**WEB安全测试分类及防范测试方法**

[1 Web 应用程序布署环境测试 2](#_Toc367267367)

[1.1HTTP 请求引发漏洞的测试 2](#_Toc367267368)

[1.2 操作系统目录安全性及Web 应用程序布署环境目录遍历问题测试 2](#_Toc367267369)

[2 应用程序测试 3](#_Toc367267370)

[2.1 SQL 注入漏洞测试 3](#_Toc367267371)

[2.1.1 SQL注入漏洞攻击实现原理 3](#_Toc367267372)

[2.1.2 SQL注入漏洞防范措施 4](#_Toc367267373)

[2.1.3 SQL注入漏洞检测方法 5](#_Toc367267374)

[2.2 表单漏洞测试 6](#_Toc367267375)

[2.2.1 表单漏洞实现原理 6](#_Toc367267376)

[2.2.2 表单漏洞防范措施 6](#_Toc367267377)

[2.2.3 表单漏洞检测方法 6](#_Toc367267378)

[2.3 Cookie欺骗漏洞测试 8](#_Toc367267379)

[2.3.1 Cookie欺骗实现原理 8](#_Toc367267380)

[2.3.2 Cookie欺骗防范措施 8](#_Toc367267381)

[2.3.3 Cookie欺骗监测方法 9](#_Toc367267382)

[2.4 用户身份验证测试 9](#_Toc367267383)

[2.4.1 用户身份验证漏洞防范措施 9](#_Toc367267384)

[2.4.2 用户身份验证检测方法 9](#_Toc367267385)

[2.5 文件操作漏洞测试 9](#_Toc367267386)

[2.5.1 文件操作漏洞实现原理 9](#_Toc367267387)

[2.5.2 文件操作漏洞防范措施 10](#_Toc367267388)

[2.5.3 文件操作漏洞检测方法 10](#_Toc367267389)

[2.6 Session 测试 11](#_Toc367267390)

[2.6.1 客户端对服务器端的欺骗攻击 11](#_Toc367267391)

[2.6.2直接对服务器端的欺骗攻击 11](#_Toc367267392)

[2.6.3 Session漏洞检测方法 12](#_Toc367267393)

[2.7 跨网站脚本(XSS)漏洞测试 13](#_Toc367267394)

[2.7.1 跨网站脚本（XSS）漏洞实现原理 13](#_Toc367267395)

[2.7.2 跨网站脚本（XSS）漏洞防范措施 13](#_Toc367267396)

[2.7.3 跨网站脚本（XSS）漏洞测试方法 14](#_Toc367267397)

[2.8 命令注射漏洞测试 14](#_Toc367267398)

[2.9 日志文件测试 14](#_Toc367267399)

[2.10 访问控制策略测试 14](#_Toc367267400)

[2.10.1 访问控制策略测试方法 14](#_Toc367267401)

[3 数据库问题测试 15](#_Toc367267402)

[3.1 数据库名称和存放位置安全检测 15](#_Toc367267403)

[3.2 数据库本身的安全检测 15](#_Toc367267404)

[3.3 数据使用时的一致性和完整性测试 15](#_Toc367267405)

[4 容错测试 15](#_Toc367267406)

[4.1 容错方案及方案一致性测试 16](#_Toc367267407)

[4.2 接口容错测试 16](#_Toc367267408)

[4.3 压力测试 16](#_Toc367267409)

# 1 Web 应用程序布署环境测试

用来架构Web 网站的UNIX、LINUX、WINDOWS 等服务器端操作系统和服务器软件都可能存在漏洞（如前不久被发现的LINUX 系统内核漏洞），这些漏洞都会对Web 应用程序造成安全威胁。因此，在布署Web 应用程序前，应对Web 应用程序的布署环境进行严格的测试，检查一切已知的漏洞，发现新的漏洞，将应用程序环境带来的安全威胁降底到最低程度。

## 1.1HTTP 请求引发漏洞的测试

超长URL 的HTTP 请求，特殊格式字符的HTTP 请求，某些不存在文件的HTTP 请求，COM Internet Services (CIS)– RPC over HTTP 漏洞，从而引发拒绝服务，源代码显示，站点物理路径泄露，执行任意命令及命令注入等安全问题。因此，对非常规URL 的HTTP 请求要做全面的测试，以发现这方面的漏洞。测试工作以人工方式为主，并配以Tripwire和AIDE 的完整性检查工具检查系统文件，对于发现的漏洞，可采取关闭所有不必要的服务和安装系统补丁加固系统。另外要保持对最新补丁和安全公告的追踪，在实验环境进行测试后正式安装在布署Web 应用程序的主机上。

## 1.2 操作系统目录安全性及Web 应用程序布署环境目录遍历问题测试

目录权限和目录安全性直接影响着Web 的安全性。测试中要检查Web 应用程序布署环境的目录权限和安全性，不给恶意用户任何可用的权限。目录遍历可能导致用户从客户端

看到或下载、删除Web 服务器文件。因此，要测试Web 应用程序及布署环境是否存在目录遍历问题；若存在该漏洞，可通过在各级目录中存放默认文档或及时升级系统来避免。

**1.3 系统中危险组件的测试**

系统中危险组件的存在，会给恶意用户留下非常危险的“ 后门”。如恶意用户可利用Windows 系统中存在的FileSystemObject 组件篡改、下载或删除服务器中的任何文件。因此，若系统中需要使用这些组件，可将这些组件更名；否则将其删除。

