

安全分享之

移动互联网开发安全

Email: phonpeng@vip.qq.com

2016年07月18日



● 目录

- ✓ 应用安全概况
- ✓ 常见Web安全漏洞
- ✓ 常见Android安全漏洞
- ✓ 安全开发总结

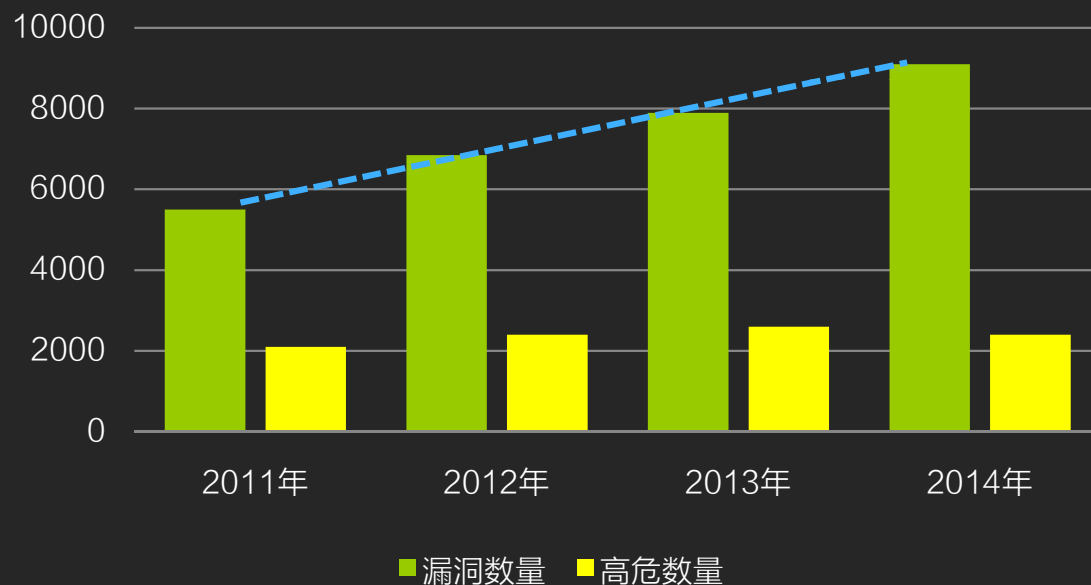


一、应用安全概况

● 中国互联网安全漏洞数逐年增加

- 2014 年，CNVD 收录并发布各类安全漏洞 9163 个，较 2013 年增长 16.7%，其中高危漏洞 2394 个，占 26.1%。

近年CNVD收录的漏洞情况



*数据来源: [CNCERT](#)

● 安全漏洞可产生诸多危害

服务器入侵



数据泄漏



木马攻击



蠕虫攻击



● 安全漏洞让企业损失惨重

- 携程爆发安全漏洞，导致股价盘前大跌11%

安全出漏洞携程遭部分用户停用 股价一度暴跌

2014年03月25日 08:30:27 来源: 中证网 [分享到:](#) [T](#) [+](#) [+](#) [+](#) [+](#) [+](#) 0 [A-](#) [A+](#)

3月22日晚间，乌云漏洞平台发布报告称，[携程](#)系统存技术漏洞，黑客可从中获取用户个人信息、银行卡号等信息。随后，携程承认了漏洞的存在。

受漏洞门事件影响，携程股价昨日盘前一度跌近10%。

- 索尼影业遭受黑客攻击，预计损失高达1亿美元

網易科技 [网易首页](#) > [网易科技](#) > [IT业界](#) > 正文

索尼影业被黑 损失可能高达1亿美元

2014-12-10 11:58:07 来源: 网易科技报道

● 中国电信曾发生过的安全事件

中国电信被曝重大漏洞 可查上亿用户信息

正文

我来说两句 (22人参与)

扫描到手机

2015-10-30 09:47:03 来源: 界面

手机看新闻 | 保存到博客 | 分享 | 打印

近日，补天漏洞响应平台再次爆出中国电信某系统的重大漏洞。通过该漏洞可以查询上亿用户信息，涉及姓名、证件号、余额，并可以进行任意金额充值、销户、换卡等操作。10月29日上午10点，该漏洞已得到中国电信厂商确认。

公开时间：2013-03-08 08:42

漏洞类型：设计缺陷/逻辑错误
危害等级：高危
漏洞来源：http://www.wooyun.org
漏洞状态：已交由第三方合作机构(cncert国家互联网应急中心)处理
漏洞介绍：林伟介绍，一般厂商会在确认系统漏洞当天完成修补并进行反馈。据了解，中国电信现已关停相关服务器。

漏洞来源：http://www.wooyun.org

漏洞状态：已交由第三方合作机构(cncert国家互联网应急中心)处理
漏洞介绍：林伟介绍，一般厂商会在确认系统漏洞当天完成修补并进行反馈。据了解，中国电信现已关停相关服务器。

● 一般黑客攻击动机

- 获取经济利益
- 打压竞争对手
- 宣泄个人不满
- 菜鸟实验行为
-

● 黑客攻击动机——获取经济利益

- 利用漏洞推广网站、软件；
- 利用漏洞恶意吸粉；
- 利用漏洞卖取特权；
- 做黑产任务获利；
- 直接漏洞交易；
- 其它漏洞变现的渠道...



