**6.2 Setting the Session/Certificate to a Valid Expiration Time (Session Timeout Test)**

|  |  |
| --- | --- |
| No. | EdgeGallery Security Compliance Test Case 6-2 |
| Test tool | N/A |
| Test description | Check whether a mechanism exists for re-login after the browser window times out. |
| Configuration Steps | 1. Known website address 2. Web services are running properly. 3. The login authentication module is available for web services. 4. The user name and password are always correct. |
| Step 1 | Log in to the system using a normal user name and password. |
| Step 2 | Increase the idle time of the browser window (11 minutes) by one minute.  Note: Currently, the validity period is one day. |
| Step 3 | Refresh the browser and check whether you need to log in again.  Note: You can also log in to the back-end web server and check the value of session-timeout in the web.Xml file. The value of session-timeout indicates the session timeout interval. |
| Test verdict | The session timeout interval is less than or equal to 10 minutes. You need to log in again after refreshing the browser. |
|  |  |
|  |  |

**6.3 Protect application sessions from information disclosure. Ensure that the session data is not used by other sessions or is invisible.**

|  |  |
| --- | --- |
| No. | EdgeGallery Security Compliance Test Case 6-3 |
| Test tool | N/A |
| Test description | Protects application sessions from information disclosure. Ensure that the session data is not used by other sessions or is invisible. The session does not access the session data. |
| Configuration Steps | 1. Known website address   <https://daily.mecm.edgegallery.org/>   1. Web services are running properly. 2. The login authentication module is available for web services. 3. The user name and password are always correct. |
| Step 1 | Log in to the system using a normal user name and password, and manually obtain the token ID from the browser,  Note: The loginusermgmt interface is invoked twice and then the mecm login interface is invoked. |
| Step 2 | Use Postman to send a request, enter a correct token ID in the header, and send the request twice.  Note: The loginusermgmt interface is invoked twice and then the mecm login interface is invoked. |
| Step 3 | Check whether the session IDs returned by the two clients are the same. |
| Step 4 | Check whether the log and response sent by the request contain sessionID. |
| Test verdict | The log and response do not contain sessionID. The Postman sends two requests. If the token ID is incorrect, the request fails. |

**6.4 When a user logs out, the session must be invalid (whether the session information is cleared during logout).**

|  |  |
| --- | --- |
| No. | EdgeGallery Security Compliance Test Case 6-4 |
| Test tool | WebScarab (open-source version can be used) download address:  <https://www.pw88.com/teach/anquan/183.html> |
| Test description | Check whether session information is cleared during logout. |
| Configuration Steps | 1. Known Website Address 2. Web services are running properly. 3. A page with the logout function exists. |
| Step 1 | Log in to the system using a normal user name and password. |
| Step 2 | Enable the WebScarab and configure the WebScarab to intercept GET and POST requests. |
| Step 3 | In the browser, set the proxy server IP address to 127.0.0.1 and port number to 8008. |
| Step 4 | Perform some operations (such as modifying personal information) on the web page. All these operations are intercepted by the WebScarab. In this case, click Accept Changes on the WebScarab page that is displayed.  Button. The request is recorded by WebScarab. |
| Step 5 | Click Logout on the web page. |
| Step 6 | Click the Manual Request tab on the WebScarab page, select the URL request generated in step 4 from the Previous Requests drop-down list, and click Fetch Response to resend the URL request. |
| Step 7 | View the command output on the Raw tab page of the Response page of the WebScarab. If the operation in step 4 can be successfully performed, a security vulnerability exists. |
| Test verdict | "HTTP/101 302 Moved Temporarily" Is Displayed on the Raw Tab Page of the Response of the WebScarab. Pages that can be accessed only by login cannot be accessed, and tasks that can be completed only by login cannot be performed. |
| Remark | If there are multiple logout pages, repeat the test procedure to test all the logout pages. |

**6.5 A new session and session ID must be generated for successful authentication and re-authentication. ()**

|  |  |
| --- | --- |
| No. | EdgeGallery Security Compliance Test Case 6-5 |
| Test tool | burpsuite |
| Test description | Successful authentication and re-authentication must generate a new session and session ID |
| Configuration Steps | 1. Known website address   2. Web services are running properly.  3. The login authentication module is available for web services.  4. The user name and password are always correct. |
| Step 1 | Log in to the system using a normal user name and password. The user manager module is invoked after login.  <https://daily.developer.edgegallery.org/> |
| Step 2 | Record the session ID generated during the first login, |
| Step 3 | Log out of the system and log in to the system again. Record the session ID generated after the second login. |
| Step 4 | Check whether the two session IDs are the same. |
| Test verdict | If the two session IDs are different, a new session and session ID are generated. |

