说明: 无标题-4说明: 无标题-4

|  |  |  |  |
| --- | --- | --- | --- |
| **Web应用安全测试指南** | | | |
| ■ **文档编号** | ST-PEN-02-01-0001 | ■ **密级** | 内部使用 |
| ■ **版本编号** | V1.1 | ■ **日期** | 2017-08-17 |



|  |
| --- |
| **© 2017 绿盟科技** |

|  |
| --- |
| ■ **版权声明** |
| 本文中出现的任何文字叙述、文档格式、插图、照片、方法、过程等内容，除另有特别注明，版权均属**绿盟科技**所有，受到有关产权及版权法保护。任何个人、机构未经绿盟科技的书面授权许可，不得以任何方式复制或引用本文的任何片断。 |

|  |  |  |  |
| --- | --- | --- | --- |
| ■ **版本变更记录** | | | |
| **时间** | **版本** | **说明** | **修改人** |
| 2017年7月7日 | V1.0 | 文档创建 | 业务标准化小组 |
| 2017年8月7日 | V1.1.0 | 新增内容 | 业务标准化小组 |
| 2017年8月17日 | V1.1.1 | 新增内容、调整格式 | 业务标准化小组 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **■ 适用性声明** |
| 本文档为北京神州绿盟科技有限公司（以下简称“绿盟科技”）安全测试服务标准化小组提供的Web应用安全测试指南，适用于实施交付人员，并用于学习Web应用安全测试时的参考资料。 |