**1.4 TCP 端口测试**

开放非必要的端口，会给Web 应用程序带来安全威胁。因此，在布署Web 应用程序前，要用端口扫描软件对布署环境进行TCP 端口测试，禁止UDP，只开启必要的TCP 端口。另

外，在系统运行过程中要不断测试，在服务器端使用lsof 工具(For Unix)或者Inzider 工具(For windows)扫描端口使用情况，必要时从远程使用Nmap 工具进行异常端口占用检测。如果发现有未知的进程占用端口，要关闭端口或杀掉进程。

# 2 应用程序测试

应用程序中存在的漏洞是影响Web 安全的主要方面，程序员编写的软件都可能有漏洞，有些漏洞可能要经过许多年后才会被发现。特别是不断新加的功能，这些改动，都会带

来安全方面的问题。因此，应用程序测试要伴随着系统开发、布署和运行的全过程。

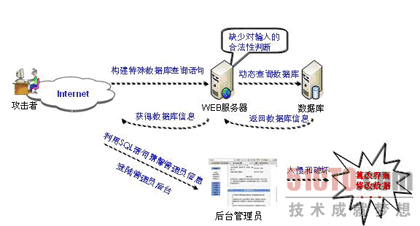
## 2.1 SQL 注入漏洞测试

### 2.1.1 SQL注入漏洞攻击实现原理

SQL（Structured Query Language）是一种用来和数据库交互的语言文本。SQL注入的攻击原理就是攻击者通过Web应用程序利用SQL语句或字符串将非法的数据插入到服务器端数据库中，获取数据库的管理用户权限，然后将数据库管理用户权限提升至操作系统管理用户权限，控制服务器操作系统，获取重要信息及机密文件。

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注入过程**



如上图所示，SQL注入攻击过程分为五个步骤：

**第一步：**判断Web环境是否可以SQL注入。如果URL仅是对网页的访问，不存在SQL注入问题，如：<http://news.xxx.com.cn/162414739931.shtml>就是普通的网页访问。只有对数据库进行动态查询的业务才可能存在SQL注入，如：<http://www.google.cn/webhp?id>＝39，其中?id＝39表示数据库查询变量，这种语句会在数据库中执行，因此可能会给数据库带来威胁。

**第二步：**寻找SQL注入点。完成上一步的片断后，就要寻找可利用的注入漏洞，通过输入一些特殊语句，可以根据浏览器返回信息，判断数据库类型，从而构建数据库查询语句找到注入点。

**第三步：**猜解用户名和密码。数据库中存放的表名、字段名都是有规律可言的。通过构建特殊数据库语句在数据库中依次查找表名、字段名、用户名和密码的长度，以及内容。这个猜测过程可以通过网上大量注入工具快速实现，并借助破解网站轻易破译用户密码。

**第四步：**寻找WEB管理后台入口。通常WEB后台管理的界面不面向普通用户

开放，要寻找到后台的登陆路径，可以利用扫描工具快速搜索到可能的登陆地址，依次进行尝试，就可以试出管理台的入口地址。

**第五步：**入侵和破坏。成功登陆后台管理后，接下来就可以任意进行破坏行为，如篡改网页、上传木马、修改、泄漏用户信息等，并进一步入侵数据库服务器。

### 2.1.2 SQL注入漏洞防范措施

SQL注入漏洞攻击的防范方法有很多种，现阶段总结起来有以下方法：  
**（1）数据有效性校验**

如果一个输入框只可能包括数字，那么要通过校验确保用户输入的都是数字。如果可以接受字母，那就要检查是不是存在不可接受的字符，最好的方法是增加字符复杂度自动验证功能。确保应用程序要检查以下字符：分号、等号、破折号、括号以及SQL关键字。另外限制表单数据输入和查询字符串输入的长度也是一个好方法。如果用户的登录名最多只有10个字符，那么不要认可表单中输入10个以上的字符，这将大大增加攻击者在SQL命令中插入有害代码的难度。  
**（2）封装数据信息**

对客户端提交的数据进行封装，不要将数据直接存入cookie中，方法就是在编程的代码中，插入session、if、try、else，这样可以有效地防止攻击者获取cookie中的重要信息。   
**（3）去除代码中的敏感信息**

将在代码中存在的用户名、口令信息等敏感字段删除，替换成输入框。  
如：SQL=" select from users where username = ’admin’and password= ’1234567’ "这样显然会暴露管理员的用户名、口令信息。可以将其修改成SQL= " select \* from users where username='" +Txtuser.Text + "' and userpwd='" + Textpwd.Text + "'"，这样就安全了很多，入侵者也是不会轻易的就获取到用户名、口令信息。  
**（4）替换或删除单引号**

使用双引号替换掉所有用户输入的单引号，这个简单的预防措施将在很大程度上预防SQL注入漏洞攻击，单引号时常会无法约束插入数据的Value，可能给予输入者不必要的权限。用双引号替换掉单引号可以使大部分SQL注入漏洞攻击失败。  
如：“select\* from users where username='" + admin + "' and userpwd='" + 1234567+ "'”显然会得到与“select \* from users where username='admin' and password= '1234567'”相同的结果。  
**（5）指定错误返回页面**

攻击者有时从客户端尝试提交有害代码和攻击字符串，根据Web Service给出的错误提示信息来收集程序及服务器的信息，从而获取想得到的资料。应在Web Service中指定一个不包含任何信息的错误提示页面。  
**（6）限制SQL字符串连接的配置文件**