**4.1 Do not use proprietary and non-standard encryption algorithms.**

|  |  |
| --- | --- |
| No. | EdgeGallery Security Compliance Test Case 4-1 |
| Test tool | Nessus |
| Test description | Proprietary algorithms are prohibited for encryption and decryption, including:  1. User-defined data conversion algorithms that are executed by transformation, character shift, and replacement.  2. Pseudo encryption for data encryption using encoding (for example, Base64 encoding).  Sub-check items:  1. Do not use self-designed cryptographic algorithms or algorithms that do not disclose the details of the algorithms.  2. Do not reconstruct or use standard cryptographic algorithm.  3. Do not use self-defined data conversion by transformation, character shift, or replacement.  4. Do not use coding (such as Base64 coding) to encrypt data.  5. Do not use error control codes (such as parity check and CRC) to implement integrity check. |
| Configuration Steps | Obtaining Product Source Codes and Design Documents |
| Step 1 | Check the design document to see whether the encryption and decryption design meets the requirements. |
| Step 2 | Check all sensitive data, such as passwords and keys, stored in the database, files, and memory. Check whether the algorithms meet requirements. |
| Step 3 | Check whether the code meets requirements. |
| Test verdict | Common open-source cryptographic components are used in the product, which meets the test requirements. No proprietary and non-standard cryptographic algorithms are used. |

**4.2 Do not use insecure encryption algorithms (strong encryption algorithms are recommended).**

|  |  |
| --- | --- |
| No. | EdgeGallery Security Compliance Test Case 4-2 |
| Test tool | Nessus  Download URL:<https://www.tenable.com/downloads/nessus?loginAttempted=true> |
| Test description | Check whether insecure encryption modes are used in the entire system.  Check whether the generation of the root key and working key is secure when the hierarchical key management structure is used.  Check whether the signature algorithm of the certificate generated or used by the system meets security requirements. |
| Configuration Steps | 1. The system is running properly. 2. Obtaining Product Source Codes and Related Documents 3. For network transmission, the communication between the client and server uses password authentication. 4. The product uses a hierarchical key management structure to distinguish between root keys and working keys. |
| Step 1 | Check the code. The keys of the database, JWT, and certificate of the user/app/developer module are transferred through environment variables. Check whether the algorithms used for password encryption contain insecure cryptographic algorithms, such as DES and RC4 and RSA-1024, insecure hash algorithms SHA-0, SHA-1, MD2, MD4, and MD5, and insecure key negotiation algorithms DH-1024. |
| Step 2 | Requirement 1: Check whether the channel or encryption mode is secure.   1. Check whether the channel is encrypted using the TLS secure encryption channel according to the encryption algorithm specifications. 2. Check whether secure hash algorithms, such as sha256, are used to encrypt passwords that are irreversible based on the encryption algorithm specifications and whether security salts are added. 3. In scenarios where the encryption algorithm is reversible, check whether the reversible algorithm such as AES256 is used for encryption and whether the key is securely stored.   Requirement 2: Check whether weak encryption algorithms and functions exist.   1. Use Nessus to scan the system and check whether weak encryption algorithms are used. 2. View the product documentation to find the feature description of sensitive design information. 3. Check whether the encryption mode of sensitive information meets requirements. 4. Search for the encryption function in the source code and check whether the encryption mode is proper.8 |
| Step 3 | 1. Search for the root key generation description in the product documentation or search for the root key generation code in the product source code to check whether the root key is hard coded in the product. 2. If the root key uses the method of generating the initialization vector and key material, check whether the initialization vector and key material are hard-coded in the product.   Note: This can be combined with the urgent interviews with developers. |
| Step 4 | Use the certificate generation function to check whether the number of RSA key pairs is optional. The default value is 2048 bits or more. Check whether the hash algorithm used for signatures is optional. Check whether the default value is SHA256 or a more complex hash algorithm. |
| Step 5 | 1. Analyze the working key usage based on the features supported by the product, search the source code, and check whether the working key is encrypted by the root key when being stored in the system, and whether different applications use different working keys. 2. Check the key usage process and check whether the key material is deleted in time after the key is used. 3. View the product documentation to check whether the root key and working key are described. 4. Check the product documentation to see whether the application scenario of the shared key exists. If yes, check the generation mode of the shared key.   Note: You can interview the R&D personnel based on. |
| Test verdict | 1. Sensitive information must be encrypted and stored in the product. In scenarios where sensitive information is not required, an irreversible encryption algorithm must be used. 2. Proprietary encryption algorithms (including pseudo encryption implemented by Base64 encoding) are prohibited. 3. The encryption algorithms used by the product must meet the complexity requirements such as SHA256 and AES256. 4. The system does not store the root key, and generates the root key based on the initial vector and key material when needed. 5. Different types of services use different working keys and use the root key to encrypt and save keys. The working key and key materials are not provided. After the keys are used up, the memory for storing the information is cleared in a timely manner. In principle, the product does not provide an interface for sending the root key to the working key. When working keys are distributed between systems, encrypted transmission (direct encryption or SSL-based encryption) is required. It is recommended that the automatic key-up mechanism be used for negotiation between systems. 6. When the system generates a certificate, the default RSA key pair is 2048 or more, and the default signature algorithm is sha256, ase256, or a more complex algorithm. If a certificate with low algorithm complexity is supported, a security warning message is displayed. The RSA public key used by the system is 2048 bits or more, and the signature algorithm is sha2RSA or a more complex algorithm. |
|  |  |