|  |
| --- |
| 目录 |
| [一. 概述 8](#_Toc498434465)  [1.1 编写目的 8](#_Toc498434466)  [1.2 适用范围 8](#_Toc498434467)  [1.3 注意事项 8](#_Toc498434468)  [二. Web应用安全测试指南 8](#_Toc498434469)  [2.1 信息泄漏 8](#_Toc498434470)  [2.1.1 robots.txt泄漏敏感信息 8](#_Toc498434471)  [2.1.2 敏感文件信息泄漏 10](#_Toc498434472)  [2.1.3 过时的、用于备份的或者开发文件残留 11](#_Toc498434473)  [2.1.4 报错页面敏感信息泄漏 12](#_Toc498434474)  [2.1.5 物理路径泄漏 13](#_Toc498434475)  [2.1.6 明文密码本地保存 14](#_Toc498434476)  [2.1.7 入侵痕迹残留 14](#_Toc498434477)  [2.1.8 HTTP头信息泄漏 15](#_Toc498434478)  [2.1.9 目录浏览 15](#_Toc498434479)  [2.1.10 默认页面泄漏 16](#_Toc498434480)  [2.1.11 存在可访问的管理后台入口 17](#_Toc498434481)  [2.1.12 存在可访问的管理控制台入口 18](#_Toc498434482)  [2.1.13 参数溢出 23](#_Toc498434483)  [2.1.14 任意文件下载 24](#_Toc498434484)  [2.2 信息猜解 26](#_Toc498434485)  [2.2.1 邮件内容中请求链接可预测 26](#_Toc498434486)  [2.3 数据猜解 27](#_Toc498434487)  [2.3.1 账号枚举 27](#_Toc498434488)  [2.3.2 账号密码共用 28](#_Toc498434489)  [2.4 认证信息泄漏 29](#_Toc498434490)  [2.4.1 传输过程泄漏 29](#_Toc498434491)  [2.4.2 会话变量泄漏 30](#_Toc498434492)  [2.5 认证信息猜解 30](#_Toc498434493)  [2.5.1 存在弱口令 30](#_Toc498434494)  [2.5.2 存在暴力破解 31](#_Toc498434495)  [2.6 认证功能失效 33](#_Toc498434496)  [2.6.1 存在空口令 33](#_Toc498434497)  [2.6.2 认证绕过 33](#_Toc498434498)  [2.6.3 Oauth认证缺陷 35](#_Toc498434499)  [2.6.4 IP地址伪造 37](#_Toc498434500)  [2.7 认证功能滥用 37](#_Toc498434501)  [2.7.1 多点认证缺陷 37](#_Toc498434502)  [2.7.2 会话固定 38](#_Toc498434503)  [2.8 业务逻辑篡改 40](#_Toc498434504)  [2.8.1 密码修改/重置流程跨越 40](#_Toc498434505)  [2.8.2 负值反冲 41](#_Toc498434506)  [2.8.3 正负值对冲 42](#_Toc498434507)  [2.8.4 业务流程跳跃 43](#_Toc498434508)  [2.9 业务功能失效 43](#_Toc498434509)  [2.9.1 通配符注入 43](#_Toc498434510)  [2.10 业务功能滥用 45](#_Toc498434511)  [2.10.1 短信定向转发 45](#_Toc498434512)  [2.10.2 邮件可定向转发 46](#_Toc498434513)  [2.10.3 业务接口调用缺陷 46](#_Toc498434514)  [2.10.4 IMAP/SMTP注入 47](#_Toc498434515)  [2.10.5 引用第三方不可控脚本/URL 49](#_Toc498434516)  [2.10.6 开启危险接口 50](#_Toc498434517)  [2.10.7 未验证的URL跳转 50](#_Toc498434518)  [2.10.8 服务器端请求伪造（SSRF） 52](#_Toc498434519)  [2.10.9 短信内容可控 56](#_Toc498434520)  [2.10.10 邮件内容可控 