SQL注入及XSS(跨站脚本)攻击防御技术方案

SQL注入

□□ 什么是SQL注入

SQL注入:利用现有应用程序,将(恶意)的SQL命令注入到后台数据库引擎执行的能力,这是SQL注入的标准释义。

SQL注入利用的是正常的HTTP服务端口,表面上看来和正常的web访问没有区别,隐蔽性极强,不易被发现。

□□ SQL注入的危害

SQL注入的主要危害包括:

1、未经授权状况下操作数据中的数据

2、恶意篡改网页内容

3、私自添加系统账号或是数据库使用者账号

4、网页挂木马。

□□ SQL注入的方法

1. **没有正确过滤转义字符**

在用户的输入没有为转义字符过滤时,就会发生这种形式的注入式攻击,它会被传递

给一个SQL语句。这样就会导致应用程序的终端用户对数据库上的语句实施操纵比方

说,下面的这行代码就会演示这种漏洞:

```
statement := "SELECT * FROM users WHERE name = '" + userName + "'; "
```

这种代码的设计目的是将一个特定的用户从其用户表中取出，但是，如果用户名被一个恶意的用户用一种特定的方式伪造，这个语句所执行的操作可能就不仅仅是代码的作者所期望的那样了。例如，将用户名变量(即username)设置为：

a’ or ’t’=’t，此时原始语句发生了变化：

SELECT * FROM users WHERE name = ’a’ OR ’t’=’t’;

如果这种代码被用于一个认证过程，那么这个例子就能够强迫选择一个合法的用户名，因为赋值’t’=’t永远是正确的。

在一些SQL服务器上，如在SQL□

Server中，任何一个SQL命令都可以通过这种方法被注入，包括执行多个语句。下面语句中的username的值将会导致删除“users”表，又可以从“data”表中选择所有的数据(实际上就是透露了每一个用户的信息)。

a’; DROP TABLE users; SELECT * FROM data WHERE name LIKE ’%

这就将最终的SQL语句变成下面这个样子：

SELECT * FROM users WHERE name = ’a’; DROP TABLE users; SELECT * FROM DATA WHERE name LIKE ’%’;

其它的SQL执行不会将执行同样查询中的多个命令作为一项安全措施。这会防止攻击者注入完全独立的查询，不过却不会阻止攻击者修改查询。

2.Incorrect type handling

如果一个用户提供的字段并非一个强类型，或者没有实施类型强制，就会发生这种形式的攻击。当在一个SQL语句中使用一个数字字段时，如果程序员没有检

查用户输入的合法性(是否为数字型)就会发生这种攻击。例如:

statement := "SELECT * FROM data WHERE id = " + a_variable + "; "

从这个语句可以看出,作者希望a_variable是一个与"id"字段有关的数字。不过,如果终端用户选择一个字符串,就绕过了对转义字符的需要。例如,将a_variable设置为:1; DROP TABLE users,它会将"users"表从数据库中删除,SQL语句变成:SELECT * FROM DATA WHERE id = 1; DROP TABLE users;

3.数据库服务器中的漏洞

有时,数据库服务器软件中也存在着漏洞,如MYSQL服务器中mysql_real_escape_string()函数漏洞。这种漏洞允许一个攻击者根据错误的统一字符编码执行一次成功的SQL注入式攻击。

4.盲目SQL注入式攻击

当一个Web应用程序易于遭受攻击而其结果对攻击者却不见时,就会发生所谓的盲目SQL注入式攻击。有漏洞的网页可能并不会显示数据,而是根据注入到合法语句中的逻辑语句的结果显示不同的内容。这种攻击相当耗时,因为必须为每一个获得的字节而精心构造一个新的语句。但是一旦漏洞的位置和目标信息的位置被确立以后,一种称为Absinthe的工具就可以使这种攻击自动化。

