```
                        break;
                case "smtp":
                        portNum = 25;
                        break;
                default:
                        portNum = 80;
                }
                try {
                        ServerSocket serverSocket = new ServerSocket(portNum);
                } catch (Exception e) {
                        System.err.println("Caught Exception: " + e.getMessage());
                }
        }
}
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

### 7.2.7 SQL injection

**Come riconoscerla**

Si verifica quando l'input non verificato viene utilizzato per comporre dinamicamente uno statement SQL che poi verrà eseguito sulla base dati. Adeguatamente manipolati, i parametri di input possono modificare le query in maniera sostanziale, causando danni di impatto notevole, come l'inserimento di dati malevoli, la cancellazione e la modifica di record e la rivelazione indebita di informazioni riservate. Se i dati utilizzati per la SQL injection sono memorizzati nel database o nel file system in generale, si parla di SQL injection di second'ordine (second order SQL injection).

**Come difendersi**

- Come prima misura, occorre validare l'input, sottoponendolo a rigidi controlli, come già illustrato nei punti precedenti;
- Le query SQL non devono mai essere realizzate concatenando stringhe con l'input esterno. Si devono invece utilizzare componenti di database sicuri come le stored procedure, le query parametrizzate e le associazioni degli oggetti (per comandi e parametri;
- Una soluzione che può essere d'aiuto consiste nell'utilizzazione di una libreria ORM, come EntityFramework, Hibernate o iBatis;
- Occorre limitare l'accesso agli oggetti e alle funzionalità del database, in base al "Principle of Least Privilege" (non fornire agli utenti permessi superiori a quelli strettamente necessari.

Esempio:
Codice vulnerabile

```
String q='SELECT r FROM User r where r.userId=''+user+''';
Query query=em.createQuery(q);
List users=query.getResultList();
```

Codice sicuro

```
Query query=em.createNamedQuery('User.findByUserId');
query.setParameter('userId', user);
List users=query.getResultList();
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

### 7.2.8 XPath injection

**Come riconoscerla**

Si ha quando l'applicazione interroga un documento xml usando una query XPath testuale, creata concatenando dinamicamente le istruzioni con stringhe provenienti dall'esterno. L'attaccante potrebbe immettere una stringa che modifica la query XPath, ottenendo dal documento xml informazioni non dovute. Se l'input è stato manipolato ad arte, durante l'esecuzione dell'applicazione parti del documento xml, che non dovevano essere raggiunte, vengono indebitamente estratte e lette.
La gravità dell'attacco dipende dal tipo di dati che è possibile estrarre dal documento xml. Se contiene dati personali riservati, il furto di informazioni può realizzare un data breach; nel caso di dati account, l'attacco può prefigurare ulteriori attacchi di spoofing ed elevation of privileges.

**Come difendersi**

Come prima cosa, occorre procedere con la validazione dell'input, come in tutti i casi di injection, adottando le precauzioni illustrate nei punti precedenti, tra le quali la depurazione della stringa da tutti i caratteri potenzialmente dannosi. La'adozione di una white list di valori ammessi è sempre un'ottima soluzione.

Evitare che la costruzione della query XPath sia dipendente dalle informazioni inserite dall'utente. Si deve mappare la query di tipo XPath con i parametri utente mantenendo la separazione tra dati e codice. Nel caso fosse necessario includere l'input dell'utente nella query, l'input stesso dovrà essere prima validato correttamente.

Esempio:

```
public class XPath_Injection {
   public static void main(String[] args) {
          Scanner userInputScanner = new Scanner(System.in);
          System.out.print("\nEnter XPath expression: ");
          String expression = userInputScanner.nextLine();

          // read a string value
          XPath XPath = XPathFactory.newInstance().newXPath();
          try {
                XPathExpression email = XPath.compile(expression);
          } catch (XPathExpressionException e) {
                e.printStackTrace();
          }
   }
}
```

L'input dell'utente deve essere ricondotto a valori ammessi (white list):

```
public class XPath_Injection_Fixed {
   public static void main(String[] args) {
          HashMap<String, String> sanitize = new HashMap<String, String>();
          sanitize.put("student", "/class/student");
          sanitize.put("graduate", "/class/graduate");
          sanitize.put("professor", "/class/professor");
          Scanner userInputScanner = new Scanner(System.in);
          System.out.print("\nEnter XPath expression: ");
          String expression = userInputScanner.nextLine();

          // read a string value
          XPath XPath = XPathFactory.newInstance().newXPath();
          try {
                XPathExpression email = XPath.compile(sanitize.get(expression));
          } catch (XPathExpressionException e) {
                e.printStackTrace();
          }
   }
}
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/643.html,
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').