**4.4 Use encrypted and secure random numbers.**

|  |  |
| --- | --- |
| No. | EdgeGallery Security Compliance Test Case 4-4 |
| Test tool | Fortify; Coverity; Souce Insight |
| Test description | Random numbers are the basis of cryptography. The random numbers of keys ensure that attackers cannot guess the keys and therefore cannot decrypt the ciphertext. The salt values are random. As a result, hackers can obtain the values queried in ciphertext, which are the passwords with salt values instead of the passwords set by users, we can't construct a rainbow table and then return the remote password ciphertext. The randomness of IVs ensures that attackers cannot find the statistical characteristics of ciphertext. If randomness is not ensured, various attacks are easy and the system is exposed to great threats. |
| Configuration Steps | Obtaining Test Object Design Documents and Codes |
| Step 1 | Use Fortify and Coverity for the user module to scan code for encryption algorithm problems. |
| Step 2 | Use Souce Insight to search for the forbidden java.util.Random class of hava. |
| Step 3 | Use Source Insight to check whether the same seed is set for java.security.SecureRandom() each time. As a result, the random number generated each time is the same. |
| Step 4 | Use Source Insight to check whether the generated random number contains more than 24 characters or 192 bits.  Note: Do not use the java.util.Random class of Java to generate random numbers for security purposes.  Do not set phase each time when java.secutiry.SecureRandom() is used to generate random numbers.  Same seed. |
| Test verdict | Secure random numbers are used to generate session IDs.  Security is considered secure if the security configuration for generating a session is used. |

**4.3 The key used to encrypt data cannot be hardcoded. ()**

|  |  |
| --- | --- |
| No. | EdgeGallery Security Compliance Test Case 4-3 |
| Test tool | N/A |
| Test description | No hard-coded encryption key exists in the code. |
| Configuration Steps | Through interviews or other methods, analyze where storage encryption keys and communication encryption keys are designed in the system. |
| Step 1 | Communicate with the R&D personnel to check whether the key is generated using a tool and determine the scenarios where the encryption key is used. |
| Step 2 | Analyze the modules that may use encryption keys based on scenarios. Focus on the code walk-through and check whether hard code exists. |
| Test verdict | Keys for encrypting sensitive data and transmitting encrypted data cannot be hardcoded in codes. Keys that need to be encrypted must be stored in files or databases. |

**5.3 All input must be limited to the appropriate size limit. ()**

|  |  |
| --- | --- |
| No. | EdgeGallery Security Compliance Test Case 5-3 |
| Test tool | WebScarab |
| Test description | The length of the interface input parameter is verified. The length of the body is not limited.  Uploading files: The file type and file name format are verified based on services to prevent ZIP bombs from being met. The maximum file size is x MB. |
| Configuration Steps | Web services are running properly.  The website to be tested has an upload page. |
| Upload file test |  |
| Step 1 | Log in to the website,Open the file upload page.<https://daily.developer.edgegallery.org/#/mecDeveloper/test/apply> |
| Step 2 | Click Browse, select a local JAR file, and click OK. |
| Step 3 | If the client restricts the type of the file to be uploaded (for example, the .cisa file is allowed), select the .cisa file to be uploaded. |
| Step 5 | Click Browse, select the cisa folder with a high compression ratio greater than 100 MB, and upload the folder. Log in to the tested system to check whether the CPU usage, memory usage, and disk usage increase abruptly. |
| Input validate test |  |
| Step 1 | Check the code to check whether the length of the input parameter is verified. |
| Step 2 | Add the interface document based on the interfaces involved in the interface document.  Use Postman to send a request, transfer parameters whose lengths are beyond the specified length, and check whether the request fails to be sent. |
| Step 3 | Log in to the web page and test the length of all parameters that need to be entered to check whether the parameter length meets requirements or an error is reported. |
| Test verdict | The system checks the size of a file to be uploaded. If the file size is too large, the system cannot be attacked and files other than the .cisa format cannot be uploaded.  The request for transferring a parameter whose length is beyond the specified length fails, and a message is displayed indicating that the parameter length is too large. |