SQL 与 NoSQL 数据库的区分及注入攻击防范

Geoffrey Wang

August 19, 2024

1 区分 SQL 与 NoSQL 数据库

1.1 SQL 数据库

SQL 数据库（关系型数据库）使用结构化查询语言（SQL）来管理和操作数据。常见的 SQL 数据库包括 MySQL、PostgreSQL、SQL Server 等。它们通常采用固定的表结构，通过联接（JOIN）等操作来实现复杂的查询。

1.2 NoSQL 数据库

NoSQL 数据库（非关系型数据库）不使用 SQL 作为查询语言，通常用于处理大规模的、非结构化的数据。NoSQL 数据库包括 MongoDB、CouchDB、Redis 等，支持灵活的数据模型，如键-值存储、文档存储、列族存储等。

1.3 如何区分 SQL 和 NoSQL 数据库

攻击者可以通过以下几种方式确定目标系统使用的是关系型数据库还是非关系型数据库：

- **错误消息**：如果应用程序在处理查询时返回详细的错误信息，攻击者可以通过这些信息推断数据库类型。
- **响应时间分析**：通过分析复杂查询的响应时间，可以推测数据库的特性。
- **URL 模式和请求格式**：分析应用程序的 API 或请求格式，推测其背后的数据库类型。
- **数据库功能测试**：通过发送特定的查询或操作，测试目标系统是否支持某种数据库特有的功能。
- **端口扫描**：使用工具扫描服务器的开放端口，识别常见的数据库服务。

2 SQL 注入 (SQL Injection)

2.1 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。

2.2 典型不安全代码示例

2.2.1 登录认证中的 SQL 注入

```
def login(username, password):
    query = f"SELECT * FROM users WHERE username = '{
        username}' AND password = '{password}'"
    cursor.execute(query)
    user = cursor.fetchone()
    if user:
        return "Login successful"
    else:
        return "Login failed"
```

Listing 1: 存在 SQL 注入漏洞的登录代码

2.2.2 基于 URL 参数的 SQL 注入

```
def get_product_details(product_id):
    query = f"SELECT * FROM products WHERE id = {
        product_id}"
    cursor.execute(query)
    return cursor.fetchone()
```

Listing 2: URL 参数导致的 SQL 注入

2.2.3 搜索功能中的 SQL 注入

```
def search_products(keyword):
    query = f"SELECT * FROM products WHERE name LIKE '%{
        keyword}%'"
    cursor.execute(query)
    return cursor.fetchall()
```

Listing 3: 搜索功能中的 SQL 注入

2.2.4 删除记录中的 SQL 注入

```
def delete_user(user_id):
    query = f"DELETE FROM users WHERE id = {user_id}"
    cursor.execute(query)
```

Listing 4: 删除记录中的 SQL 注入

2.2.5 报告生成中的 SQL 注入

```
def generate_report(start_date, end_date):
    query = f"SELECT * FROM sales WHERE date >= '{
        start_date}' AND date <= '{end_date}'"
    cursor.execute(query)
    return cursor.fetchall()
```

Listing 5: 报告生成中的 SQL 注入

2.2.6 批量更新中的 SQL 注入

```
def update_user_roles(user_ids, new_role):
    query = f"UPDATE users SET role = '{new_role}' WHERE
        id IN ({user_ids})"
    cursor.execute(query)
```

Listing 6: 批量更新中的 SQL 注入

2.2.7 插入数据中的 SQL 注入

```
def add_new_user(username, email, password):
    query = f"INSERT INTO users (username, email,
        password) VALUES ('{username}', '{email}', '{
        password}')"
    cursor.execute(query)
```

Listing 7: 插入数据中的 SQL 注入

2.3 如何防止 SQL 注入

为了防止 SQL 注入，推荐使用参数化查询、ORM 框架，以及对用户输入进行严格的验证和过滤。

3 NoSQL 注入 (NoSQL Injection)

3.1 漏洞描述

NoSQL 注入是一种通过操控 NoSQL 查询语句中的数据结构或逻辑，来攻击非关系型数据库的手段。这种攻击通常利用了应用程序在构建查询时对用户输入处理不当的漏洞。

3.2 典型不安全代码示例

3.2.1 MongoDB 中的 NoSQL 注入

```
def find_user(username, password):
    query = {"username": username, "password": password}
    user = db.users.find_one(query)
    return user
```

Listing 8: 存在 NoSQL 注入漏洞的代码

3.2.2 CouchDB 中的 NoSQL 注入

```
function getDocumentById(id) {
    return db.get(id);
}
```

Listing 9: 存在 NoSQL 注入漏洞的 CouchDB 代码

3.2.3 Redis 中的 NoSQL 注入

```
def get_cache_value(key):
    value = redis_client.get(key)
    return value
```

Listing 10: Redis 中的 NoSQL 注入

3.2.4 Cassandra 中的 NoSQL 注入

```
def get_user_activity(user_id):
    query = f"SELECT * FROM user_activities WHERE user_id
        = {user_id}"
    session.execute(query)
    return session.fetchall()
```

Listing 11: Cassandra 中的 NoSQL 注入

3.2.5 Firebase 中的 NoSQL 注入

```
function getUserData(userId) {
    return firebase.database().ref('/users/' + userId).
        once('value').then(function(snapshot) {
        return snapshot.val();
    });
}
```

Listing 12: Firebase 中的 NoSQL 注入

3.2.6 Elasticsearch 中的 NoSQL 注入

```
def search_documents(query):
    body = {
        "query": {
            "match": {
                "content": query
            }
        }
    }
    res = es.search(index="documents", body=body)
    return res['hits']['hits']
```

Listing 13: Elasticsearch 中的 NoSQL 注入

3.3 如何防止 NoSQL 注入

为了防止 NoSQL 注入，开发者应确保对所有用户输入进行严格的验证和过滤，使用安全的查询构建方法，并限制数据库的访问权限。

4 结论

SQL 和 NoSQL 数据库在应用中都有其独特的优势，但它们各自的注入攻击方式也有所不同。了解和防范这些攻击，有助于提升应用程序的安全性，保护用户数据。