● 黑客攻击动机——打压竞争对手

- 同行之间互相竞争，雇佣黑客，攻击对方；

雇用“黑客”攻击竞争对手网站

南山网警跨省破案，河南“黑客”及其北京雇主相继落网

和讯网 > 科技 > 科技要闻 > 正文

  扫一扫下载手机和讯网

网络攻击竞争对手三被告获刑

字号   

2014-11-21 15:05:33 来源：新民晚报 作者：袁玮

[评论](#) [邮件](#) [纠错](#)

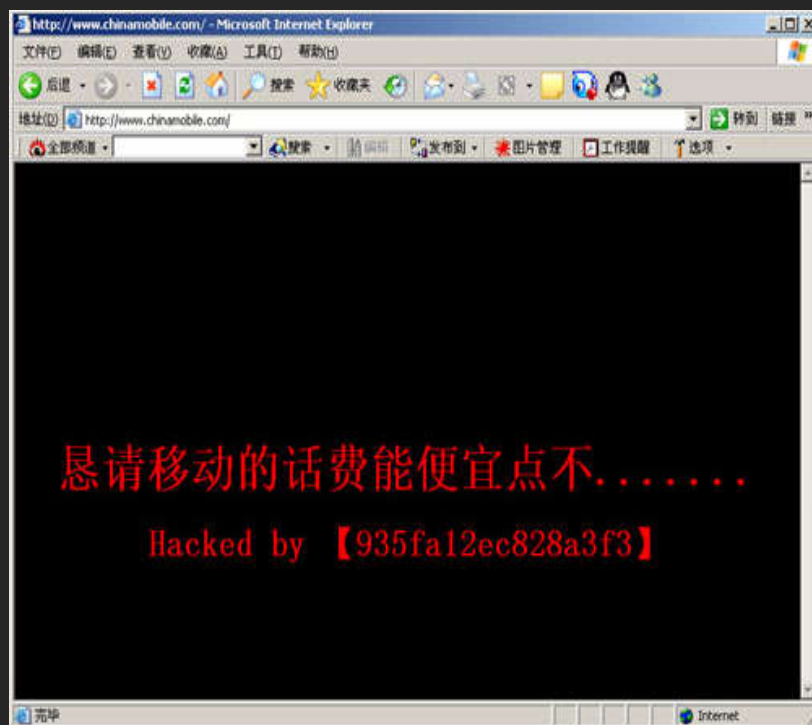
● 黑客攻击动机——宣泄个人不满

- 为了表达个人的不满情绪，攻击目标网站泄愤：



● 黑客攻击动机——菜鸟实验行为

- 菜鸟黑客在初期阶段行为缺少目的性，往往是随便拿个网站开刀；
- 自认为是高手，喜欢炫耀，通过攻击一个网站来证明自己的实力。



23岁温州黑客攻击服务器纯为炫耀 入侵CSDN被捕

2012-03-22 15:56 来源：浙江在线 我要评论 (0)

硕士炫耀“黑客”技术 非法入侵公司网站获刑

晶报讯（记者 吴欣 通讯员 周丽华）硕士研究生刘某将攻击网站作为“业余技术爱好”，还为向同学炫耀其“黑客”技术，结果因非法入侵公司网站被判刑。经南山区检察院提起公诉，法院近日作出一审判决：以非法获取计算机信息系统数据罪判处刘某有期徒刑6个月，缓刑1年，并处罚金1万元。



二、常见Web漏洞

SQL注入

宝刀未老，危害巨大

● SQL注入漏洞是什么

SQL注入攻击（SQL injection），简称注入攻击

是发生于应用程序数据库层的安全漏洞，简而言之，是在输入的字符串之中注入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，因此遭到破坏。

——From 维基百科

● SQL注入漏洞原理

原始SQL:

```
SELECT * FROM users WHERE user='user' AND password='password';
```

正常情况(user=aaa, password=123456):

```
SELECT * FROM users WHERE username='aaa' AND password='123456';
```

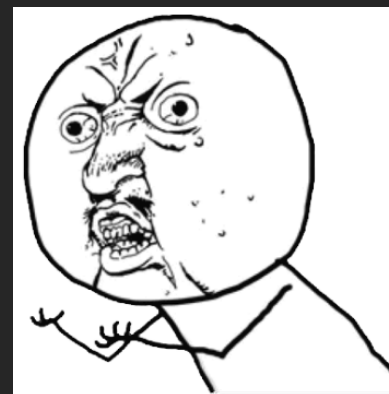
如果username=admin, password值为' OR '1'='1', 那么结果将会是:

```
SELECT * FROM users WHERE user='admin'
```

```
AND password=' ' OR '1'='1';
```

不用密码就以管理员身份登录!

