Security

1. What is Authentication?
    a. Authentication is the process of identifying some party to a web application. Here party can be a human user or other system.
    b. A most common way of authentication is providing a user ID and a password, sent unencrypted over the network. More secure approaches might go for encrypting the authentication information, or even the entire request and response, which will be mainly through digital certificates.
    c. **Servlet spec definition:** The means by which communicating entities prove to one another that they are acting on behalf of specific identities that are authorized for access.
2. What is Authorization?
    a. Authorization rules determine what an identifi ed party is allowed to do within a web application—which resources can be accessed and what can be done with those resources.
    b. **Servlet spec definition:** The means by which interactions with resources are limited to collections of users or programs for the purpose of enforcing integrity, confidentiality, or availability constraints.
3. How is Authentication and authorization related?
    a. Authentication and authorization go hand in hand. You can't have authorization without authentication happening first. It's rare to find a system with Authentication, but without employing some kind of authorization rule as well.
4. What is data integrity?
    a. Data integrity is the business of proving that what you sent to or from a web application has not been tampered with on the way.
    b. If a client encrypts its request in a way that only the server will understand (and vice versa), that's a guarantee that no modification has occurred.
    c. **Servlet spec definition:** The means used to prove that information has not been modified by a third party while in transit.

5. What is confidentiality?
    a. The "confidentiality" (or "data privacy") refers to mechanisms to encrypt your network traffic so that no code-cracking approach can reveal the plain contents.
    b. **Servlet spec definition:** The means used to ensure that information is made available only to users who are authorized to access it.
6. How are users associated with roles in tomcat?
    a. We find within it a configuration file called tomcat-users.xml. This contains a list of roles, followed by a list of users together with a comma-separated list of the roles to which they belong.
    b. Example:
        i. <?xml version='1.0' encoding='utf-8'?>
        ii. <tomcat-users>
        iii. <role rolename="tomcat"/>
        iv. <role rolename="role1"/>
        v. <role rolename="manager"/>
        vi. <user username="tomcat" password="tomcat" roles="tomcat"/>
        vii. <user username="role1" password="tomcat" roles="role1"/>
        viii. <user username="both" password="tomcat" roles="tomcat,role1"/>
        ix. </tomcat-users>
7. Is associating users and systems with particular roles server specific?
    a. Yes. You say nothing about individual users in the deployment descriptor: What you do is associate a URL with a particular role. Only users within that role can use that URL. The user set up is server specific.
8. Give the basic idea of encryption using public and private keys?
    a. The encryption process usually involves public and private keys (sometimes called asymmetric—they're a matched pair, but not identical). Private keys are kept strictly private, whether client-side or server-side. Public keys are broadcast to interested parties. If a server uses a client's public key to encrypt a message, only the client's private key can decode it.
    b. An example scenario:
        i. Client makes request for secure resource.

    ii. Server provides its public key.
    iii. Client provides its public key, encrypted with the server's public key.
    iv. Server provides secure resource, encrypted with the client's public key.

9. How does encryption using public and private keys help enforce integrity?
    a. You can encode a message with your own private key instead of someone else's public key. As long as those who have your public key are sure it's your public key and no one else's, then they can be sure that a message you encoded with your private key comes from you. So encryption helps to enforce integrity.
10. Which is the deployment descriptor element used when requesting data integrity for a particular resource?
    a. <transport-guarantee>INTEGRAL</transport-guarantee>
11. Which is the deployment descriptor element used when requesting confidentiality for a particular resource?
    a. <transport-guarantee>CONFIDENTIAL</transport-guarantee>
12. ……… Is a private/public key technology for communicating privately over the Internet that was originally developed by the Netscape Communications Corporation and is now incorporated in practically every web device under the sun.
    a. SSL

