SI (상)	SQL 인젝션
취약점 개요	
점검목적	■ 대화형 웹 사이트에 비정상적인 사용자 입력 값 허용을 차단하여 악의적인 데이터베이스 접근 및 조작을 방지하기 위함
보안위협	■ 해당 취약점이 존재하는 경우 비정상적인 SQL 쿼리로 DBMS 및 데이터(Data)를 열람하거나 조작 가능하므로 사용자의 입력 값에 대한 필터링을 구현하여야 함
참고	※ **SQL인젝션:** 사용자의 입력 값으로 웹 사이트 SQL 쿼리가 완성되는 약점을 이용하며, 입력 값을 변조하여 비정상적인 SQL 쿼리를 조합하거나 실행하는 공격. 개발자가생각지 못한 SQL문을 실행되게 함으로써 데이터베이스를 비정상적으로 조작 가능함 ※ SQL인젝션 공격 관련 코드 검토 필요 ※ 소스코드 및 취약점 점검 필요
조치방법	소스코드에 SQL 쿼리를 입력 값으로 받는 함수나 코드를 사용할 경우, 임의의 SQL 쿼리 입력에 대한 검증 로직을 구현하여 서버에 검증되지 않는 SQL 쿼리 요청 시 에러 페이지가 아닌 정상 페이지가 반환되도록 필터링 처리하고 웹 방화벽에 SQL 인젝션 관련 룰셋을 적용하여 SQL 인젝션 공격을 차단함

보안설정 방법

* SQL 쿼리에 사용되는 문자열의 유효성을 검증하는 로직 구현
* 아래와 같은 특수문자를 사용자 입력 값으로 지정 금지
  (아래 문자들은 해당 데이터베이스에 따라 달라질 수 있음)

문자	설명
`	문자 데이터 구분 기호
;	쿼리 구분 기호
--, #	해당 라인 주석 구분 기호
/* */	* 와 */ 사이 구문 주석

* Dynamic SQL 구문 사용을 지양하며 파라미터에 문자열 검사 필수적용
* 시스템에서 제공하는 에러 메시지 및 DBMS에서 제공하는 에러 코드가 노출되지 않도록 예외처리
* 웹 방화벽에 인젝션 공격 관련

■ ASP.net

* 문자열 유효성 검증 로직 구현

(예) 특정 문자열 필터링 적용 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

request로 입력 값을 가져오는 경우 입력 값에서 특수문자를 제거하여 바인딩하는 소스 삽입

replaceAll() 메소드를 사용하여 구현

```
private string SafeSqlLiteral(string inputSQL)
{
   Str = inputSQL.Replace("","''");
   Str = str. Replace(";","");
   Str = str. Replace("--","");
   Str = str. Replace("|","");
   Str = str. Replace(":","");
   Str = str. Replace("+","");
   Str = str. Replace("₩","");
   Str = str. Replace("/","");
   .....
   return str;
}
```

* Dynamic SQL

```
Private void cmdLogin_Click(object sender, System.EventArgs e) {
   string strCnx = ConfigurationSettings.AppSettings["cnxNWindBad"];
   Using (SqlConnection cnx = new SqlConnection(strCnx))
   {
      SqlParameter prm;
      Cnx.Open();
      string strQry =
      "SELECT Count(*) FROM Users WHERE UserName = @username " +
      "AND Password = @password";
      Int intRecs;
      SqlCommand cmd = new SqlCommand(strQry, cnx);
      cmd.CommandType = CommandType.Text;
      prm = new SqlParameter("@username",SqlDbType.VarChar,50);
      prm.Direction = ParameterDirection.Input;
      prm.Value = txtUser.Text;
      cmd.Parameters.Add(prm);
      prm = new SqlParameter("@password",SqlDbType.VarChar,50);
      prm.Direction = ParameterDirection.Input;
      prm.Value = txtPassword.Text;
      cmd.Parameters.Add(prm);
      intRecs = (int) cmd.ExecuteScalar();
      if(intRecs > 0) {
         FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false);
      }
      else {
         lblMsg.Text = "Login attempt failed.";
      }
   }
}
```

■ **JSP**

* 문자열 유효성 검증 로직 구현

(예) 특정 문자열 필터링 적용 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

request로 입력 값을 가져오는 경우 입력 값에서 특수문자를 제거하여 바인딩하는 소스 삽입

replaceAll() 메소드를 사용하여 구현

```
public static String makeQuery(String str) {
   String result = "";
   if(str != null) {
      result = chkNull(replace(str, "'", ""));
      result = chkNull(replace(str, ";", ""));
      result = chkNull(replace(str, "--", ""));
      result = chkNull(replace(str, "|", ""));
      result = chkNull(replace(str, ":", ""));
      result = chkNull(replace(str, "+", ""));
      result = chkNull(replace(str, "₩", ""));
      result = chkNull(replace(str, "/", ""));
      result = chkNull(replace(str.toLowerCase(), "select", ""));
      result = chkNull(replace(str.toLowerCase(), "update", ""));
      result = chkNull(replace(str.toLowerCase(), "delete", ""));
      result = chkNull(replace(str.toLowerCase(), "insert", ""));
      result = chkNull(replace(str.toLowerCase(), "where", ""));
      result = chkNull(replace(str.toLowerCase(), "from", ""));
      result = "'"+result+"'";
    }
   return result;
 }
public static String chkNull(String str) {
   if (str == null)
   return "";
   else
   return str;
}
```