案例: <http://wooyun.org/bugs/wooyun-2014-065786>



● 如何注入？

猜测后台所使用的SQL:

```
SELECT * FROM XXX WHERE id= request.getParameter("id")
```

获取查询的字段数:

```
SELECT * FROM XXX WHERE id=1 ORDER BY x
```

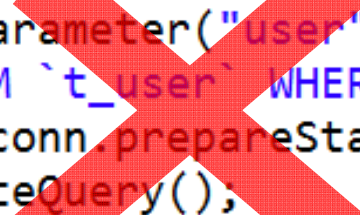
使用UNION查询获取需要的数据:

```
SELECT * FROM XXX WHERE id= xxx UNION ALL SELECT xx,yy,...
```

因为查询一次只返回一条数据，所以要让UNION的前半部分查询为空:

```
SELECT * FROM XXX WHERE id= 0 UNION ALL SELECT xx,yy,...
```

● SQL注入漏洞防衛



```
String user = request.getParameter("user");
String sql = "SELECT * FROM `t_user` WHERE username='" + user + "'";
PreparedStatement pstmt = conn.prepareStatement(sql);
ResultSet rs = pstmt.executeQuery();
```



```
String user = request.getParameter("user");
String sql = "SELECT * FROM `t_user` WHERE username=?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, user);
ResultSet rs = pstmt.executeQuery();
```

XSS漏洞

历史悠久，普遍存在

● XSS漏洞是什么

跨站脚本（ Cross-site scripting，通常简称为XSS或跨站脚本或跨站脚本攻击）

是一种站点应用程序的安全漏洞攻击，是代码注入的一种。它允许恶意用户将代码注入到网页上，其他用户在观看网页时就会受到影响。这类攻击通常包含了HTML以及用户端脚本语言。

