>`. Either or both may be used to protect resources. It makes little sense to go to the trouble of defining a `<web-resource-collection>`, then to omit both these elements, but it is legal to do so.
- After `<security-constraint>`, the next security-related deployment descriptor element is `<login-config>`. This determines how users (or other systems) authenticate themselves to a web application.
- The last security-related deployment descriptor element is `<security-role>`.
- Each `<security-role>` element (there can be as many as you like) must contain one `<role-name>` element.
- The value of `<role-name>` is a logical role name against which resources are authorized.
- Role names listed here may be used in the `<role-name>` element in `<auth-constraint>` and in `<security-constraint>`, and in the `<role-link>` element in `<security-role-ref>` in `<servlet>`.
- A logical role name must not contain embedded spaces or punctuation.

Authentication Types

- Authentication types are set up in the `<login-config>` deployment descriptor element.
- The subelement `<auth-method>` names the authentication scheme. There are four standard values: BASIC, DIGEST, FORM, and CLIENT-CERT.

- The simplest `<auth-method>` of BASIC will trigger a browser to show a standard dialog when accessing a secure resource for the first time in your

web application. The dialog allows entry of a user name and password, and displays a realm name.

- The realm name (appearing in the browser dialog) can be set with the `<realm-name>` subelement of the deployment descriptor and should name a registry of user credentials accessible from your web server.
- The password details are Base64-encoded when passed from the browser to your web application. This provides a small measure of protection, but very little — Base64 decoders are freely available to hackers.
- BASIC is not useless, however, if SSL is used to encrypt all network traffic from browser to web server.
- DIGEST authentication (`<auth-method>DIGEST</auth-method>`) imposes better security by encrypting authentication information.
- The encryption process uses as input transient server information (often called a "nonce") and authentication information (including the user ID and password).

- The digest is sent to the server. The server has access to all the same input values to make its own digest. If the digests match, the user is authenticated.
- Not all browsers support digests or make the digests in the same way — this makes adoption of the DIGEST method more difficult.
- FORM authentication associates a custom web page with the login process, as an alternative to a browser dialog.
- The custom web page for logging in must contain an HTML <FORM> whose action is j_security_check and whose method is POST.
- The form must include input fields named j_username and j_password.
- A custom error page must also be provided — there are no rules about the HTML for this.
- The login and error pages are then specified in the deployment descriptor in the `<form-login-config>` element, which is a subelement of `<login-config>`.
- The login page goes in the `<form-login-page>` subelement of `<form-login-config>`.

- The error page is specified in the `<form-error-page>` subelement of `<form-login-config>`.
- The values for these elements *must* begin with a forward slash, which denotes the root of the web context.
- `<form-login-config>` is — obviously — only relevant when the `<auth-method>` is FORM.
- The first secured resource you request in a web application will cause you to be redirected to the login page.
- If there is an error logging in, you are redirected to the error page.
- Log-in information (user, password) is not protected in any way across the network with FORM authentication. As for BASIC authentication, you need to also use a secure protocol such as SSL if greater protection is needed.
- CLIENT-CERT is the fourth and final authentication method. It is the most secure but also the trickiest to set up.
- This method relies on asymmetric keys — that is, a pair of keys, one public (available to anyone) and one private (kept secure on the key owner's hardware). Anything encrypted with the public key can be decrypted with the private key — and vice versa.
- The client (and would-be digital certificate owner) first generates a private and public key. The client sends the public key — and other information — to a third-party certificate authority. The certificate authority binds this information and the client public key into a certificate.
- The certificate authority adds a digital signature encrypted with its private key, which makes a digest of information already in the certificate.
- This process serves two purposes: (1) to prove the certificate is vouched for by the certificate authority (only its public keys can be used to read the signature) and (2) to prevent tampering with any aspect of the certificate.
- The certificate is returned to the client, who installs it in his or her browser (or other client device).
- When the server requests authentication from the client browser, the browser supplies the certificate. If the certificate is on the server's approved list of certificates, authentication takes place.