57](#_Toc498434521)  [2.10.11 请求重放攻击 57](#_Toc498434522)  [2.10.12 批量提交 58](#_Toc498434523)  [2.11 防护功能失效 59](#_Toc498434524)  [2.11.1 账号弱锁定机制 59](#_Toc498434525)  [2.11.2 图形验证码可自动获取 59](#_Toc498434526)  [2.11.3 图形验证码绕过 60](#_Toc498434527)  [2.11.4 短信验证码绕过 61](#_Toc498434528)  [2.11.5 短信验证码可暴力破解 62](#_Toc498434529)  [2.11.6 参数覆盖 63](#_Toc498434530)  [2.11.7 关键逻辑判断前端验证 66](#_Toc498434531)  [2.12 防护功能缺失 67](#_Toc498434532)  [2.12.1 Cookie属性问题 67](#_Toc498434533)  [2.12.2 会话重用 69](#_Toc498434534)  [2.12.3 会话失效时间过长 70](#_Toc498434535)  [2.13 防护功能滥用 71](#_Toc498434536)  [2.13.1 恶意锁定问题 71](#_Toc498434537)  [2.13.2 短信炸弹 71](#_Toc498434538)  [2.13.3 邮件炸弹 72](#_Toc498434539)  [2.14 权限缺失 73](#_Toc498434540)  [2.14.1 Flash跨域访问 73](#_Toc498434541)  [2.14.2 jsonp跨域请求 76](#_Toc498434542)  [2.14.3 未授权访问 77](#_Toc498434543)  [2.15 权限篡改 80](#_Toc498434544)  [2.15.1 任意用户密码修改/重置 80](#_Toc498434545)  [2.15.2 SSO认证缺陷 82](#_Toc498434546)  [2.15.3 越权 83](#_Toc498434547)  [2.15.4 Cookies伪造 85](#_Toc498434548)  [2.15.5 会话变量可控 86](#_Toc498434549)  [2.15.6 跨站请求伪造（CSRF） 86](#_Toc498434550)  [2.16 综合利用 91](#_Toc498434551)  [2.16.1 跨站脚本攻击（XSS） 91](#_Toc498434552)  [2.16.2 FLASH跨站脚本攻击 105](#_Toc498434553)  [2.16.3 HTTP响应分割 106](#_Toc498434554)  [2.16.4 HTTP参数污染 109](#_Toc498434555)  [2.16.5 Host头攻击 110](#_Toc498434556)  [2.16.6 SQL注入 112](#_Toc498434557)  [2.16.7 NoSQL注入 116](#_Toc498434558)  [2.16.8 LDAP注入 118](#_Toc498434559)  [2.16.9 XML注入 119](#_Toc498434560)  [2.16.10 XXE 121](#_Toc498434561)  [2.16.11 XPATH注入 123](#_Toc498434562)  [2.16.12 命令注入 125](#_Toc498434563)  [2.16.13 任意文件上传 127](#_Toc498434564)  [2.16.14 反序列化漏洞 131](#_Toc498434565)  [2.17 专项漏洞 132](#_Toc498434566)  [2.17.1 Web组件（SSL/WebDAV）漏洞 132](#_Toc498434567)  [2.17.2 中间件相关漏洞 133](#_Toc498434568)  [2.17.3 第三方应用相关漏洞 135](#_Toc498434569)  [2.17.4 第三方插件相关漏洞 138](#_Toc498434570)  [2.17.5 开发框架 139](#_Toc498434571)  [2.17.6 通用型系统 143](#_Toc498434572)  [三. 致谢 143](#_Toc498434573)  [附录A 漏洞分类依据 144](#_Toc498434574)  [附录B 名词解释 144](#_Toc498434575)  [附录C 风险等级依据 145](#_Toc498434576) |