——From 维基百科

● 同源策略

- 由Netscape提出的一个著名安全策略
- 是浏览器最核心、最基本的安全功能
- 同源是指：协议、域名、端口相同

`http://www.aa.com/` & `https://www.aa.com/` ❌

`http://www.aa.com/` & `http://www.bb.com/` ❌

`http://www.aa.com:80/` & `http://www.aa.com:8080/` ❌

`http://www.aa.com/` & `http://www.aa.com/a/` ✅

● XSS的历史

- 萌芽于上个世纪九十年代中期
- 正名于1999年
- 流行于2005年之后

历史

欲知大道，必先为史

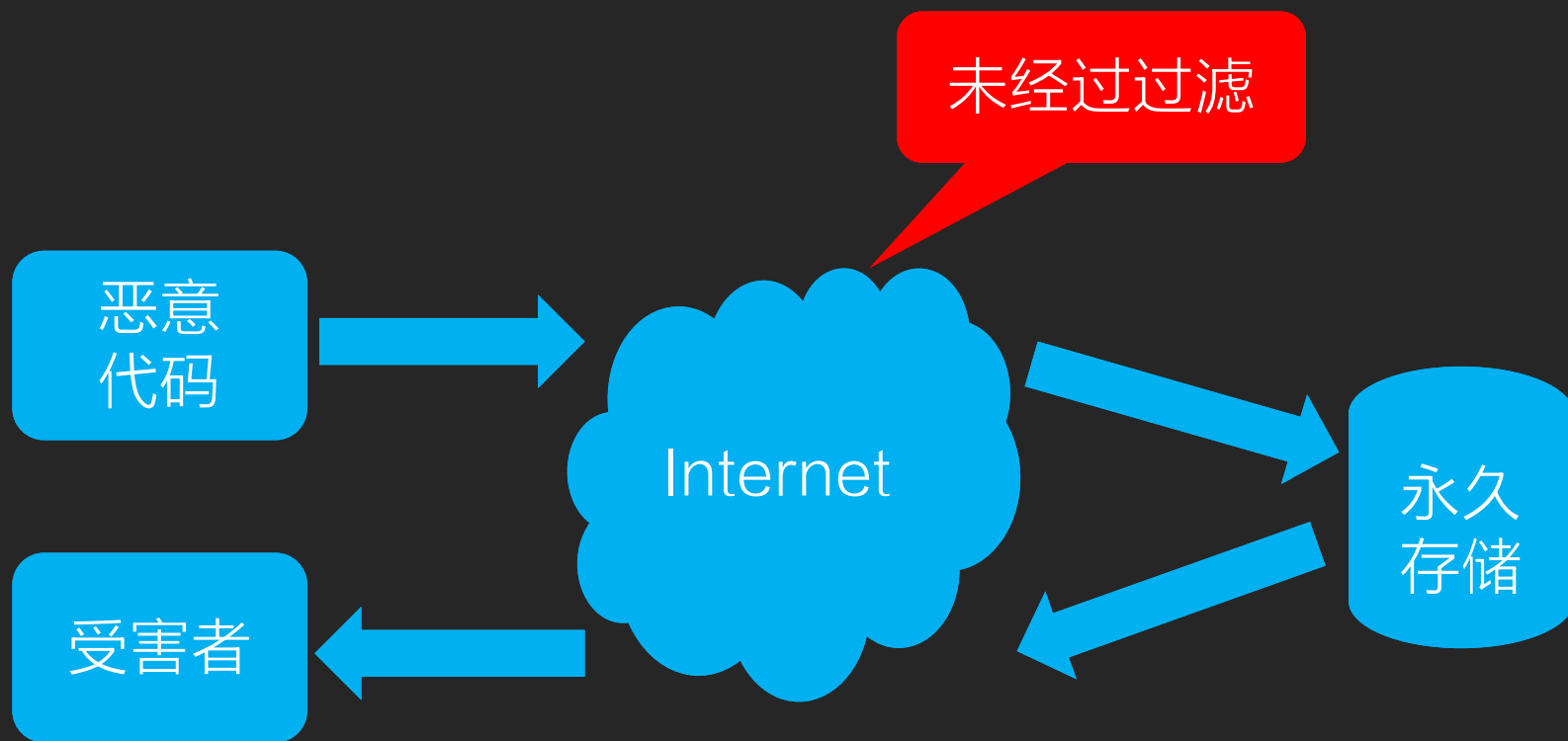
● XSS的分类

- Persistent型（存储型）
- Non-persistent型（反射型）
- DOM-based型

分类

业界对跨站漏洞的常见分类

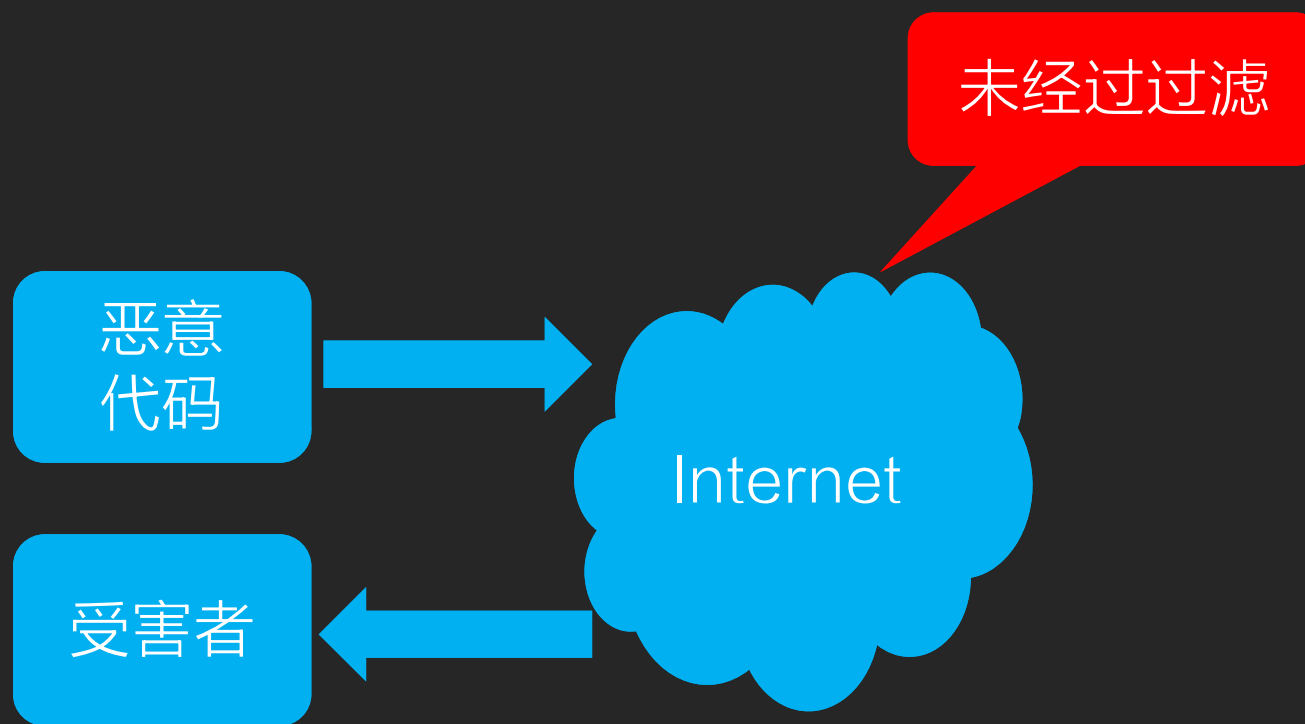
● 存储型XSS



特点

攻击代码会被永久存储

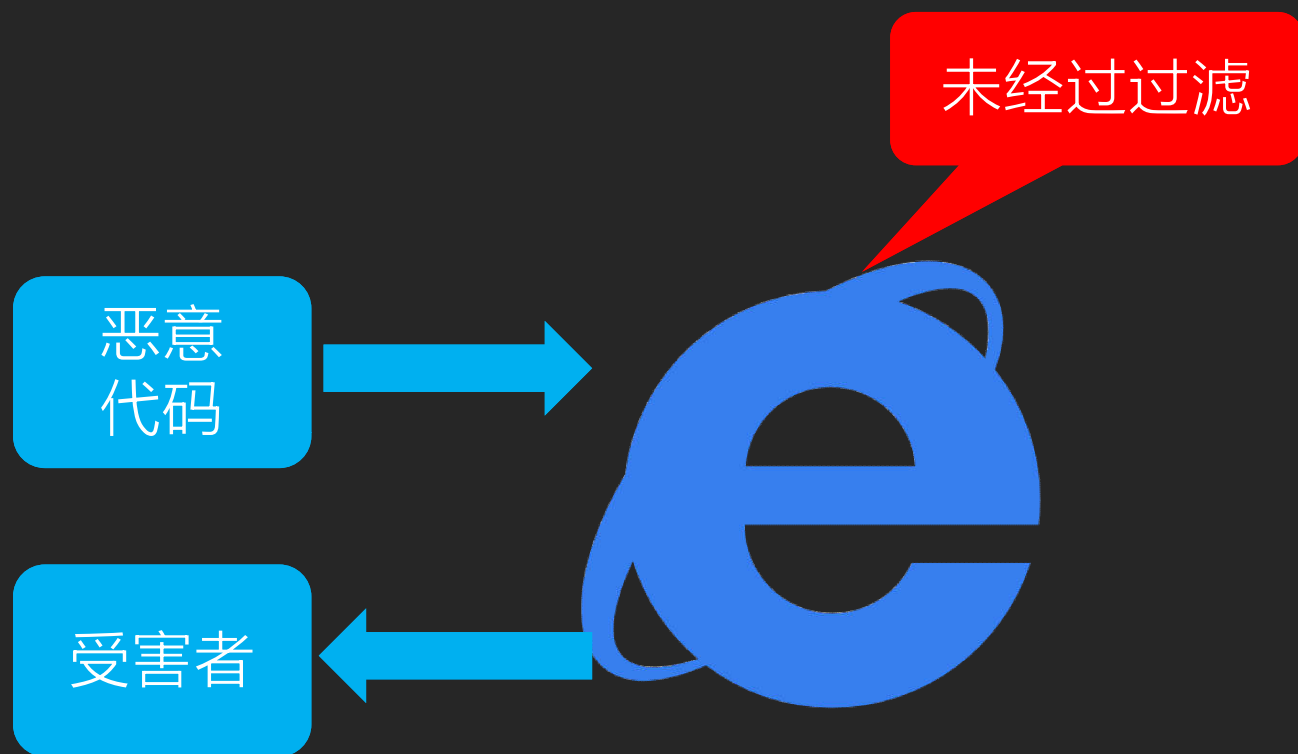
● 反射型XSS



特点

攻击代码被服务器原样返回

● DOM-based型XSS



特点

攻击代码不经过服务器，
直接在本地触发

● XSS盲打



一种非常猥琐的攻击方法，容易被忽略，**但危害极大**

- 攻击者无法直接测试网站是否存在XSS漏洞
- 一般是通过提交恶意代码，然后通过恶意代码是否被触发来判断有没有XSS漏洞
- 典型的场景是用户在前台提交资料到管理后台审核，管理后台审核页面没有过滤，导致XSS
- 案例：

我是如何通过一个 XSS 探测搜狐内网扫描内网并且蠕动到前台的

● 一般的XSS防御方法

- 严格校验用户的输入，例如：

```
String id =  
request.getParameter("id")
```



```
int id =  
Integer.parseInt(request.getParameter("id"))
```

```
String email =  
request.getParameter("email")
```



```
String email = request.getParameter("email");  