Deployment Descriptor Security Declarations

13. What do you mean by declarative security?
    a. You declare the security you want in the deployment descriptor instead of enshrining it in code.
14. List down the **top level security elements** in the web.xml file?
    a. There are three top-level deployment descriptor elements:
        i. **<security-constraint>** - it defines what resource we're securing, what roles can access the resources defined in the web resource collection.
        ii. **<login-config>** - defines what authentication mechanism is to be used.
        iii. **<security-role> -** catalogues any security roles in use by the web application.
15. List down the elements of **<security-constraint>**?
    a. **<web-resource-collection>** - defines the resource to be protected and also the HTTP methods by which it can be accessed.
    b. **<auth-constraint>** - determines which roles are allowed access to the resource.
    c. **<user-data-constraint>** - decides what protection (if any) is required when transporting the resource over the network. These apply to the requests and responses that pass to and from the web container.
16. List down the sub elements of **<web-resource-collection>**?
    a. **<web-resource-name>** - logical name for the web resource collection. it has no technical significance—it's just a memory aid to help identify the group.
    b. **<description>** - Any number of description lines contained by <description> tags can follow—including none at all.
    c. **<url-pattern>** - define a URL pattern for any resource in your web application. There must be at least one <url-pattern> included for the web-resource collection.
    d. **<http-method>** - Using **<http-method>** you can include a list of HTTP methods which can be only executed by the roles mentioned in the associated **<auth-constraint>** on the associated resource. Will contain one each for every HTTP method we need to define. If you don't specify this method, it is as if every HTTP method is added.
17. Give an example for <web-resource-collection>?

   a. **Example:**
      i. **<security-constraint>**
      ii. **<web-resource-collection>**
      iii. **<web-resource-name>**All Resources</web-resource-name>
      iv. **<url-pattern>**/</url-pattern>
      v. **<http-method>**DELETE</http-method>
      vi. <http-method>PUT</http-method>
      vii. </web-resource-collection>
      viii. <auth-constraint />
      ix. </security-constraint>
   b. The URL pattern here (/) encompasses all resources within the context for the current web application. So this will deny DELETE or PUT on any web resource.

18. List down the sub elements of <auth-constraint>?
   a. Elements are:
      i. <description>
      ii. <role-name>
   b. The roles names that you choose must (according to the servlet specification) be listed in the <security-role> element.
   c. Example
      i. <auth-constraint>
      ii. <role-name>employee</role-name>
      iii. <role-name>supervisor</role-name>
      iv. </auth-constraint>

19. What if there is no <auth-constraint> for your security constraint?
   a. It simply means that the web resource collection is open to all, regardless of role or authentication.

20. What if the <auth-constraint> for your security constraint is present with no role names?
   a. It will deny access to the resource for any role whatsoever.

21. How can you define the <auth-constraint> for all roles defined in the web application?
   a. <role-name>*****</role-name>

22. When more than one security constraint is set up for the same web resource collection, what will happen?
    a. Permissions for the roles in the <auth-constraint> elements will get added up. However if there is a no value <auth-constraint> it will override others. Even if you set up a web resource collection open to all roles, if the same URL patterns (for the same HTTP methods) are protected elsewhere with the no-value authority constraint, access will be denied.
23. When a resource is requested to which access is always denied (because of the "no-value" authorization constraint), which http response code will result?
    a. If there are potential roles, then authentication must happen. If it hasn't happened already, the web server sends a 401 (SC_UNAUTHORIZED) response code, which causes the browser to supply authentication information in some form or other. If subsequent checking shows that the authenticated user is not a role entitled to the resource, the web server rejects the request with a 403 (SC_FORBIDDEN) response code.
24. List down the sub elements and their functions of the <user-data-constraint> element?
    a. <user-data-constraint> must have a **<transport-guarantee>** element within it. This has three valid values: NONE, INTEGRAL, CONFIDENTIAL.
        i. **NONE:** No constraints are applied to the traffic in and out of the container. The requests and responses can pass in plain text over the network. Setting the transport guarantee to this value is the same as leaving out <user-dataconstraint> altogether.
        ii. **INTEGRAL**: The web container must impose data integrity on requests and responses. How it does this is up to the web container, but typically it will use SSL (secure sockets layer) as the communication medium.
        iii. **CONFIDENTIAL:** The web container must ensure that communicated data remains private. If you have confidentiality, integrity is implied. However, web containers mostly achieve the guarantee by using SSL.
25. Give any use of a web resource collection without an <auth-constraint>—such that any user (even an unathenticated user) can access the resource?