使用SQL变量，因为变量不是可以执行的脚本，即在Web页面中将连接数据库的SQL字符串替换成指定的Value，然后将Web.config文件进行加密，拒绝访问。   
**（7）设置Web目录的访问权限**

将虚拟站点的文件目录禁止游客用户（如：Guest用户等）访问，将User用户权限修改成只读权限，切勿将管理权限的用户添加到访问列表。  
**（8）最小服务原则**

Web服务器应以最小权限进行配置，只提供Web服务，这样可以有效地阻止系统的危险命令，如ftp、cmd、vbscript等。   
**（9）鉴别信息加密存储**

将保存在数据库users表中的用户名、口令信息以密文形式保存，也可以对users表进行加密处理，这样可以大大增加对鉴别信息访问的安全级别。  
**（10）用户权限分离**

应尽可能的禁止或删除数据库中sa权限用户的访问，对不同的数据库划分不同的用户权限，这样不同的用户只能对授权给自己的数据库执行查询、插入、更新、删除操作，就可以防止不同用户对非授权的数据库进行访问。

### 2.1.3 SQL注入漏洞检测方法

SQL注入漏洞攻击检测分为入侵前的检测和入侵后的检测。入侵前的检测，可以通过手工方式，也可以使用SQL注入漏洞扫描工具软件。检测的目的是为预防SQL注入漏洞攻击，而对于SQL注入漏洞攻击后的检测，主要是针对审计日志的查看，SQL注入漏洞攻击成功后，会在Web Service和数据库的审计日志中留下“痕迹”。检测方法如下：

**（1）动态SQL检查**

动态的SQL语句是一个进行数据库查询的强大的工具，但把它和用户输入混合在一起就使SQL注入成为了可能。将动态的SQL语句替换成预编译的SQL或者存储过程对大多数应用程序是可行的。预编译的SQL或者存储过程可以将用户的输入作为参数而不是命令来执行，这样就限制了入侵者的行动。当然，它不适用于存储过程中利用用户输入来生成SQL命令的情况。在这种情况下，用户输入的SQL命令仍可能得到执行，数据库仍然存在SQL注入漏洞攻击的危险。  
**（2）有效性校验**

如果一个输入框只可能包括数字，那么要通过验证确保用户输入的都是数字。如果可以接受字母，检查是不是存在不可接受的字符，那就需要设置字符串检查功能。确保应用程序要检查以下字符：分号、等号、破折号、括号以及SQL关键字。  
**（3）数据表检查** 使用SQL注入漏洞攻击工具软件进行SQL注入漏洞攻击后，都会在数据库中生成一些临时表。通过查看数据库中最近新建的表的结构和内容，可以判断是否曾经发生过SQL注入漏洞攻击。  
**（4）审计日志检查** 在Web服务器中如果启用了审计日志功能，则Web Service审计日志会记录访问者的IP地址、访问时间、访问文件等信息，SQL注入漏洞攻击往往会大量访问某一个页面文件（存在SQL注入点的动态网页），审计日志文件会急剧增加，通过查看审计日志文件的大小以及审计日志文件中的内容，可以判断是否发生过SQL注入漏洞攻击事件；另外还可以通过查看数据库审计日志，查询某个时间段是否有非法的插入、修改、删除操作。  
**（5）其他**  
 SQL注入漏洞攻击成功后，入侵者往往会添加特权用户（如：administrator、root、sa等）、开放非法的远程服务以及安装木马后门程序等，可以通过查看用户帐户列表、远程服务开启情况、系统最近日期产生的一些文件等信息来判断是否发生过入侵。

SQL 注入攻击源于英文“SQL Injection Attack”。微软技术中心从两个方面对SQL 注入攻击进行了描述：一是脚本注入式的攻击；二是恶意用户输入用来影响被执行的SQL脚本。Stephen Kost 对这种攻击形式的描述是“从一个数据库获得未经授权的访问和直接检索”。

SQL 注入就其本质而言，是利用SQL 语法，对应用程序中的漏洞的攻击。当攻击者能够操纵数据，在应用程序中插入一些SQL 语句时，SQL 注入攻击就发生了。理论上，这种攻击对于所有基于SQL 语言标准的数据库软件都是有效的，包括MS SQL Server，Oracle，DB2，Sybase，MySQL等。特别是现在一些AQL 注入攻击工具的出现，使得Web应用更易遭到SQL 注入攻击。原始的手工测试不适用于大型Web 应用程序，可使用N-Stealth、WebInspect、Wikto WebScarab、Nikto 等工具进行扫描，测试系统是否存在SQL 注入的安全漏洞。为防止SQL 注入，程序员编写代码时，要对客户端和服务端进行两级检查。检查数据类型、数据长度和敏感字符的合法性。客户端检查可减少网络流量，降低服务器负荷，将一般误操作、低等级攻击与高等级攻击行为区分开来。对于绕开客户端检查的攻击，提交的数据被直接发往服务端，服务端检查到的提交异常基本可以认定为恶意攻击行为所致，就应中止提交信息的处理，进行攻击备案，并对客户端给出出错或警告提示。另外，在构造查询时，应根据用户输入的内容设置参数值来创建参数化查询，从而避免SQL 注入及由此带来的安全问题。