String regex = "[0-9a-zA-Z]+@[0-9a-zA-Z-  
Z]+\\. [0-9a-zA-Z-  
Z]+";  
Pattern pattern = Pattern.compile(regex);  
Matcher matcher = pattern.matcher(email);  
if(!matcher.matches()){  
    out.println("非法的Email");  
    return;  
}
```

● 一般的XSS防御方法

- 对于无法校验的输入，在输出之前进行转义，例如：

```
<html>
  <body>
    <%=request.getParameter("a")%>
  </body>
</html>
```

```
<%@ page import="org.apache.commons.lang.StringEscapeUtils" %>
<html>
  <body>
    <%=StringEscapeUtils.escapeXml(request.getParameter("a"))%>
  </body>
</html>
```



用上面的方法转义之后就安全了吗？



● XSS转义方案总结

- **HTML编码**: 把特殊字符转义成HTML实体, 例如把 < 编码为 <
- **Javascript编码**: 把字符串转成十六进制\xhh的形式, 例如把 < 编码为 \x3c
- **URL编码**: 把特殊字符转成%hh的格式, 例如把< 编码为 %3c

● XSS转义方案总结

● 场景一：

在HTML标签中输出，如：

```
<div>$var</div>
```

● 修复：

对var参数进行HTML编码

● XSS转义方案总结

● 场景二：

在HTML属性中输出，如：

```
<div name="$var" ></div>
```

● 修复：

对var参数进行HTML编码

● XSS转义方案总结

● 场景三:

在<script>标签中输出, 如:

```
<script>var x="$var";</script>
```

● 修复:

对var参数进行Javascript编码

● XSS转义方案总结

● 场景四：

在事件中或javascript伪协议中输出，如：

```
<a href=# onclick="func('$var')" >test</a>
```

```
<a href="javascript:func('$var')" >test</a>
```