5.条件响应

注意，有一种SQL注入迫使数据库在一个普通的应用程序屏幕上计算一个逻辑语句的值：

SELECT booktitle FROM booklist WHERE bookId = 'OOk14cd' AND 1=1

这会导致一个标准的面面，而语句

SELECT booktitle FROM booklist WHERE bookId = 'OOk14cd' AND 1=2在页面易于受到SQL注入式攻击时，它有可能给出一个不同的结果。如此这般的一次注入将会证明盲目的SQL注入是可能的，它会使攻击者根据另外一个表中的某字段内容设计可以评判真伪的语句。

6.条件性差错

如果WHERE语句为真，这种类型的盲目SQL注入会迫使数据库评判一个引起错误的语句，从而导致一个SQL错误。例如：

SELECT 1/0 FROM users WHERE username='Ralph'。显然，如果用户Ralph存在的话，被零除将导致错误。

7.时间延误

时间延误是一种盲目的SQL注入，根据所注入的逻辑，它可以导致SQL引擎执行一个长队列或者是一个时间延误语句。攻击者可以衡量页面加载的时间，从而决定所注入的语句是否为真。

□□ SQL注入防御手段

1.使用参数化的过滤性语句

要防御SQL注入，用户的输入就绝对不能直接被嵌入到SQL语句中。恰恰相反，用

户的输入必须进行过滤，或者使用参数化的语句。参数化的语句使用参数而不是将用户输入嵌入到语句中。在多数情况中，SQL语句就得以修正。然后，用户输入就被限于一个参数。下面是一个使用Java和JDBC API例子：

```
PreparedStatement prep = conn.prepareStatement("SELECT * FROM USERS WHERE PASSWORD=?");

prep.setString(1, pwd);
```

总体上讲，有两种方法可以保证应用程序不易受到SQL注入的攻击，一是使用代码复查，二是强迫使用参数化语句的。强迫使用参数化的语句意味着嵌入用户输入的SQL语句在运行时将被拒绝。不过，目前支持这种特性的并不多。如H2数据库引擎就支持。

2.还要避免使用解释程序，因为这正是黑客们借以执行非法命令的手段。

3.防范SQL注入，还要避免出现一些详细的错误消息，因为黑客们可以利用这些消息。要使用一种标准的输入确认机制来验证所有的输入数据的长度、类型、语句、企业规则等。

4.使用专业的漏洞扫描工具。但防御SQL注入攻击也是不够的。

攻击者们目前正在自动搜索攻击目标并实施攻击。其技术甚至可以轻易地被应用于其它的Web架构中的漏洞。企业应当投资于一些专业的漏洞扫描工具，如大名鼎鼎的Acunetix的Web漏洞扫描程序等。一个完善的漏洞扫描程序不同于网络扫描程序，它专门查找网站上的SQL注入式漏洞。最新的漏洞扫描程序可以查找最新发现的漏洞。

5.最后一点，企业要在Web应用程序开发过程的所有阶段实施代码的安全检查。

首先，要在部署Web应用之前实施安全测试，这种措施的意义比以前更大、更深远。企业还应当在部署之后用漏洞扫描工具和站点监视工具对网站进行测试。

XSS(跨站脚本)攻击

□□ XSS攻击原理

XSS 属于被动式的攻击。攻击者先构造一个跨站页面，利用 script、<IMG>、FRAME>等各种方式使得用户浏览这个页面时，触发对被攻击站点的 http 请求。此时果被攻击者如果已经在被攻击站点登录，就会持有该站点 cookie。这样该站点会认为攻击者发起了一个 http 请求。而实际上这个请求是在被攻击者不知情的情况下发起，由此攻击者在一定程度上达到了冒充被攻击者的目的。精心的构造这个攻击请求，以达到冒充发文，夺取权限等等多个攻击目的。在常见的攻击实例中，这个请求是通 script 来发起的，因此被称为 Cross Site Script。

XSS 攻击主要分为两类:一类是来自内部的攻击，主要指的是利用 WEB 程序自身漏洞，提交特殊的字符串，从而使得跨站页面直接存在于被攻击站点上，这个字符串称为跨站语句。这一类攻击所利用的漏洞非常类似于 SQL Injection漏洞，都是 WEB