## 2.2 表单漏洞测试

### 2.2.1 表单漏洞实现原理

表单提交是当前Web应用中的重要内容，用户可以通过这种方式与服务器进行数据传递。在通常情况下，会在提交表单之前在服务器上进行表单数据的验证，这样可以节省服务器资源，但同时也为服务器带来了安全漏洞。

表单提交的数据的验证和服务端数据接收的方法直接影响到Web 的安全。随着大量的支持参数的“模糊化”（“fuzzing”）、腐朽（corruption）、以及野蛮强制增长工具的出现，使用非校验输入进行攻击造成的安全问题越来越多。因此，表单漏洞测试是Web 安全所必需的。

### 2.2.2 表单漏洞防范措施

为防止表单漏洞的攻击，编程时应有一个中心化的、强大的验证机制来对所有HTTP 请求的输入进行验证，过滤可能危及后台数据库的特殊字符、脚本语言和命令。

为防止攻击者绕过客户端的安全机制，对这些字符的检测应在Web 服务端实现，采用清除或者强制替换的方法避免服务器端的安全，并且使用MD5 哈希(hash)函数或者时间戳数字签名技术对客户端敏感数据必须进行完整性保护。

解决这种漏洞的方法为在提交表单页面进行校验的同时，在接收表单的处理页面也进行校验，这样即使用户使用非法方式提交的非法数据通过了页面验证也无法通过服务器上的验证。

### 2.2.3 表单漏洞检测方法

**⑴ 表单数据提交测试。**

* 对表单数据提交的测试，主要检查程序中是否对表单所提交数据的完整性、正确性进行了验证（如果在页面部分进行验证的话），如：查询条件输入一些特殊字符，比如“--”，“‘，，’”，““”等会使查询的SQL语句出错
* 检查程序中是否屏蔽了表单提交的html 语句、VBScript 和Jscript 等客户端脚本语句
* 检查是否会出现“脚本利用”问题
* 检查程序是否对表单域长度进行了真正的限制
* 检查是否存在重复提交数据的问题
* 检查这些验证是否在服务器端进行

对表单提交数据的测试，可以采用手工和编写可重复使用的脚本代码相结合的方法，进行边界值测试、等价类测试，以及异常类测试。编写的脚本代码可以在测试、回归测试时运行。

若在测试中发现数据完整性、正确性验证只是在客户端进行，应在服务器端增加对表单提交数据的验证，防止出现本地提交表单的漏洞。

**⑵ 本地提交表单的漏洞测试。**

本地提交表单的漏洞容易受到参数篡改的攻击。这类测试可用手工的方式进行。

对于如下用户注册页面：

1. <script>
2. **function** checkUser(){
3. **if**(document.getElementById("userName").value==""){
4. document.getElementById("button").disabled=true;
5. alert("用户名不可以为空");
6. **return** false;
7. }**else**{
8. document.getElementById("button").disabled=false;
9. }
10. }
11. **function** checkPsw(){
12. **if**(document.getElementById("password").value==""){
13. document.getElementById("button").disabled=true;
14. alert("密码不可以为空");
15. **return** false;
16. }**else**{
17. document.getElementById("button").disabled=false;
18. }
19. }
20. </script>
21. <form action="regist.php" method="POST">
22. <input type="text" name="userName" onblur="checkUser()"/>
23. <input type="password" name="password" onblur="checkPsw()"/>
24. <input type="submit" name="button" value="提交"/>
25. </form>

此页面中的JavaScript对表单中用户名和密码是否为空进行了判断，如果用户名或密码有一者为空时就会将"提交"按钮设置为不可用，这样可以阻止用户的提交。

但是这个页面的内容可以完全通过查看页面源代码的方式看到，用户可以通过查看源代码的方式将其中的JavaScript部分去掉，同时将表单action请求指向此页面原来的地址，然后将修改后的页面保存为一个静态HTML页面，这样就可以完成一个非法的数据提交。修改之后的页面代码如下：

1. <form action="http://www.abcdefg.com/testbug/regist.php" method="POST">
2. <input type="text" name="userName" onblur="checkUser()"/>
3. <input type="password" name="password" onblur="checkPsw()"/>
4. <input type="submit" name="button" value="提交"/>
5. </form>

如此一个页面，完全允许用户提交任何一种数据，包括空的用户名和密码。

## 2.3 Cookie欺骗漏洞测试

### 2.3.1 Cookie欺骗实现原理

Cookie最先是由Netscape（网景）公司提出的，Netscape官方文档中对Cookie的定义是这样的：Cookie是在HTTP协议下，服务器或脚本可以维护客户工作站上信息的一种方式。

Cookie的用途非常广泛，在网络中经常可以见到Cookie的身影。它通常被用来辨别用户身份、进行session跟踪，最典型的应用就是保存用户的账号和密码用来自动登录网站和电子商务网站中的“购物车”。

Cookie注入简单来说就是利用Cookie而发起的注入攻击。从本质上来讲，Cookie注入与传统的SQL注入并无不同，两者都是针对数据库的注入，只是表现形式上略有不同罢了。