● 修复：

对var参数进行Javascript编码

● XSS转义方案总结

● 场景五：

在全地址中输出，如：

```
<a href="$var" >test</a>
```

● 修复：

先检查var参数是不是以http： 开头，然后再对其进行URL编码。

命令注入

可造成机器入侵

● 命令注入漏洞原理

原语句:

```
Runtime.getRuntime().exec("ping " + request.getParameter("ip"));
```

正常情况下的请求(ping.jsp?ip=127.0.0.1),相当于:

```
Runtime.getRuntime().exec("ping 127.0.0.1");
```

如果请求为: ping.jsp?ip=127.0.0.1|cat /etc/passwd,那么结果将会是:

```
Runtime.getRuntime().exec("ping 127.0.0.1|cat /etc/passwd");
```

● 命令注入漏洞防御

- 严格校验输入的参数，例如前面的案例中，检测参数是否是IP或域名，否则，直接返回错误
- 非必要情况下，不要使用可以执行系统命令的方法，尤其是执行的命令或者参数外界可控时，使用更应该谨慎。

后台弱口令

容易忽略的重大漏洞

● 后台弱口令漏洞介绍

● 原理

主要指网站后台使用弱密码，且安全防护措施不足，导致可以通过暴力枚举、撞库等方式进行攻击。

● 案例

1. [中国电信某系统后台弱口令](#)
2. [中国电信北京研究院安全网关后台弱口令](#)
3. [中国电信广西公司某系统弱口令泄露大量员工信息](#)

● 后台弱口令漏洞的防御

- 禁止用户使用弱密码；
- 增加图片验证码；
- 后台增加频率限制和出错次数限制。

* 密码

❗ 密码过于简单，请尝试“字母+数字”的组合

* 确认密码

请再次填写密码

* 验证码

请填写图片中的字符，不区分大小写 看不清楚？换张图片

> 登录

您尝试的错误密码个数过多，为了您的帐号安全，请通过手机验证解除限制

帐 号:

密 码:



三、常见Android漏洞

Android的安全机制

● Android的安全机制

- Android 是一个权限分离的系统，这是利用 Linux 已有的权限管理机制，通过为每一个应用分配不同的 uid 和 gid，从而使不同的应用之间的私有数据和访问达到隔离的目的。

```
app_36    app_36    2014-08-21 12:26 com.google.android.music
app_39    app_39    2014-08-21 12:26 com.google.android.onetimeinitializer
app_22    app_22    2014-08-21 12:25 com.google.android.partnersetup
app_44    app_44    2014-08-21 12:25 com.google.android.setupwizard
app_47    app_47    2014-08-21 12:25 com.google.android.street
app_10    app_10    2014-08-21 12:25 com.google.android.syncadapters.bookmarks
app_10    app_10    2014-08-21 12:25 com.google.android.syncadapters.contacts
app_48    app_48    2014-08-21 12:25 com.google.android.tag
app_24    app_24    2014-08-21 12:25 com.google.android.tts
app_55    app_55    2014-08-21 12:28 com.google.android.voicesearch
```

● Permission机制

- Permission机制主要是用来对应用可以执行的某些具体操作进行权限细分和访问控制
- 每个权限通过 `protectionLevel` 来标识保护级别：
 - `normal`: 安装时即授予权限，不需要确认
 - `dangerous`: 危险的权限，需要用户安装时确认授予
 - `signature`: 具有相同签名的APP才能获取的权限
 - `signatureOrSystem`: 具有相同签名或者是放在system image中的APP才能获取的权限。

● Permission机制

● 定义Permission:

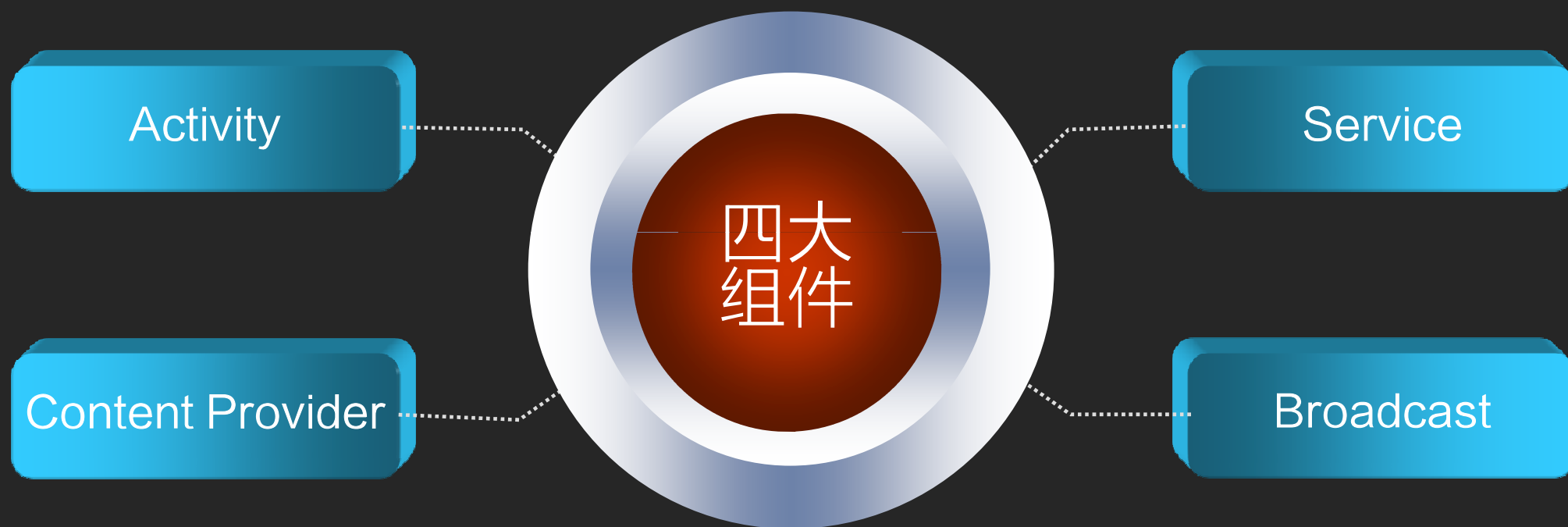
```
<permission  
    android:name="com.yourapp.permission"  
    android:protectionLevel="signature" />
```