1. 概述
   1. 编写目的

结合公司的内部人员培养体系，本手册旨在为安全测试人员提供测试指导，安全测试人员通过查阅该指南可快速掌握Web应用安全测试，提高工作能力。

* 1. 适用范围

本手册适用于Web应用安全测试人员。

* 1. 注意事项

本手册旨在为测试人员提供漏洞测试的基本思路，随着安全技术的发展，更多新的漏洞和测试方法将被爆出，安全测试人员应具备不断学习新知识的能力。

1. Web应用安全测试指南
   1. 信息泄漏
      1. robots.txt泄漏敏感信息

**漏洞描述：**搜索引擎可以通过robots文件可以获知哪些页面可以爬取，哪些页面不可以爬取。Robots协议是网站国际互联网界通行的道德规范，其目的是保护网站数据和敏感信息、确保用户个人信息和隐私不被侵犯，如果robots.txt文件编辑的太过详细，反而会泄露网站的敏感目录或者文件，比如网站后台路径，从而得知其使用的系统类型，从而有针对性地进行利用。

**测试方法：**

1、检测形式多样，工具爬虫扫描得到敏感文件的路径，从而找到robots文件；

2、手工挖掘，直接在域名后输入/robots.txt进行查看。

**风险分析：**攻击者可通过发现robots.txt文件，收集网站的敏感目录或文件，从而有针对性的进行利用。

**风险等级：**

【**低危**】：robots.txt中存在allow和disallow的具体内容泄露敏感目录信息。

**修复方案：**可根据实际情况，进行如下对应的修复：

1、 User-agent: \* 这里的\*代表的所有的搜索引擎种类，\*是一个通配符

2、 Disallow: / 这里定义是禁止爬寻站点所有的内容

3、 Disallow: /admin/ 这里定义是禁止爬寻admin目录下面的目录

4、 Disallow: /ABC/ 这里定义是禁止爬寻ABC目录下面的目录

5、 Disallow: /cgi-bin/\*.htm 禁止访问/cgi-bin/目录下的所有以".htm"为后缀的URL(包含子目录)。

6、 Disallow: /\*?\* 禁止访问网站中所有包含问号 (?) 的网址

7、 Disallow: /.jpg$ 禁止抓取网页所有的.jpg格式的图片

8、 Disallow:/ab/adc.html 禁止爬取ab文件夹下面的adc.html文件。

9、 Allow: /cgi-bin/　这里定义是允许爬寻cgi-bin目录下面的目录

10、Allow: /tmp 这里定义是允许爬寻tmp的整个目录

11、Allow: .htm$ 仅允许访问以".htm"为后缀的URL。

12、Allow: .gif$ 允许抓取网页和gif格式图片

13、Sitemap: 网站地图 告诉爬虫这个页面是网站地图。

**注意事项：**暂无

* + 1. 敏感文件信息泄漏

**漏洞描述：**敏感数据包括但不限于：口令、密钥、证书、会话标识、License、隐私数据（如短消息的内容）、授权凭据、个人数据（如姓名、住址、电话等）等，在程序文件、配置文件、日志文件、备份文件及数据库中都有可能包含敏感数据。

**测试方法：**

1. 检测形式多样，工具爬虫扫描得到敏感文件的路径，从而找到敏感数据，
2. 手工挖掘，根据web容器或者网页源代码的查看，找到敏感信息。

**风险分析：**攻击者可通过上述方式获取网站敏感文件，收集网站敏感信息，从而有针对性的进行利用。

**风险等级：**

【**高危**】：返回含有重要的敏感信息的文件，比如数据库文件、代码的备份文件或svn、git版本控制文件等。

**修复方案：**

1. 禁止在代码中存储敏感数据：禁止在代码中存储如数据库连接字符串、口令和密钥之类的敏感数据，这样容易导致泄密。用于加密密钥的密钥可以硬编码在代码中。
2. 禁止密钥或帐号的口令以明文形式存储在数据库或者文件中：密钥或帐号的口令必须经过加密存储。例外情况，如果Web容器的配置文件中只能以明文方式配置连接数据库的用户名和口令，那么就不用强制遵循该规则，将该配置文件的属性改为只有属主可读写。
3. 禁止在 cookie 中以明文形式存储敏感数据：cookie信息容易被窃取，尽量不要在cookie中存储敏感数据；如果条件限制必须使用cookie存储敏感信息时，必须先对敏感信息加密再存储到cookie。
4. 禁止在隐藏域中存放明文形式的敏感数据。
5. 禁止用自己开发的加密算法，必须使用公开、安全的标准加密算法。
6. 禁止在日志中记录明文的敏感数据：禁止在日志中记录明文的敏感数据（如口令、会话标识jsessionid等）， 防止敏感信息泄漏。
7. 禁止带有敏感数据的Web页面缓存：带有敏感数据的Web页面都应该禁止缓存，以防止敏感信息泄漏或通过代理服务器上网的用户数据互窜问题。

**注意事项：**暂无

* + 1. 过时的、用于备份的或者开发文件残留

**漏洞描述：**应用遗留的过时文件、备份页面、渗透测试遗留文件、开发文件残留的测试文件等。

**测试方法：**

1. 常见检测方法是通过对网站进行web漏洞扫描，直接利用爬虫来爬取网站可能存在的路径以及链接，如果存在备份文件，则可通过web直接进行下载。
2. 也可以通过自行构造字典，对网站某一目录下，指定字典进行爆破，常见的扫描工具有wwwscan、御剑后台扫描工具等。

**风险分析：**攻击者可通过上述方式获取网站备份文件、过时文件、遗留文件等内容，收集网站敏感信息，从而有针对性的进行利用。

**风险等级：**

【**高危**】：泄露重要敏感信息，或能够进行核心业务操作。

【**中危**】：泄露一般重要信息，做只能进行一般功能操作。

【**低危**】：页面泄露非重要信息，不能进行相关功能操作。

**修复方案：**

1. 网站管理员严格检查web中可访问的路径下是否存在备份文件，常见备份文件后缀为.jsp.bak、.bak、.sql、.txt、等等。如果有这些文件，直接将该备份文件进行转移到其他目录或者直接删除即可。
2. 严格控制可网站可访问目录下的文件敏感度的存放问题，不要将敏感文件置于该目录。

**注意事项：**暂无

* + 1. 报错页面敏感信息泄漏

**漏洞描述：**错误页面由服务器产生403、404、500等错误时，返回详细错误信息。报错信息中可能会包含服务器代码信息、数据库连接信息、SQL语句或者敏感文件的路径，为攻击者收集信息提供了方便。

**测试方法：**

1. 通过目录扫描或手工输入不存在的文件或路径，触发服务器产生404错误并返回404页面；
2. 通过目录扫描或手工输入一个无权限查看的文件或路径，触发服务器产生403错误并返回403页面；
3. 手工输入不存在的参数或特殊构造的字符串，如单引号，尖括号等，触发服务器产生500错误并返回500页面或异常信息。