序没有对用户输入作充分的检查和过滤。另一类则是来来自外部的攻击，主要指的自己构造 XSS
跨站漏洞网页或者寻找非目机以外的有跨站漏洞的网页。如当我们要渗透一个站点，我们自己构造一个跨站网页在自己的服务器上，然后通过结合其它技术，如社会工程学等，欺骗目标服务器的管员打开。这一类攻击的威胁相对较低，至少 ajax要发起跨站调用是非常困难的。

□□ 常见XSS攻击手法

1、依赖跨站漏洞，需要在被攻击网站的页面种入脚本的。

(1)\Cookie 盗取，通过 javascript 获取被攻击网站种下的 cookie，并发送给攻击者，从 cookie 中提取密码等隐私 ，利用 cookie伪造 session，发起重放攻击。

(2)、Ajex信息盗取，通过 javascript 发起ajex请求。 从 ajex结果中获取隐私，模拟用户完成多页表单。

2、不依赖跨站漏洞的手法

(1)、单向 HTTP 动作，通过 img.src 等方法发起跨站访问，冒充被攻击者执行特权操作。
但是很难拿到服务器的返回值。

(2)、 双向 HTTP 动作，如果服务器产生一段动态的 script，那么可以用 script.src 的方法，发起跨站访问并拿到服务器的返回值。

□□ XSS攻击防范措施

1. **防堵跨站漏洞。**

阻止攻击者利用在被攻击网站上发布跨站攻击语句，不可以信任用户提交的任何内容，

首先代码里对用户输入的地方和变量都需要仔细检查长度和对”<”,”>”,”;”,””” 等字符做过滤;其次任何内容写到页面之前都必须加以encode，避免不小心把 html tag 弄出来。这一个层面做好，至少可以堵住超过一半的 XSS 攻击。

2. Cookie 防盗

首先避免直接在 cookie中泄露用户隐私，例如 email、密码等等。

其次通过使 cookie 和系统 ip 绑定来降低 cookie泄露后的危险。这样攻击者得到的 cookie 没有实际价值，不可能拿来重放。

3. 尽量采用 POST 而非 GET 提交表单

POST 操作不可能绕开 javascript 的使用，这会给攻击者增加难度，减少可利用的跨站漏洞。

4. 严格检查 refer

检查 http refer 是否来自预料中的 url。这可以阻止第 2 类攻击手法发起的 http 请求，也能防止大部分第 1 类攻击手法，除非正好在特权操作的引用页上种了跨站访问。

5. 将单步流程改为多步

在多步流程中引入效验码中每一步都产生一个验证码作为 hidden 表单元素嵌在中间 页面，下一步操作时这个验证码被提交到服务器，服务器检查这个验证码是否匹配。首先这为第 1

类攻击者大大增加了麻烦。其次攻击者必须在多步流程中拿到上一步产生的效验码才有可能发起下一步请求，这在第 2 类攻击中是几乎无法做到的。

6. 引入用户交互

简单的一个看图识数可以堵住几乎所有的非预期特权操作。

7. 只在允许 anonymous 访问的地方使用动态的 javascript。

8. 对于用户提交信息的中的 img 等 link，检查是否有重定向回本站、不是真的图
可疑操作。

9. 内部管理网站的问题

很多时候，内部管理网站往往疏于关注安全问题，只是简单的限制访问来源。这种
网站往往对 XSS 攻击毫无抵抗力，需要多加注意。

安全问题需要长期的关注，从来不是一锤子买卖。XSS
攻击相对其他攻击手段更加
隐蔽和多变，和业务流程、代码实现都有关系，不存在什么一劳永逸的解决方案。此外，
面对
XSS，往往要牺牲产品的便利性才能保证完全的安全，如何在安全和便利之间平衡
也是一件需要考虑的事情。