Zoho Practice Questions 3

Java — 15 Code Review Questions

---

1. SQL Injection

```
String q = "SELECT * FROM users WHERE id = " + request.getParameter("id");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(q);
```

👉 Bug: SQL injection via concatenation.
✅ Fix: Use `PreparedStatement`.

---

2. Hardcoded Credentials

```
String dbUser = "admin";
String dbPass = "password123";
```

👉 Bug: Hardcoded creds.
✅ Fix: Store in env vars / vault.

---

3. Insecure Password Storage

```
MessageDigest md = MessageDigest.getInstance("MD5");
md.update(password.getBytes());
```

👉 Bug: MD5 is weak.
✅ Fix: Use bcrypt/argon2.

---

4. Cross-Site Scripting (XSS)

```
<%= request.getParameter("username") %>
```

👉 Bug: Reflecting unescaped input.
✅ Fix: Encode with `<c:out>` or OWASP encoder.

---

5. Path Traversal

```
File f = new File("/uploads/" + request.getParameter("file"));
```

👉 Bug: Attacker can pass `../../etc/passwd`.
✅ Fix: Validate filename, canonicalize path.

---

6. Insecure Deserialization

```
ObjectInputStream ois = new ObjectInputStream(req.getInputStream());
Object obj = ois.readObject();
```

👉 Bug: Remote Code Execution.
✅ Fix: Avoid native deserialization, use JSON with schema validation.

---

7. Debug Info Leakage

```
e.printStackTrace(response.getWriter());
```

👉 Bug: Sends stack trace to user.
✅ Fix: Log server-side, show generic error to user.

---

8. CSRF Missing Token

```
<form action="/transferMoney" method="post">
   <input type="text" name="amount"/>
</form>
```

👉 Bug: No CSRF token.
✅ Fix: Add CSRF token hidden field, validate server-side.

---

9. Session Fixation

```
// Login but keeps old session
session.setAttribute("user", username);
```

👉 Bug: Old session reused.
✅ Fix: `request.changeSessionId()` after login.

---

10. Insecure Cookies

```
// Vulnerable: cookie not secure or HttpOnly
Cookie c = new Cookie("JSESSIONID", sessionId);
c.setHttpOnly(false);  // accessible via JavaScript (XSS risk)
c.setSecure(false);    // sent over HTTP (risk of MITM)
response.addCookie(c);
```

Bug:

- `HttpOnly=false` → JavaScript can read the cookie → XSS can steal session.
- `Secure=false` → cookie can be sent over unencrypted HTTP → session hijacking.

Fix (secure version):

```
Cookie c = new Cookie("JSESSIONID", sessionId);
c.setHttpOnly(true);   // prevent access via JS
c.setSecure(true);     // only sent over HTTPS
c.setPath("/");        // restrict to application path
c.setMaxAge(3600);     // optional: expire after 1 hour
response.addCookie(c);
```

Notes:

- Always use `HttpOnly + Secure` flags in production.
- Optionally use `SameSite=Lax` or `Strict` to mitigate CSRF:

```
c.setComment("SameSite=Lax"); // for servlet 4.0+, use newer APIs to set
SameSite
```

Continuing Java code review questions (11–15) and then move to Python and C/C++:

---

11. Open Redirect

```
String redirect = request.getParameter("url");
response.sendRedirect(redirect);
```

👉 Bug: Attacker can redirect to malicious sites.
✅ Fix: Whitelist allowed URLs or use relative paths.

---

12. Broken Access Control (IDOR)

```
String userId = request.getParameter("userId");
User u = db.getUser(userId);
```

👉 Bug: User can access other users' data.
✅ Fix: Check `currentUser.id == userId` or proper role check.

---

13. Logging Sensitive Data

```
logger.info("User password: " + password);
```

👉 Bug: Logs sensitive info.
✅ Fix: Do not log passwords, use masked logs.

---

14. Using Outdated Library

```
import org.apache.commons.codec.binary.Base64; // old version with CVE
```

👉 Bug: Vulnerable dependency.
✅ Fix: Update to latest secure library.

---

15. Improper Input Validation

```
int age = Integer.parseInt(request.getParameter("age"));
```

👉 Bug: May crash on non-integer input or injection.
✅ Fix: Validate input type and range.