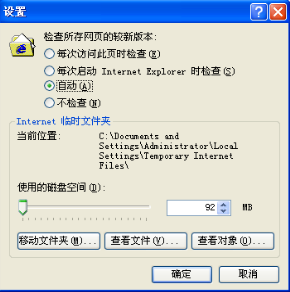
### 2.3.2 Cookie欺骗防范措施

**（1）删除Cookie记录**

在IE浏览器【工具】🡪【Internet选项】中删除Cookie，也可借助相应安全软件来实现。

**（2）更改Cookie文件的保存位置**

**在**【Internet选项】对话框中单击【设置】按钮，在设置页面单击【移动文件夹】出现如下图，在其中设置相应保存位置（如F:\），即可成功更改Cookie文件的保存位置。

****

**（3）添加防注入代码**

### 2.3.3 Cookie欺骗监测方法

如果系统使用了cookie，就要对cookie 的使用情况进行检测。检查Cookies 在生存期内能否正常工作而且对这些信息是否加密，是否按预定的时间进行保存，是否存在cookie 可被伪造提交的问题，刷新对Cookie 有什么影响及过期处理等。

## 2.4 用户身份验证测试

### 2.4.1 用户身份验证漏洞防范措施

* 限制用户名、密码最大字符数、字符类型
* 限制登录次数，超出允许次数后给出友好提示
* 限制用户权限

用户身份验证，客户端提交的密码需加密，服务端验证密码使用MD5，若在测试中发现问题，应及时修改代码，使用加密码算法对密码加密。

### 2.4.2 用户身份验证检测方法

* 测试有效和无效的用户名和密码，测试是否大小写敏感，是否有最大字符数的限制规则等。
* 测试重试次数的限制，如果登录失败的次数超过允许值，应用程序将会做出何种反应 （譬如拒绝此IP地址在短时间内的登录）。
* 测试是否可以利用历史登录信息或以前的URL来绕开登录程序。
* 测试执行添加、删除、修改等动作中是否需要登录操作，退出系统之后的操作是否仍可继续等。
* 测试用户密码是否符合指定要求（字符、长度），如果不符合，对有什么影响，新用户自己修改密码后，创建时分配的密码是否会失效。
* 测试用户账户过期后，是否完全、正确的删除其信息或使其失效。
* 是否存在不验证而直接进入Web 应用系统的问题，是否存在不登录就可查看非会员页面和权限问题。

用户身份验证测试一般使用手工和测试工具相结法的方法，若在测试中发现问题，应及时修改代码，使用加密码算法对密码加密，采用Session 对象进行登录验证。

## 2.5 文件操作漏洞测试

### 2.5.1 文件操作漏洞实现原理

上存漏洞常见有，[文件名检测漏洞](http://blog.chacuo.net/136.html)，还有就是[文件格式检查漏洞](http://blog.chacuo.net/141.html)。 另外还有一个，就是保存文件存在漏洞。这类漏洞，主要是可以读取用户传入路径名称，采用不正确的过滤方法，导致恶意用户，将文件上存到非预期的地方，带来安全隐患。

### 2.5.2 文件操作漏洞防范措施

抓住几个地方即可，先来分析下，既然用户要上存文件，而且文件将是多种多样格式；可能有的文件内容与用户传入格式不一致，有的文件内容还夹杂木马代码。 那么，我们让用户上存文件，跟站点文件做一个分别授权，做隔离。

* **让保存上存目录独立开来，目录权限只读不能执行**

这一步从系统设计加以授权，无论你上次什么文件，都不可能执行到。就算我不做任何检测，你的文件都上存到这里了，也不会对我系统构成安全。（如果有用户上存一些反动言语的图片，那另外需要处理的）

* **不直接使用服务器传入值，所有都要进行检测**

这类跟我们做一切输入都是有害原则一样，对于客户端传入的：type, name ，都要进行判断，不直接使用。对于要生成到某个目录，某个文件名。

文件名最好方法是：自己写死目录（不要读取传入目录），文件名，最好自己随机生成，不读取用户文件名。文件扩展名，可以取最右边”.”后面字符。

以上2个方法，刚好从2个方面对上存做了整体约束。

方法1：只要保证文件写对了位置，然后从配置上，对写入目录进行权限控制，这个是治本。可以做到，你无论上存什么文件，都让你没有权限跳出去可以运行。

方法2 ： 保存上存文件名，按照自己指定目录写入，并且文件名自己生成的。

以上2个方法，一起使用，可以保证文件正确存到地方，然后，权限可以控制。 这里顺便说明下， 判断用户上存文件是否满足要求类型，就直接检查文件扩展名，只要满足扩展名就让上存。 反正，做了执行权限限制，你不按要求上存内容，也无妨。 反正，不能执行，也不会有多大危害性的。

正确步骤：

1.读取文件名，验证扩展名是不是在范围内

2.自己定义生成的文件名，目录，扩展名可以来自文件名扩展名。 其它值，都自己配置，不读取上存中内容

3.将文件 移到新目录(这个目录权限设置只读)

### 2.5.3 文件操作漏洞检测方法

* 测试系统是否允许上传脚本文件、可执行文件等有可能给系统带来危害的文件。
* 若有下载功能，可供下载的文件是否与系统的程序分别存放，是否存在数据库文件、包含文件和页面文件下载的可能。

文件操作漏洞测试一般使用手工测试的方法，若发现问题，应及时修改代码并将可供下载的文件重新布署。

## 2.6 Session 测试

### 2.6.1 客户端对服务器端的欺骗攻击

#### 2.6.1.1 攻击原理

在用户访问并登录了某个网站后，该网站的服务器就会给该用户机子上分配一个sessionid，此信息是通过客户机的cookies存放在某一文件夹里面的。session本身并不是长期有效，每一个session在客户端关闭浏览器后sessionid都会自动撤销，但服务端默认保存20分钟。正是有这20分钟的时间，给欺骗带来了可能。 服务器端对不同用户的识别，只能通过sessionid进行，也就是说网站服务器端是“只认id不认人”，只要id符合，就认为是合法用户，所以攻击者只要得到了被攻击对象的sessionid就可以以被攻击对象的身份合法登录，而20分钟的默认保留值，也使得攻击者即使在被攻击对象关闭浏览器后依然有一定的时间成功登录。