● 使用Permission:

```
<uses-permission android:name="com.yourapp.permission" />
```

Android的组件安全

● Android的四大组件



● 组件公开的安全风险

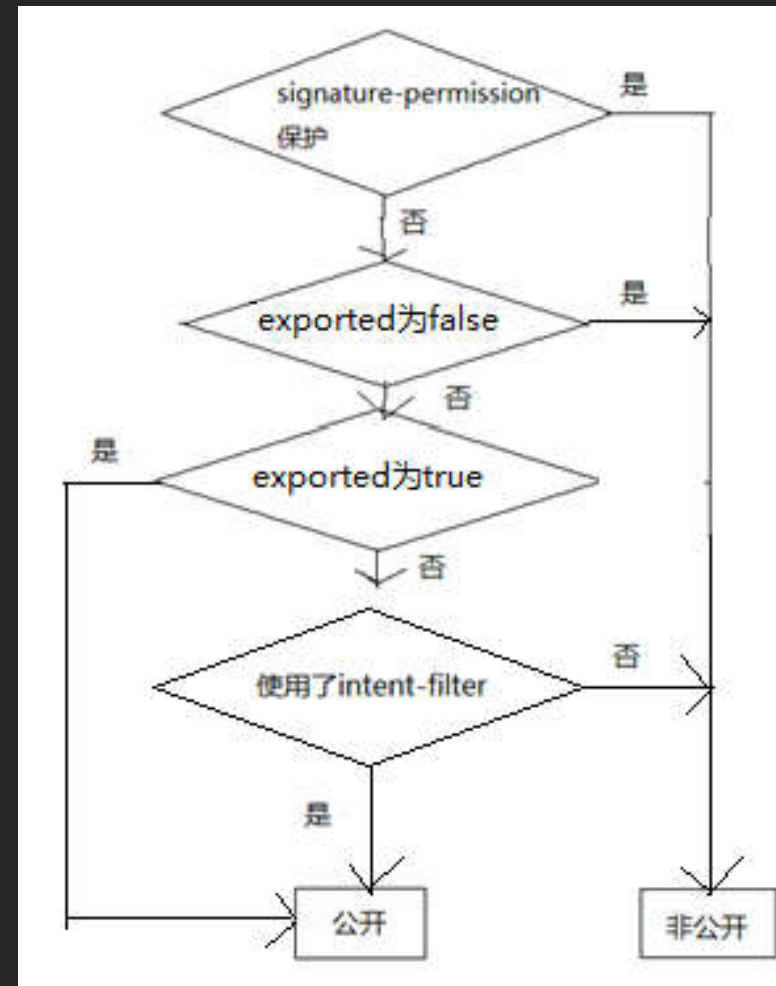
- 所谓**组件公开**是指应用定义的组件可以被第三方应用所调用；
- 组件公开存在以下安全风险：
 - 1、导致敏感操作被恶意调用；
 - 2、导致敏感信息泄露；
 - 3、导致应用Crash。

● Activity安全

```
<activity
    android:name=".activity.SettingActivity"
    android:exported="false"
    android:permission="com.yourapp.permission" >
    <intent-filter>
        <action android:name="com.xxx" />
        <category android:name="com.yyy" />
    </intent-filter>
</activity>
```

● 判断一个Activity是否公开的三要素：

- 1、exported属性
- 2、是否定义了intent-filter
- 3、是否限制了permission



● Service安全

```
<service
    android:name=".ConnectionService"
    android:exported="false"
    android:permission="com.yourapp.permission" >
    <intent-filter>
        <action android:name="com.xxx" />
        <category android:name="com.yyy" />
    </intent-filter>
</service>
```

- Service是在后台运行的，一般会把一些敏感操作放在这里，如和服务器的通讯，退出等，如果Service是公开的，这些敏感操作就可能被恶意调用，而且还有泄露敏感信息的风险；
- 判断Service是否公开，与Activity的判断过程是一样的。

