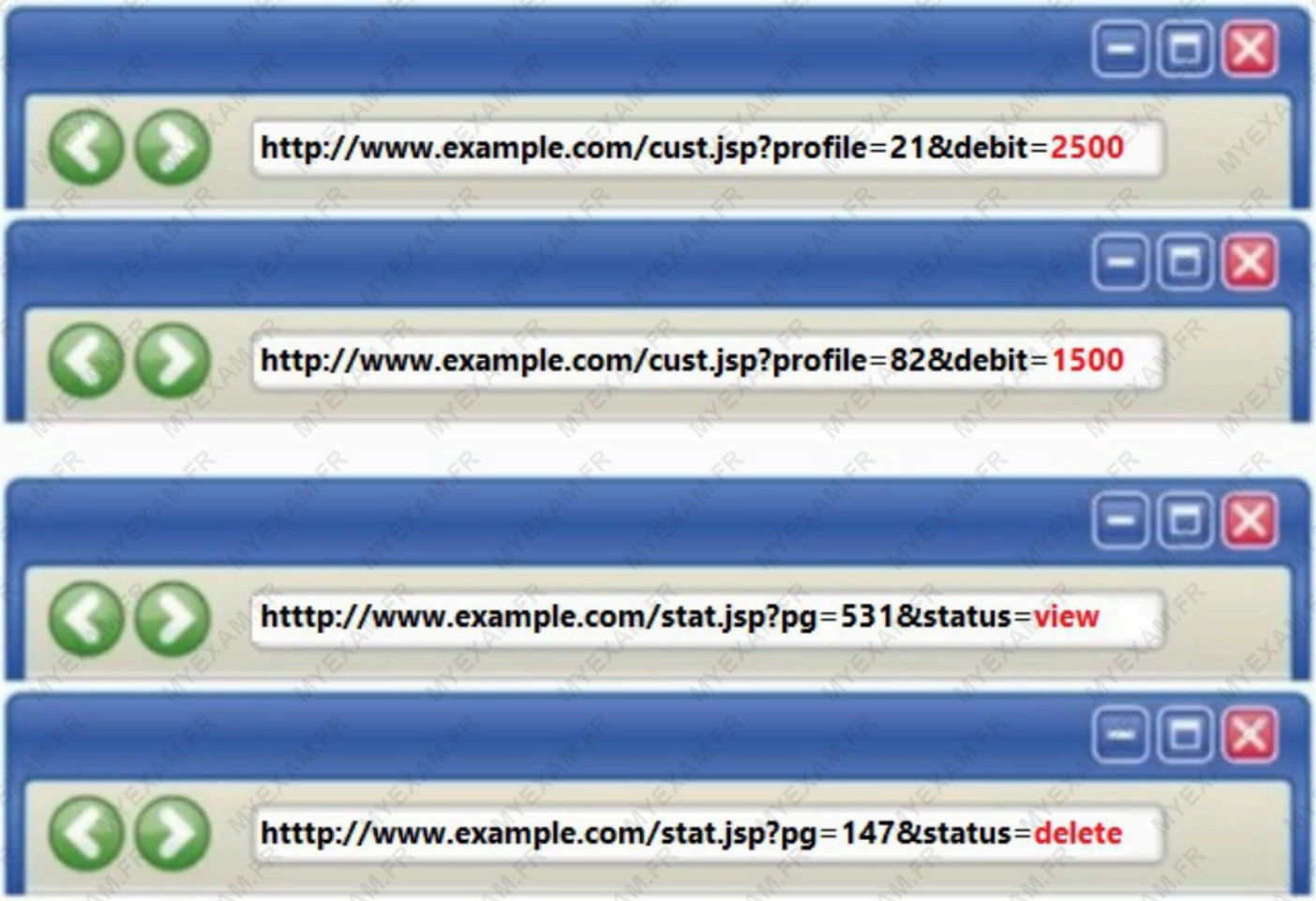
Eccouncil

(312-96)

Certified Application Security Engineer (CASE) - JAVA

Total: **49 Questions**

Link:



A. SQL Injection Attacks
B. Session Fixation Attack
C. Parameter Tampering Attack
D. Denial-of-Service Attack

Answer: C

Question: 3

According to secure logging practices, programmers should ensure that logging processes are not disrupted by:

A. Catching incorrect exceptions
B. Multiple catching of incorrect exceptions
C. Re-throwing incorrect exceptions
D. Throwing incorrect exceptions

Answer: D

Explanation:

D is correct because throwing incorrect exceptions can lead to unhandled application states, disrupting logging processes.

as a threat-classification taxonomy.
DREAD – A scoring model that evaluates the severity of a threat based on several factors; it builds on a classification but does not define the categories themselves.
SMART – A goal-setting framework (Specific, Measurable, Achievable, Relevant, Time-bound) unrelated to threat classification.