当前利用此原理进行攻击的常用手法是利用网站本身的xss漏洞或者是诱骗被攻击者点击相应链接，以使其隐蔽访问攻击者事先假设好的网站并执行恶意代码，获取被攻击者的cookies信息从而得到sessionid。最后用可以修改sessionid的浏览器或其它可以提交数据的工具伪装成被攻击者的id合法登录。

#### 2.6.1.2 防范措施

此类攻击过程较为隐蔽，但也有其局限性。因为session本身有时间限制，特别是用户在关闭浏览器后，留给攻击者的时间也只有20分钟。另外，在触发机制上，盗窃cookies代码的执行必须是用户自己触发，也就是说，用户什么时候触发代码，攻击者是不知道的，从用户触发恶意代码到session失效，攻击者只有很短的时间进行非法活动。

所以，要对此类攻击进行防范，客户端本身应对陌生人给出的超级链接保持警惕，特别是对比较长的超链接更要小心。每次登录网站后应该及时利用网站的退出功能退出和清除本机的cookies。另外登录密码的设置不要过于简单，尽量使用字母与数字组合，密码长度应该在8位以上。网站管理员在开发网站时要注意检查网站的xss漏洞，要注意session有效期的设置，一般不要把有效期设置太长，这样即使真的被攻击也能让其非法活动时间大大缩短。

### 2.6.2直接对服务器端的欺骗攻击

#### 2.6.2.1 攻击原理

与客户端欺骗不同，此类攻击是对服务器端的直接欺骗。网站开发者在管理员管理页面通常都会有session验证，目的是为了验证当前登录者是否为合法用户。

但如果攻击者能够在登录管理页面前，使用某种手段使得session（”admin”）被赋值（不一定是网站开发者所赋予的值）则验证代码则无法拦截此类非法登录。从而达到了直接欺骗服务器而以管理员身份直接登录的目的。

当前利用此原理进行攻击的常用手法是都是在取得了某网站域名下的某个webshell进行的。如许多免费空间网站都会在主域名下允许用户有自己的二级域名，而此域名的目录又和网站主目录在同一站点下，这样就为欺骗攻击提供了条件。而其它一些网站由于自身的xss等漏洞，使得用户拿下某个webshell，从而也使欺骗攻击成为了可能。在具备了欺骗必备条件后，攻击者还必须知道session验证的源码，从而才能编写恶意代码绕过系统原有的验证。当然，如果验证源码是上文所列的情况，则攻击者只需知道session所用变量即可，因为其并没有给出具体的赋值，只是简单的验证是否为空。攻击者只要赋任意值即可通过此验证。

#### 2.6.2.2 防范措施

此类攻击手法也相当隐蔽，危害极大。因为攻击一旦成功，则攻击者即可以管理员身份非法登录。从此类欺骗攻击的原理我们知道，此攻击要成功必须具备多个先决条件：获得该域名下的一个webshell或者攻击网站与被欺骗网站在同一主站的同一目录下；被攻击主站采取了session验证管理页面；获得被攻击站点的session验证源码；知道被攻击主站的管理页面地址。

所以我们要防范此类攻击，就只能从几个限制条件考虑。网站开发者应该尽量避免各种漏洞，站点在使用前应该经过周密而详尽的测试，从而降低被发现漏洞的可能。对于网站源码不能轻易泄露，特别是在公开场合。如果是使用公开源码假设的网站，更要把源码关键部位更改。本攻击欺骗的关键源码部位即session验证源码。

### 2.6.3 Session漏洞检测方法

**（1）Session互窜**

  Session互窜即是用户A的操作被用户B执行了。

  验证Session互窜，其原理还是基于权限控制，如某笔订单只能是A进行操作，或者只能是A才能看到的页面，但是B的session窜进来却能够获得A的订单详情等。   Session互窜方法：   多TAB浏览器，在两个TAB页中都保留的是用户A的session记录，然后在其中一个TAB页执行退出操作，登陆用户B， 此时两个TAB页都是B的session，然后在另一个A的页面执行操作，查看是否能成功。 预期结果：有权限控制的操作，B不能执行A页面的操作，应该报错，没有权限控制的操作，B执行了A页面 操作后，数据记录是B的而不是A的。

