常见 Web 安全漏洞及其修复

GeoffreyWang

August 19, 2024

1 SQL 注入 (SQL Injection)

1.1 漏洞描述

SQL 注入是一种通过将恶意 SQL 代码注入到查询语句中，来操控数据库的攻击方式。这种攻击通常利用了应用程序直接将用户输入嵌入到 SQL 查询中的漏洞。

1.2 攻击方式

攻击者可能通过在输入字段中插入恶意代码，例如输入 'admin' OR '1'='1'，使得 SQL 查询返回更多数据，甚至可能泄露敏感信息。

1.3 不安全的代码示例

```python
import sqlite3

def get_user_data(username):
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    query = f"SELECT * FROM users WHERE username = '{username}'"
    cursor.execute(query)
    return cursor.fetchall()
```

Listing 1: 存在 SQL 注入漏洞的代码

1.4 修复方案

应使用参数化查询来避免直接将用户输入拼接到 SQL 查询中，从而防止 SQL 注入攻击。

1.5 修正后的代码示例

```python
def get_user_data(username):
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE username = ?"
    cursor.execute(query, (username,))
    return cursor.fetchall()
```

Listing 2: 使用参数化查询的安全代码

2 跨站脚本攻击 (XSS)

2.1 漏洞描述

跨站脚本攻击（XSS）是一种通过在网页中注入恶意脚本，从而在其他用户浏览网页时执行这些脚本的攻击方式。这可能导致用户的敏感信息泄露，甚至控制用户的会话。

2.2 攻击方式

攻击者可能在评论或输入字段中注入类似 '<script>alert('Hacked!');</script>'的恶意 JavaScript 代码。

2.3 不安全的代码示例

```python
def display_comment(comment):
    return f"<p>{comment}</p>"
```

Listing 3: 存在 XSS 漏洞的代码

2.4 修复方案

应对用户输入进行 HTML 转义，以防止注入的脚本被浏览器执行。

2.5 修正后的代码示例

```python
import html

def display_comment(comment):
    safe_comment = html.escape(comment)
    return f"<p>{safe_comment}</p>"
```

Listing 4: 经过 HTML 转义的安全代码

3 命令注入 (Command Injection)

3.1 漏洞描述

命令注入是一种通过将恶意输入注入系统命令中，来执行未经授权的命令或操作的攻击方式。这种漏洞通常发生在应用程序直接调用系统命令时。

3.2 攻击方式

攻击者可能输入类似 ‘; rm -rf /‘ 的命令，导致删除服务器上的所有文件。

3.3 不安全的代码示例

```
import os

def delete_file(filename):
    os.system(f"rm {filename}")
```

Listing 5: 存在命令注入漏洞的代码

3.4 修复方案

应使用安全的命令处理方法，例如通过 ‘shlex.quote‘对用户输入进行处理，以防止输入被解释为多个命令。

3.5 修正后的代码示例

```
import os
import shlex

def delete_file(filename):
    safe_filename = shlex.quote(filename)
    os.system(f"rm {safe_filename}")
```

Listing 6: 经过安全处理的命令执行代码

4 文件上传漏洞

4.1 漏洞描述

文件上传漏洞是指攻击者通过上传恶意文件，并使其在服务器上执行的攻击方式。这可能导致服务器被攻击者控制或敏感数据泄露。

4.2 攻击方式

攻击者可能上传一个包含恶意代码的文件，例如 PHP 脚本，然后访问该文件以执行代码。

4.3 不安全的代码示例

```
def save_uploaded_file(file):
    with open(f"/uploads/{file.filename}", "wb") as f:
        f.write(file.read())
```

Listing 7: 存在文件上传漏洞的代码

4.4 修复方案

应限制上传文件的类型，并确保文件名不包含路径信息，避免目录遍历攻击。

4.5 修正后的代码示例

```
import os

def save_uploaded_file(file):
    filename = os.path.basename(file.filename)
    if not filename.endswith(('.png', '.jpg', '.jpeg', '.
        gif', '.pdf')):
        raise ValueError("Unsupported file type")
    with open(f"/uploads/{filename}", "wb") as f:
        f.write(file.read())
```

Listing 8: 限制文件类型的安全代码

5 弱密码存储 (Insecure Password Storage)

5.1 漏洞描述

如果密码以明文存储，一旦数据库被攻击者获取，所有用户的密码将立即被泄露。这是一个严重的安全隐患。

5.2 攻击方式

攻击者通过获取数据库中的明文密码，可以直接使用这些密码登录用户的其他账户（如果用户在多个地方使用相同的密码）。

5.3 不安全的代码示例

```
def store_password(username, password):
    with open("passwords.txt", "a") as f:
        f.write(f"{username}:{password}\n")
```

Listing 9: 明文存储密码的代码

5.4 修复方案

应使用加密算法（如 bcrypt）对密码进行哈希存储，而不是明文存储。

5.5 修正后的代码示例

```
import bcrypt

def store_password(username, password):
    hashed = bcrypt.hashpw(password.encode(), bcrypt.
        gensalt())
    with open("passwords.txt", "a") as f:
        f.write(f"{username}:{hashed.decode()}\n")
```

Listing 10: 使用哈希函数存储密码的安全代码