a. You might still protect your resource collection with a <user-data-constraint> but allow open access. Anyone can access this page, but it's best to ensure that their details remain private when transmitted back to the web server. By specifying a CONFIDENTIAL transport guarantee without any authority constraint, you achieve this end.

26. List down the important sub elements of <login-config>?
    a. Elements are:
        i. <description>
        ii. **<auth-method> -** Will have one of the four authentication types BASIC, FORM, DIGEST, and CLIENT-CERT.
        iii. **<realm-name> -** The realm is the registry used to store user account information. If server has more than one realm at its disposal and that a deployer will need to specify which one is meant here in web.xml. However, it's not mandatory—mostly, servers are concerned only with validating against one realm. Only for BASIC and DIGEST.
        iv. **<form-login-config>** - Only for FORM.
    b. Example:
        i. <login-config>
        ii. <auth-method>BASIC</auth-method>
        iii. <realm-name>MyUserRegistry</realm-name>
        iv. </login-config>

27. List down the important sub elements of **<security-role>**?
    a. Elements are:
        i. <description>
        ii. <role-name>
    b. You only include one role name per security role, but <security-role> can appear as many times as needs be in the deployment descriptor.

28. Are the role names case sensitive?
    a. The web container matches role names case sensitively when determining access to secured resources

29. List down the major security related methods in the HttpServletRequest?

    a. **getRemoteUser()** - returns a String containing the name of the authenticated user.
    b. **isUserInRole()** – Returns a boolean indicating whether the authenticated user is included in the specified logical "role". Roles and role membership can be defined using deployment descriptors. If the user has not been authenticated, the method returns false. The <**security-role-ref>** element is used when an application uses the HttpServletRequest.**isUserInRole**(String role) method.
    c. **getUserPrincipal()** - Instead of returning a plain String, it returns a java.security.Principal object. ultimately the only useful thing you can do with a Principal object is to call getName() to return a String with the user's (principal's) name.

30. How is the <**security-role-ref>** element important for **isUserInRole()?**

    a. The <**security-role-ref>** element is used when an application uses the HttpServletRequest.**isUserInRole**(String role) method. The value of the <**role-name>** element must be the String used as the parameter to the HttpServletRequest.isUserInRole(String role) method. The role-link must contain the name of one of the security roles defined in the **<security-role>** elements. The container uses the mapping of security-role-ref to security-role when determining the return value of the call.
    b. Example Usage:
        i. <security-role-ref>
        ii. <role-name>MGR</role-name>
        iii. <!-- role name used in code -->
        iv. <role-link>employee</role-link>
        v. </security-role-ref>
        vi. <security-role>
        vii. <role-name>employee</role-name>
        viii. </security-role>

31. List down the Java EE **Authentication Types**?

    a. **BASIC** - forces the appearance of a dialog box in browsers inviting user and password details. They're not very secure.

   b. **FORM** - improved version of BASIC that you supply your own design of web page to solicit user and password details, instead of being stuck with the browser’s dialog box. The transmission details are however the same as BASIC.
   c. **DIGEST** - imposes encryption rules on the password.
   d. **CLIENT-CERT** - all security details are kept in an electronic document called a certificate.