Therefore, **STRIDE** is the correct answer because it directly provides the threat categories employed during threat modeling.

References

Microsoft Security Development Lifecycle – STRIDE Threat Modeling: https://learn.microsoft.com/en-us/security/security-development-lifecycle/threat-modeling
OWASP – DREAD Threat Scoring Model (historical context): https://owasp.org/www-project-cheat-sheets/cheatsheets/Threat_Modeling_Cheat_Sheet.html

Question: 5

Which line of the following example of Java Code can make application vulnerable to a session attack?

```
1:  Public class storecookie extends HttpServlet {
2:  Public void doGet () {
3:  Cookie ssnCookie = new Cookie ("Session Cookie"+ value);
4:  ssnCookie.setMaxAge (3600);
5:  response.addCookie (ssnCookie);
6:  }
```

A. Line No. 1
B. Line No. 3
C. Line No. 4
D. Line No. 5

Answer: B

Question: 6

Alice, a Server Administrator (Tomcat), wants to ensure that Tomcat can be shut down only by the user who owns the Tomcat process. Select the appropriate setting of the CATALINA_HOME/conf in server.xml that will enable him to do so.

A. < server port="" shutdown="" >
B. < server port="-1" shutdown="" >
C. < server port="-1" shutdown="SHUTDOWN" >

frameworks, such as SLF4J, used in Java applications.

In Java logging, particularly with frameworks like Log4j or SLF4J, the method isDebugEnabled() is explicitly designed to verify if the DEBUG level logging is activated for a specific logger instance. Utilizing isDebugEnabled() allows developers to conditionally execute logging code based on the logging level configuration, thus optimizing performance by avoiding unnecessary string concatenation or complex object creations when debugging is not enabled.

Evaluation of Incorrect Options:

B. EnableDebug(): This is not a standard method in logging frameworks; it suggests an action rather than a check. Logging frameworks typically do not have methods that enable logging dynamically during runtime, as logging configuration is typically handled externally or at application startup.

C. IsEnableDebug(): The naming convention here is not consistent with established logging APIs. Java method names usually employ camelCase, and the suggested method form does not exist in common use, leading to potential misunderstandings and misimplementations in code.

D. DebugEnabled(): While this may seem similar, the absence of a preceding "is" indicates it does not follow the standard getters format employed in Java. This could lead to confusion as it is not a recognized method in the logging context, potentially resulting in compilation errors.

For further reading on Java logging practices, refer to the following authoritative resources:

SLF4J API Documentation
Log4j Documentation
Java Logging Framework Comparison

Question: 8

Which of the following elements in web.xml file ensures that cookies will be transmitted over an encrypted channel?

A. < connector IsSSLEnabled="Yes" / >
B. < connector EnableSSL="true" / >
C. < connector SSLEnabled="false" / >
D. < connector SSLEnabled="true" / >

Answer: D

Explanation:

D is correct because setting <connector SSLEnabled="true" /> ensures that all communications, including cookies, are securely transmitted over HTTPS, thereby enhancing web application security.

In web applications, cookies often hold sensitive session data. Transport Layer Security (TLS) protects this data from interception during transmission. By enabling SSL, the application forces secure communication, thus preventing cookies from being exposed to potential middleman attacks (MITM). This allows secure cookies to be transmitted, promoting adherence to secure coding practices.

Evaluation of Incorrect Options:

A. <connector IsSSLEnabled="Yes" />: This option is invalid as "IsSSLEnabled" is not a standard attribute in the context of web.xml connectors. Consequently, it would exert no effect on the transmission of cookies.

B. <connector EnableSSL="true" />: Similar to option A, "EnableSSL" is not a recognized standard attribute. Its use would not result in the enabling of SSL encryption, therefore failing to ensure secure cookie transmission.