● Broadcast安全

- Broadcast为分发送端和接收端：

接收端：负责对注册的广播进行响应，然后进行处理；

发送端：负责把广播发送出去，让别人去响应。

- 为避免接收端的操作被恶意调用，需要把Broadcast设置为非公开；
- 为了避免敏感信息泄漏，发送端需要设置签名级别的权限。

● Broadcast安全

● Broadcast接收端分两种：

1、在AndroidManifest.xml中声明的receiver

```
<receiver  
    android:name=".my.broadcast"  
    android:exported="false"  
    android:permission="com.yourapp.permission" >  
    <intent-filter>  
        <action android:name="com.xxx" />  
        <category android:name="com.yyy" />  
    </intent-filter>  
</receiver>
```

在AndroidManifest.xml中声明的receiver，它是否公开的判断过程和Activity是一样的。

● Broadcast安全

2、在代码中动态注册的receiver

有两种注册方法：

(1) `Context. RegisterReceiver(BroadcastReceiver receiver, IntentFilter filter)`

用此方法注册的广播是公开的。

(2) `Context. RegisterReceiver(BroadcastReceiver receiver, IntentFilter filter, String broadcastPermission, Handler scheduler)`

用此方法注册的广播，如果broadcastPermission是签名级别的，那么是非公开的，否则就是公开的。

● Content Provider安全

```
<provider
    android:name=".my.provider"
    android:exported="false"
    android:permission="com.yourapp.permission" >
</activity>
```

- Content Provider提供了一种在多个应用间进行数据共享的方式，应用可以利用Content Provider完成对数据的增删改查；
- 所以在Content Provider中不要出现敏感信息，以免被其他应用窃取和修改。

● 如何提高组件使用的安全性

- 公开的组件是存在风险的，如非特殊需要，应该把组件设置成非公开；
- 涉及到敏感操作和敏感信息的组件，应该设置成非公开；
- 对于需要公开的组件，要处理异常情况，避免Crash的发生。

Android的数据安全

● Intent劫持

- Intent是组件间通信的一种机制。如：

```
Intent myIntent = new Intent("com.my.action");  
myIntent.putExtra("password", "123456");  
sendBroadcast(myIntent);
```

- Intent分显式Intent和隐式Intent

1、**显式Intent**：指明了通讯目标的包名和类名，如：

```
Intent myIntent = new Intent();  
myIntent.setClassName("com.xxx", "com.xxx.activity.myActivity");
```

2、**隐式Intent**：未指明通讯目标。

隐式Intent会被其他应用劫持，从而获取该Intent中所存放的信息，所以不要在隐式Intent存放敏感信息。

● 敏感信息明文保存

- 指Android应用将敏感信息明文保存在本地存储中。
- Android应用存储数据的地方：

1、私有目录（ /data/data/package_name/ ），只有相同UID的应用才有权限访问

在私有目录下创建文件时的三种模式：

MODE_PRIVATE: 默认模式，只有文件的所有者对这个文件有控制权

MODE_WORLD_WRITABLE: 所有应用可写

MODE_WORLD_READABLE: 所有应用可读

2、SDCard，所有应用都可读可写

Android的其它安全

● WebView的远程命令执行漏洞

- 指Webview的Javascript导出接口可导致远程命令执行。
- 利用方法：

1、假设Android应用注册了以下Javascript接口：

```
myWebView.addJavascriptInterface(new MyJsInterface(this), "test");
```

2、那么就可以用Javascript实现命令执行：

```
test.getClass().forName("java.lang.Runtime").  
    getMethod("getRuntime",null).invoke(null,null).exec(cmd);
```

● WebView的远程命令执行漏洞

- 除了调用addJavascriptInterface导出的接口受影响外，Android系统（大于3.0，小于4.2）本身的webkit还默认提供另一个接口searchBoxJavaBridge_，同样可通过反射方法来远程执行代码；
- 故开发中应注意：
 - 1、禁止使用WebView的addJavascriptInterface这个函数；
 - 2、移除系统自带的默认接口 searchBoxJavaBridge_。
- 可通过以下链接检查Webview是否有问题：

http://security.tencent.com/lucky/check_tools.html

● 逻辑漏洞

- 找回密码功能存在逻辑漏洞；

- 常见的问题：

- 1、提交手机验证码和修改密码分两个请求进行，导致验证码校验被绕过；
- 2、把验证码加密后返回到客户端，在本地验证；
- 3、手机验证码校验功能没有做频率限制和出错次数限制，导致可以被暴力破解。



四、安全开发总结

● 树立正确的安全意识

- 一、**非常重要：** 使用任何外部数据之前，先进行严格校验或过滤
 - （1）整型数据：先强制转换为整型；
 - （2）字符串型数据：按参数用途严格校验合法性，如邮箱，校验是否符合邮箱格式；
 - （3）数组型数据：对每一个数组元素进行如上过滤。
- 二、严禁使用拼接SQL语句查询数据库，使用参数化查询。
- 三、用白名单代替黑名单。
- 四、禁止使用弱口令。

谢谢！

END