32. The realms are applicable only to ………. and ……… forms of authentication.
    a. BASIC and DIGEST
33. How is the password sent in **BASIC** authentication? Is it as plain text?
    a. The password is not sent as plain text but is passed through a process called Base64 encoding. But it’s not the same as encryption. Base64 encoding and decoding tools are freely available on the Web.
34. How can you ensure encryption with **BASIC** authentication?
    a. BASIC authentication can be made secure by providing a transport guarantee to ensure encryption. That way, all parts of the request—including password details—are concealed from theft across the network.
35. Are the client and server both authenticated in basic authentication?
    a. When using Basic Authentication only client is authenticated, the target server is NOT authenticated.
36. How does **DIGEST** Authentication work?
    a. DIGEST authentication uses a secure algorithm to encrypt the password and other security details.
    b. When the browser tries to access a secure resource, the server generates a random value called a “nonce” and passes this to the browser. The nonce can be based on anything—often, a unique identifier for the server (such as an IP address) and a timestamp.
    c. The browser uses this value and, together with other pieces of information—always the user ID and password, sometimes URI and HTTP method as well—applies the digest encryption algorithm.

d. The idea is to turn the seed information into junk, which can never be translated back into human-readable text. This junk is called the digest, and the client sends it to the server.
e. The server can't use the digest to decode security details. What it can do, though, is to generate its own digest—using the nonce value it provided and known user and password details—and compare this with the digest passed from the browser. If they match, the client is considered authenticated.

37. Give any disadvantages of **DIGEST** authentication?
    a. What DIGEST authentication rely on is that the server and browser have identical expectations about the digest: which algorithm to use and which pieces of information to apply the algorithm to.
    b. But different browser vendors do different things in support of DIGEST authentication. You need to know (and test) the clients you expect your web application to support, and that may not be easy. Hence, DIGEST lags behind other authentication types in terms of adoption.
38. What are the rules for creating the login page when using **FORM Authentication**?
    a. Rules are:
        i. The HTML form must use the POST method (GET is not acceptable).
        ii. The form must have "j_security_check" as its action.
        iii. The form must include an input-capable field for user called "j_username."
        iv. The form must also include an input-capable field for password called "j_password."
    b. Example:
        i. <html>
        ii. <head><title>Login Form</title></head>
        iii. <body>
        iv. <form action="j_security_check" method="POST">
        v.
Name: <input type="text" name="j_username" />
        vi.
Password: <input type="password" name="j_password" />
        vii.
<input type="submit" value="Log In" />

viii. </form>
ix. </body>
x. </html>

39. Can we have dynamic JSP pages as login pages in FORM authentication?
    a. Yes
40. List down the sub elements of **<form-login-config>**?
    a. <form-login-config> is unique to FORM authentication. Its sub elements are:
        i. **<form-login-page>** - The login page
        ii. **<form-error-page>** - Form-based authentication also demands that you provide an error page. There are no rules for the content of such a page.
    b. Example:
        i. <login-config>
        ii. <auth-method>FORM</auth-method>
        iii. <form-login-config>
        iv. <form-login-page>/login.html</form-login-page>
        v. <form-error-page>/error.html</form-error-page>
        vi. </form-login-config>
        vii. </login-config>
41. How does **FORM** authentication work?
    a. The first secure resource you attempt to access in a web application will not be sent to you directly. Instead, the server caches the URL you are trying to reach and redirects you to the form login page. You supply a user ID and password; assuming that the server is happy with these credentials, you are then passed on to the URL you requested in the first place. However, if your login fails for any reason, the server redirects you to the error page you specified.
42. Can we use Form authentication with SSL? If possible, give any advantage/need of that approach?
    a. For FORM Authentication, there is no intrinsic protection for security information. You don't even get the Base64 encoding of the password. But you can get around that by using a virtually private network, or a secure protocol such

as SSL for your transport guarantee. Therefore FORM Authentication can be used with them.