C. <connector SSLEnabled="false" />: This explicitly disables SSL, rendering all communications, including cookies, unencrypted. Such a configuration contradicts best practices for securing sensitive data in transit.

For robust web application security, enabling SSL/TLS through legitimate configurations must be prioritized.

References:

https://owasp.org/www-project-secure-coding-practices-1-0/ (OWASP: Secure Coding Practices)
https://docs.oracle.com/cd/E19776-01/821-1755/abqcj/index.html (Oracle: Using SSL with Tomcat)
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet (OWASP: Transport Layer Protection Cheat Sheet)

Question: 9

In which phase of secure development lifecycle the threat modeling is performed?

A. Coding phase
B. Testing phase
C. Deployment phase
D. Design phase

Answer: D

Explanation:

D is correct because threat modeling is intrinsically linked to the design phase, where potential security threats are identified and mitigated before development begins.

In the secure development lifecycle (SDLC), the design phase is critical for establishing a security foundation. Threat modeling allows teams to evaluate potential vulnerabilities by systematically identifying assets, potential threats, and risk factors associated with the software design. This proactive approach ensures that security considerations are embedded into architectural decisions, enabling the development team to design countermeasures effectively.

Evaluation of Other Options:

A. **Coding Phase:** At this stage, developers implement the application based on the design specifications. While security practices like secure coding should be applied, it is not the appropriate time to conduct threat modeling, which precedes coding to inform developers of possible risks.

B. **Testing Phase:** Testing focuses on validating functionalities and ensuring conformity to requirements. Although vulnerability assessment and penetration testing may occur during this phase, threat modeling aims to address security issues early in the SDLC, making it ineffective to conduct in this phase.

C. **Deployment Phase:** Threat modeling is not applicable here as the application is already in production. This stage is geared towards ensuring that the application operates correctly and securely in the live environment, rather than identifying and addressing initial design vulnerabilities.

In summary, incorporating threat modeling during the design phase significantly enhances security posture by enabling identification and mitigation of risks before they can be exploited in later stages.

< input type="Submit" Id="Btn_Search" value="Search" / >
< /form >
However, when the application went to security testing phase, the security tester found an XSS vulnerability on this page. How can he mitigate the XSS vulnerability on this page?

A. He should write code like out.Write ("You Searched for: " +ESAPI.encoder().encodeForHTML(search));
B. He should write code like out.write ("You Searched for: " + request.getParameter("search").toString());
C. He should write code like out.write ("You Searched for: " + request.getParameter("txt_Search"));
D. He should write code like out.write (("You Searched for: " +(search));

Answer: A

Explanation:

A is correct because it employs proper output encoding that effectively mitigates XSS vulnerabilities by ensuring that any user-supplied input is safely represented in HTML.

Reasoning:

1. **Output Encoding:** Option A utilizes ESAPI.encoder().encodeForHTML(), which is a robust encoder specifically designed to handle user-generated content. This function transforms characters that could be interpreted as HTML (like <, >, &, etc.) into their corresponding HTML entities, preventing malicious scripts from executing.
2. **Security Best Practices:** Following the principle of least privilege and ensuring data sanitization at the output stage is crucial in web application security. By encoding outputs, the application effectively minimizes risks associated with XSS, a common vulnerability that results from improper handling of user inputs.

Critical Evaluation of Other Options:

B: out.write ("You Searched for: " + request.getParameter("search").toString());

This option directly outputs user input without encoding, exposing the application to XSS attacks if the input contains malicious scripts.

C: out.write ("You Searched for: " + request.getParameter("txt_Search"));

Similar to option B, this approach lacks any form of sanitization. It directly renders the user input in the response, making the application vulnerable to XSS attacks.