**（2）Session超时**

  基于Session原理，需要验证系统session是否有超时机制，还需要验证session超时后功能是否还能继续走下去。   测试方法：

  1、打开一个页面，等着10分钟session超时时间到了，然后对页面进行操作，查看效果。

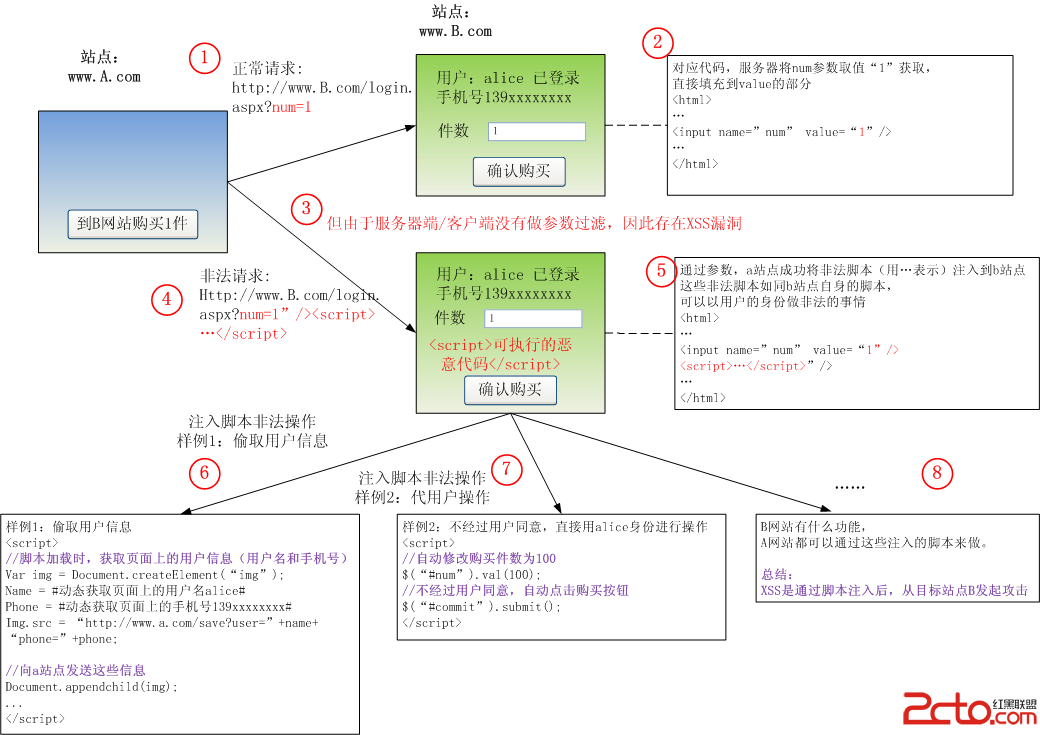
  2、多TAB浏览器，在两个TAB页中都保留的是用户A的session记录，然后在其中一个TAB页执行退出操作，马上在另外一个页面进行要验证的操作，查看是能继续到下一步还是到登录页面。

Session 测试主要检查Web 应用系统是否有超时的限制，也就是检查用户登录后在一定时间内没有点击任何页面，是否需要重新登录才能正常使用，检查超时后能否自动退出，退出之后，浏览器回退按钮是否可以回到登录页面。Session 测试一般使用手工测试和工具测试相结合的方法，若发现问题，应及时修改代码。

## 2.7 跨网站脚本(XSS)漏洞测试

### 2.7.1 跨网站脚本（XSS）漏洞实现原理

跨站脚本漏洞（Cross Site Scripting，常简写作XSS）是Web应用程序在将数据输出到网页的时候存在问题，导致攻击者可以将构造的恶意数据显示在页面的漏洞。因为跨站脚本攻击都是向网页内容中写入一段恶意的脚本或者HTML代码，故跨站脚本漏洞也被叫做HTML注入漏洞（HTML Injection）。与SQL注入攻击数据库服务器的方式不同，跨站脚本漏洞是在客户端发动造成攻击，也就是说，利用跨站脚本漏洞注入的恶意代码是在用户电脑上的浏览器中运行的。

****

### 2.7.2 跨网站脚本（XSS）漏洞防范措施

下列规则旨在防止所有发生在应用程序的XSS攻击，虽然这些规则不允许任意向HTML文档放入不可信数据，不过基本上也涵盖了绝大多数常见的情况。你不需要采用所有规则，很多企业可能会发现第一条和第二条就已经足以满足需求了。请根据自己的需求选择规则。

1. **不要在允许位置插入不可信数据**

第一条规则就是拒绝所有数据，不要将不可信数据放入HTML文档，除非是下列定义的插槽。这样做的理由是在理列有解码规则的HTML中有很多奇怪的context，让事情变得很复杂，因此没有理由将不可信数据放在这些context中。

1. **在向HTML元素内容插入不可信数据前对HTML解码**

这条规则适用于当你想把不可信数据直接插入HTML正文某处时，这包括内部正常标签(div、p、b、td等)。大多数网站框架都有HTML解码的方法且能够躲开下列字符。但是，这对于其他HTML context是远远不够的，你需要部署其他规则。

1. **在向HTML常见属性插入不可信数据前进行属性解码**

这条规则是将不可信数据转化为典型属性值(如宽度、名称、值等)，这不能用于复杂属性(如href、src、style或者其他事件处理程序)。这是及其重要的规则，事件处理器属性(为HTML JavaScript Data Values)必须遵守该规则。

### 2.7.3 跨网站脚本（XSS）漏洞测试方法

对于呈增长趋势的跨站脚本（XSS）攻击，可使用内嵌检测的方式进行处理。使用WebInspect 工具识别所有的假造参数，使用DevInspect 工具通过特定代码关联在页面上发现安全缺陷，对于显示代码，采用CSE HTML Validator工具进行测试。若在检测中发现系统存在跨站脚本(X SS)漏洞，使用输出数据编码也就是将任何数据返回给用户前均采用HTML 编码，可以有效防止跨站点脚本攻击。因为通过HTML 编码，可将大多数脚本命令自动转换为无害文本。