🔹 Python — 15 Code Review Questions

---

1. SQL Injection

```
cur.execute("SELECT * FROM users WHERE id = " + user_id)
```

✅ Fix: `cur.execute("SELECT * FROM users WHERE id = %s", (user_id,))`

---

2. Command Injection

```
os.system("ping " + ip)
```

✅ Fix: Use `subprocess.run(["ping", ip], check=True)` and validate IP.

---

3. Insecure Deserialization

```
obj = pickle.loads(request.data)
```

✅ Fix: Use `json.loads` with schema validation.

---

4. Debug Mode in Production

```
app.run(debug=True)
```

✅ Fix: Set `debug=False` in production.

---

5. XSS in Template

```
return f"<h1>Hello {user_input}</h1>"
```

✅ Fix: Use template engine with autoescape (Jinja2).

---

6. Hardcoded Secret

```
API_KEY = "abcd1234"
```

✅ Fix: Use environment variables.

---

7. Weak Hashing

```
hashlib.sha1(password.encode()).hexdigest()
```

✅ Fix: Use `bcrypt` or `argon2`.

---

8. Path Traversal

```
open("/uploads/" + filename)
```

✅ Fix: Validate filename, remove `../`, or use safe directory APIs.

---

9. Logging Sensitive Info

```
logging.info("User token: %s" % token)
```

✅ Fix: Mask or avoid logging sensitive info.

---

10. Open Redirect

```
return redirect(request.args.get("next"))
```

✅ Fix: Only allow whitelisted URLs or relative paths.

---

11. Broken Access Control (IDOR)

```
user_id = request.args.get("user_id")
user = db.get_user(user_id)
```

✅ Fix: Only fetch `current_user.id` or validate roles.

---

12. Insecure Cookie

```
resp.set_cookie("sessionid", session_id, httponly=False, secure=False)
```

✅ Fix: `httponly=True`, `secure=True`.

---

13. Excessive Permissions

```
os.chmod("/data/file.txt", 0o777)
```

✅ Fix: Restrict to least permissions: `0o600`.

---

14. Using eval on untrusted input

```
result = eval(request.form["expr"])
```

✅ Fix: Remove eval; use safe parser.

---

15. Insecure File Upload

```
file.save("/uploads/" + file.filename)
```

✅ Fix: Validate extension, sanitize filename, store outside web root.

🔹 C / C++ — 15 Code Review Questions

---

1. Buffer Overflow

```
char buf[32];
gets(buf);
```

✅ Fix: Use `fgets(buf, sizeof(buf), stdin)`

---

2. Format String

```
printf(user_input);
```

✅ Fix: `printf("%s", user_input)`

---

3. Heap Overflow / Use-After-Free

```
char *p = new char[10];
delete [] p;
strcpy(p, "overflow");
```

✅ Fix: Do not use freed memory; bounds check.

---

4. Command Injection

```
system(("ls " + userInput).c_str());
```

✅ Fix: Validate input; avoid system(), use native API.

---

5. Path Traversal

```
ifstream f("/uploads/" + filename);
```

✅ Fix: Sanitize filename, canonicalize path.

---

6. Integer Overflow

```
int size = a * b; // a and b from user
char buf[size];
```

✅ Fix: Check multiplication overflow, validate inputs.

7. Memory Leak

```
char* p = new char[100];
// no delete
```

✅ Fix: delete[] p after use or use smart pointers.

---

8. Stack Overflow (Recursion)

```
void recurse() { recurse(); }
```

✅ Fix: limit recursion depth, iterative alternative.

---

9. Uninitialized Variable

```
int x;
if(flag) { printf("%d", x); }
```

✅ Fix: Initialize variables before use.

---

10. Insecure Randomness

```
srand(time(NULL));
int token = rand();
```

✅ Fix: Use `std::random_device` or crypto PRNG.

---

11. Double Free

```
delete p;
delete p;
```

✅ Fix: Set pointer to `nullptr` after delete.

---

12. Information Leakage

```
cout << secret;
```

✅ Fix: Do not print sensitive data.

---

13. Race Condition

```
if(access(file, W_OK) == 0) remove(file);
```

✅ Fix: Use atomic operations or locks.

---

14. Hardcoded Password

```
char pw[] = "admin123";
```

✅ Fix: Fetch from secure source / vault.

---

15. Insecure Socket / Plaintext

```
send(sock, data, strlen(data), 0);
```

✅ Fix: Use TLS/SSL wrapper.