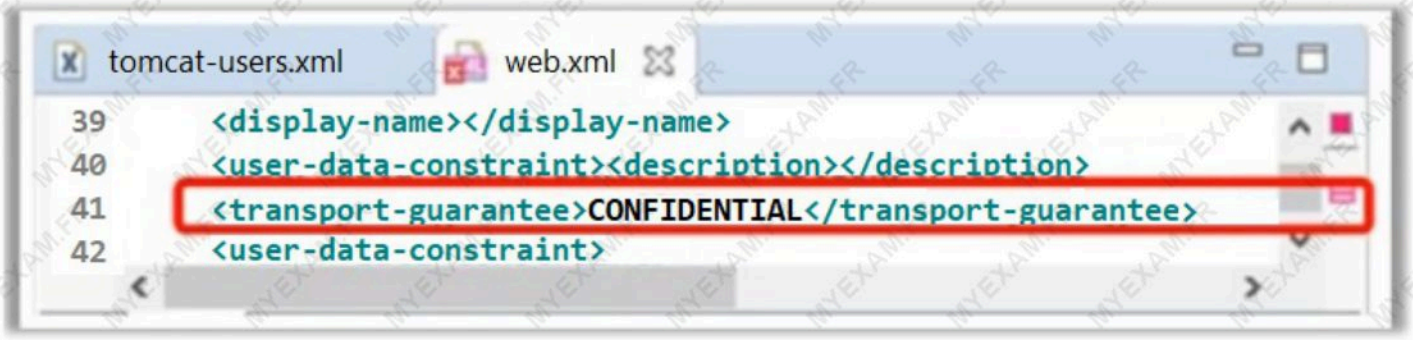
D: out.write (("You Searched for: " +(search));

This option appears to reference a variable search without prior context of its initialization or sanitization. If search contains unsanitized user data, it remains vulnerable to XSS.

Each of the incorrect options fails to encode or sanitize user inputs before rendering them in the HTML output, demonstrating a fundamental oversight in web security practices.

References:

https://owasp.org/www-community/attacks/xss/
https://www.owasp.org/index.php/Cross-site_scripting_(XSS)
https://www.owasp.org/index.php/ESAPI



A. He wants to transfer the entire data over encrypted channel

B. He wants to transfer only response parameter data over encrypted channel

C. He wants to transfer only request parameter data over encrypted channel

D. He wants to transfer only Session cookies over encrypted channel

Answer: A

Question: 14

Alice works as a Java developer in Fygo software Services Ltd. He is given the responsibility to design a bookstore website for one of their clients. This website is supposed to store articles in .pdf format. Alice is advised by his superior to design ArticlesList.jsp page in such a way that it should display a list of all the articles in one page and should send a selected filename as a query string to redirect users to articledetails.jsp page.
Alice wrote the following code on page load to read the file name.

```
String myfilename = request.getParameter("filename");
String txtFileNameVariable = myfilename;
String locationVariable = request.getServletContext().getRealPath("/");
String PathVariable = "";
PathVariable = locationVariable + txtFileNameVariable;
BufferedInputStream bufferedInputStream = null;
Path filepath = Paths.get(PathVariable);
```

After reviewing this code, his superior pointed out the security mistake in the code and instructed him not repeat the same in future. Can you point the type of vulnerability that may exist in the above code?

A. URL Tampering vulnerability

B. Form Tampering vulnerability

C. XSS vulnerability

D. Directory Traversal vulnerability

Answer: D

Explanation:

D. Directory Traversal vulnerability is correct because the code does not adequately validate the input filename, potentially allowing attackers to manipulate the path and access unintended files on the server.

In the provided code, user input from the query string is directly concatenated to the file path without proper validation or sanitization. An attacker could exploit this lack of validation by inserting special characters like ../, which can allow them to traverse the directory structure and access sensitive files outside the intended directory. This could lead to unauthorized file access, allowing exposure of critical information within the application or even the host system.

mechanisms, allowing unauthorized access. While the lockout feature aims to prevent such access, it can still be considered part of proper authentication methods, rather than a flaw or attack vector.

C. **Unvalidated Redirects and Forwards** - This type of vulnerability involves redirecting users to untrusted sites based on unvalidated content. It does not connect to the account lockout mechanism or its potential abuse, as it focuses on navigating away from legitimate user sessions rather than disrupting those sessions.

The potential for a DoS attack through account lockout should prompt organizations to consider adaptive security measures, such as increasing the threshold for lockouts or implementing CAPTCHA solutions after multiple failed attempts.

References:

https://owasp.org/www-project-top-ten/2017/A05_2017-Broken_Authentication.html
https://www.acs.com.hk/en/products/1975/acr35-ux-contacts-card-reader-with-ccid-support/
https://www.cloudflare.com/learning/security/glossary/denial-of-service-do-s-attack/