* Dynamic SQL 구문 사용 금지

(예1) PreparedStatement 객체 사용 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

```
try{
   String tableName = props.getProperty("jdbc.tableName");
   String name = props.getProperty("jdbc.name")
   String qury = "SELECT * FROM ? WHERE Name = ?";
   stmt = con.perpareStatement(query);
   stmt.setString(1, tableName);
   stmt.setString(2, name);
   rs = stmt.executeQuery();
   .....
}
catch (SQLException sqle){ }
finally { }
```

(예2) JDO API 사용 시 외부 입력 값이 위치하는 부분을 "?"로 설정하여 실행 시 해당 파라미터가 실행되도록 수정 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

```
try{
   Properties props = new Properties();
   String filename = "contacts.txt";
   FileInputStream in = new FileInputStream(filename);
   Props.load(in);
   name = props.getProperty("name");
   if (name = null || "".equals(name)) return null;
   query += " where name = ?";
}
catch (IOException e)
{
   Javax.jdo.Query q = pm.newQuery(query);
```

(예3) J2EE Persistence API 사용 시 파라미터를 받는 쿼리를 생성하고 파라미터를 설정하여 실행 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

```
try{
   Properties props = new Properties();
   String filename = "contacts.txt";
   FileInputStream in = new FileInputStream(filename);
   Props.load(in);
   String id = props.getProperty("id");
   If (id == null || "".equals(id)) id = "itemid";
   Query query = em.createNativeQuery("Select OBJECT(i) from Item I where i.itemID > :id");
   Query.setParameter("id", id);
   .....
}
```

(예4) mybatis Data Map 사용 시 쿼리에 삽입되는 Name 파라미터를 #name# 형태로 받아 실행 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

```
<?xml version="1.0" encoding="UTF-8"?>
   .....
<!-- static SQL 사용 -->
<delete id="delStudent" parameterClass="Student">
   DELETE STUDENTS
   WHERE NUM = #num# AND Name = '#name#'
</delete>
```

■ **PHP**

* 문자열 유효성 검증 로직 구현

(예1) addslashes 함수를 이용한 특정 문자열 필터링 적용 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

```
$query = sprintf("SELECT id,password,username FROM user_table WHERE_
id='%s';",addslashes($id));
// id 변수를 문자형으로 받고, id 변수의 특수문자를 일반문자로 변환
// @로 php 에러 메시지를 막음
$result = @OCIParse($conn, $query);
if (!@OCIExecute($result))
error("SQL 구문 에러");
exit;
@OCIFetchInto($result,&$rows);
... 중략 ...
```

(예2) eregi_replace 함수를 이용한 특정 문자열 필터링 적용 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

```
function SQL_Injection($get_Str) {
   return eregi_replace("( select| union| insert| update| delete| drop|₩"||₩/₩*|₩*₩/|₩₩₩|₩;)","",
            $get_Str);
}
```

(예3) php.ini 설정 중 magic_quotes_gps 옵션을 이용하여 특정 문자열 필터링 적용
GPC(Get, Post, Cookie)를 통해 넘어오는 문자열 중 ', ", ₩, NULL 값의 앞에 자동으로 백슬래쉬 문자를 붙여주는 기능을 함 (PHP 6.0 이후 버전 사용 불능)

```
magic_quotes_gpc = on
```

* Dynamic SQL 구문 사용 금지
(예1) Static SQL 구문 사용 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

```
$sql = 'SELECT ID, PASSWORD, USER_NAME FORM DB WHERE VALUES = ? ';
$stmt = $mysqli->prepare($sql);
$stmt->bind_param('s', '1');
$stmt->execute();
$stmt->bind_result($ID, $PASSWORD, $USER_NAME); // 칼럼수만큼 변수로 지정
while($stmt->fetch()) {
   printf("%s %s₩n", $ID, $PASSWORD, $USER_NAME);
}
$stmt->close();
$mysqli->close();
```

(예2) mybatis Data Map 사용 시 쿼리에 삽입되는 Name 파라미터를 #name# 형태로 받아 실행 (※ 예로 제시한 것으로, 구현 시 다를 수 있음)

```
<?xml version="1.0" encoding="UTF-8"?>
   .....
<!-- static SQL 사용 -->
<delete id="delStudent" parameterClass="Student">
   DELETE STUDENTS
   WHERE NUM = #num# AND Name = '#name#'
</delete>
```