## 2.8 命令注射漏洞测试

命令注射漏洞测试主要检查所有调用外部资源(例如system、exec、fork，或者所有的发出请求的语法的源代码，查找那些来自于HTTP 请求的输入可能发起调用的所有地方。使用相同功能的专门的库函数来代替shell 命令和一些系统调用，可以抵御命令注射的攻击，另外一种避免命令注射的保护措施就是确保Web 应用程序只是根据它所要执行某个功能时所需的最小权限来实现这个功能。如果必须使用外部命令的话，任何被插入命令的用户信息必须仔细地审查，设立一定的机制来处理可能出现的错误、超时、或者在调用过程中出现的阻塞。

## 2.9 日志文件测试

日志文件测试主要检查Web 运行的相关信息是否写进了日志文件、是否可追踪，是否记录了系统运行中发生的所有错误，是否记录了用户的详细信息，包括用户的浏览器、用户停留的时间、用户IP 等。记录了用户的IP，就能通过追捕查出用户的具体地点。错误作为日志保留下来，可供技术人员分析错误是由系统实现漏洞引起的还是由于黑客攻击引起的。

## 2.10 访问控制策略测试

访问控制策略是网络安全防范和保护的主要策略，其任务是保证网络资源不被非法使用和非法访问。各种网络安全策略必须相互配合才能真正起到保护作用，而访问控制是保证网络安全最重要的核心策略之一。访问控制策略包括入网访问控制策略、操作权限控制策略、目录安全控制策略、属性安全控制策略、网络服务器安全控制策略、网络监测、锁定控制策略和防火墙控制策略等7个方面的内容。

### 2.10.1 访问控制策略测试方法

* 主要检查管理接口是否只有授权的管理员才允许进行访问（对于支持多种管理角色的网站接口往往是内部或者外部攻击者的攻击目标）
* 是否有完善的访问控制策略文档（如果该文档不存在的话，这个网站很可能存在着漏洞），文档中是否精确定义了每类用户可以访问的功能和内容
* 检查是否只有授权的数据可被访问（如果存在着不同类型或不同组别的数据可以通过接口被访问到）。

主要用于需要验证用户身份以及权限的页面，复制该页面的url地址，关闭该页面以后，查看是否可以直接进入该复制好的地址；

例：从一个页面链到另一个页面的间隙可以看到URL地址，直接输入该地址，可以看到自己没有权限的页面信息；

# 3 数据库问题测试

在Web 应用中，数据库起着重要的作用，数据库为Web应用系统的管理、运行、查询和实现用户对数据存储的请求等提供空间。因此，Web 应用系统中数据库安全测试是一个重要的方面。与数据库相关的应用级漏洞如SQL 注入和跨站点脚本攻击等问题前面已提及，在此只讨论数据库本身及数据库使用方面的漏洞。

## 3.1 数据库名称和存放位置安全检测

一个常规的数据库名称，并且存放在与Web 应用程序文件相同或相关的位置，很容易被下载。若在程序代码中包含有数据库名称和数据库文件绝对位置，一旦代码丢失，同样存在暴库的危险。因此，在布署数据库和编写相关的代码时，要避免问题的发生。

## 3.2 数据库本身的安全检测

对数据库本身的安全检测主要检查数据库是否配置了不同的存取权限，所有操作是否都可以审计追踪，敏感数据是否加密等。为了保证数据库的安全，不同权限的用户定义不同的视图，以限制用户的访问范围；不同的敏感数据采取不同的加密算法，重要的数据分开存储。

## 3.3 数据使用时的一致性和完整性测试

Web 应用系统中，使用数据库时，可能发生数据的一致性和完整性错误，因此，要检测系统中是否有事务管理和故障恢复功能，确认事务数据是否正确保存，检测系统是否有定期数据备份功能。

# 4 容错测试

正常操作时，Web 应用程序也总会有错误出现，如内存溢出、空指针异常、系统调用失败、数据库不可用、网络超时等。不当的出错处理可能给网站带来各种各样的安全问题，因此，要对Web 应用程序的错误处理进行测试，以保证为用户提供一份有意义的出错信息，为网站维护人员提供诊断信息，而不是为攻击者提供有用的信息。

## 4.1 容错方案及方案一致性测试

出错处理应该在整个网站中保持一致性，并且每一个出32错处理片断都应该是一个整体设计方案中的一部分。通过代码检查，测试系统差错处理方案是否合理，方案是否可以处理所有可能发生的错误，方案中是否存在泄漏设计细节的问题，是否存在不同的差错处理方案。

## 4.2 接口容错测试

检测浏览器与服务器的接口是否正确，中断用户到服务器的网络连接时，系统是否能够正确处理数据；对于有外部接口的Web 系统，如网上商店可能要实时验证信用卡数据以减少欺诈行为的发生，中断Web 服务器到信用卡验证服务器的连接，检测系统是否能够正确处理这些错误，是否对信用卡进行收费。另外，还要测试系统是否能够处理外部服务器返回的所有可能的消息。

## 4.3 压力测试

压力测试是实际破坏一个Web 应用系统，也就是测试Web应用系统会不会崩溃，在什么情况下会崩溃。黑客常常提供错误的数据负载，直到Web 应用系统崩溃，接着当系统重新

启动时获得存取权。压力测试的区域包括表单、登录和其他信息传输页面，可以采用ab（Apache 的测试工具）和OpenSTA（开发系统测试架构）等相应的工具进行自动化测试。