43. Give any disadvantages of **FORM** authentication?
    a. You can’t go directly to the login form. You have to attempt to access an otherwise secured resource and let the server redirect you to the login page.
44. If you want to design an unsecured home page with a login field for registered users in the top right-hand corner, then form-based authentication is not for you. Explain?
    a. You can’t go directly to the login form. You have to attempt to access an otherwise secured resource and let the server redirect you to the login page. If you try to access the login page directly, an error will be thrown.
45. Of the four authentication types HTTP Basic Authentication, Form Based Authentication, HTTP Digest Authentication, HTTPS Client Authentication, which can be considered optional?
    a. FORM & Client-Cert authentication are inside the J2EE specs. Basic authentication is supported by all browsers/J2EE web containers. So that makes Digest authentication, the only that is optional.
46. What will an X.509 certificate contain?
    a. Version of the X.509 standard (v1, v2, or v3).
    b. A serial number unique to the certificate authority (VeriSign, Thawte) issuing the certificate.
    c. The signature algorithm used to digitally sign the certificate.
    d. Validity period: when the certificate will start and expire.
    e. The subject: in other words, you, the requester of the certificate. This is held as a “distinguished name”. A distinguished name has several components, including a common name (an individual), organizational unit, organization name, locality name, state name, and country.
    f. Issuer name: the name of the certificate authority, again held as a distinguished name.
    g. A digital signature, encoded with the certificate authority’s private key. This will be a digest of information within the certificate—which also means that

tampering with the certificate (not just the digest itself) will render it immediately invalid.

h. Last but very much not least: the subject's (your) public key.

47. How does **CLIENT-CERT** Authentication work?

a. CLIENT-CERT, which uses digital certificates to achieve authentication. This is the most secure form of authentication.

b. Anyone can create a certificate, using specialized (but publicly available) software such as "keytool," which is shipped with the J2SDK. By passing the right parameters, keytool (or the equivalent) generates a private key and a matching public key, usually stored in some fully encrypted form on the creating computer's hard drive.

c. From this "keystore" you can extract a certificate file that is fully technically valid.

d. The usual procedure is to pass on your "self-signed" certificate details to a properly established certificate authority(VeriSign, Thawte). These companies verify your identity and "rubber-stamp" your certificate—or, more correctly, produce a new certificate based on the details you supplied, vouched for by them.

e. The most important action they take is to use their own private keys to digitally sign your certificate. Practically all browsers and web servers are in possession of these company's public keys. This gives them the means to check a digital signature (from VeriSign, Thawte, or whomever) on your certificate, verifying that your certificate is at least vouched for by a trusted third party.

f. Once you have your certificate, you can install it into your browser. Then, when you access a web application that demands client certification, your browser can supply a client certificate. If this is on the approved list of the server's allowed certificates, the transaction can continue.

g. In general use of the Internet, the server provides a certificate to your browser. Depending on your browser's security settings, you generally see a dialog box asking whether or not you want to trust the certificate the server is offering you (we hope signed by Thawte, VeriSign, or whomever). If you accept, the transaction can continue, and the server's public key can be used to encrypt communications between you.

48. Can we have new values (other than BASIC, FORM, DIGEST, and CLIENT-CERT) for the <auth-method> element?
    a. There's nothing stopping a web server vendor from supporting its own style of authentication and permitting new values (other than BASIC, FORM, DIGEST, and CLIENT-CERT) for the <auth-method> element.
49. What is HTTPS? Is it necessary for the J2EE compliant web containers should support the HTTPS protocol?
    a. HTTPS (Hypertext Transfer Protocol over Secure Socket Layer) is a secured communication protocol between web browser and web server. It encrypts any communication that a user sends to a web server and decrypts at server side. Similarly, it encrypts any communication that a web server sends to a web browser and decrypts at browser side. HTTPS Client Authentication **requires the client to provide its public key certificate.**
    b. J2EE compliant web containers should support the HTTPS protocol.
50. (Yes/No) Web containers are required to support unauthenticated access to unprotected web resources.
    a. Yes. That is the default behavior.