**风险分析：**攻击者可通过上述几种方式触发Web应用程序报错，提取报错信息泄露的敏感信息，如Web中间件的版本信息、数据库连接信息。

**风险等级：**

【**高危**】：开启调试模式，泄露大量应用的敏感信息如代码、报错信息等；

【**低危**】：未开启调试模式，泄露部分中间件版本、少量代码信息等。

**修复方案：**

1. 编码时增加异常处理模块，对错误页面做统一的自定义返回界面，隐藏服务器版本信息；
2. 不对外输出程序运行时产生的异常错误信息详情。

**注意事项：**暂无

* + 1. 物理路径泄漏

**漏洞描述：**应用中泄露出应用在主机中的绝对地址路径。

**测试方法：**

1. 打开网页源代码，查看图片等媒体的链接及超链接；
2. 通过报错信息获取

**风险分析：**攻击者可通过获取网站物理路径，为下一步攻击做准备。

**风险等级：**

【**低危**】：泄露应用绝对路径

**修复方案：**

1. 媒体链接和超链接采用相对路径的表达方式；
2. 报错信息中不对外输出网站物理路径等敏感信息。

**注意事项：**暂无

* + 1. 明文密码本地保存

**漏洞描述：**明文密码保存在本地客户端

**测试方法：**

1. 查看网页源代码
2. 查看网站在本地客户端的缓存文件

**风险分析：**攻击者可通过嗅探或直接查看源代码的方式获取传输到前端的账号及密码，登录他人账号。

**风险等级：**

【**高危**】：全部账号的明文密码保存在本地客户端

【**低危**】：只有本账号的明文密码保存在本地客户端

**修复方案：**禁止将密码保存到本地客户端，即便是加密后的密码也不建议保存在本地，攻击者可利用密文格式的密码登录或修改其他账户的密码。

**注意事项：**暂无

* + 1. 入侵痕迹残留

**漏洞描述：**在渗透过程中发现应用中存在曾经的入侵痕迹，如存在的webshell文件。

**测试方法：**通常使用Web应用安全漏洞扫描工具或目录扫描工具发现入侵痕迹。

**风险分析：**残留的入侵痕迹可被其他攻击者用于二次攻击，对网站造成一定的影响。

**风险等级：**

【**高危**】：发现存在入侵痕迹，比如存在Webshell或者网页被篡改。

**修复方案：**可借助工具全盘清理入侵痕迹，如D盾可以扫描Windows系统中的webshell。

**注意事项：**暂无

* + 1. HTTP头信息泄漏

**漏洞描述：**在服务器返回的HTTP头中泄露服务器信息

**测试方法：**

1. 在浏览器的调试窗口中查看HTTP响应头
2. 使用代理软件如burpsuite、fiddler，拦截HTTP响应头

**风险分析：**攻击者可通过获取服务器banner信息，针对某个版本存在的漏洞进行定向攻击。

**风险等级：**

【**低危**】：泄露服务器版本等信息

**修复方案：**隐藏或者修改banner信息**。**

**注意事项：**暂无

* + 1. 目录浏览

**漏洞描述：**目录浏览漏洞是由于网站存在配置缺陷，存在目录可浏览漏洞，这会导致网站很多隐私文件与目录泄露，比如数据库备份文件、配置文件等，攻击者利用该信息可以更容易得到网站权限，导致网站被黑。

**测试方法：**可以利用web漏洞扫描器扫描web应用进行检测，也可通过搜索，网站标题包含“index of”关键词的网站进行访问。

**风险分析：**攻击者通过访问网站某一目录时，该目录没有默认首页文件或没有正确设置默认首页文件，将会把整个目录结构列出来，将网站结构完全暴露给攻击者； 攻击者可能通过浏览目录结构，访问到某些隐秘文件（如PHPINFO文件、服务器探针文件、网站管理员后台访问地址、数据库连接文件等）。

**风险等级：**

【**高危**】：目录可以浏览，泄露包含密码、个人信息等敏感文件。

【**低危**】：目录可以浏览，未泄露包含密码、个人信息等敏感文件。

**修复方案：**目前存在该漏洞的常见中间件为apache和IIS,以下列出其相关的修复方式：

1、 IIS中关闭目录浏览功能：在IIS的网站属性中，勾去“目录浏览”选项，重启IIS。

2、 Apache中关闭目录浏览功能：打开Apache配置文件httpd.conf，查找“Options Indexes FollowSymLinks”，修改为“ Options -Indexes”(减号表示取消，保存退出，重启Apache。

3、 Nginx中默认不会开启目录浏览功能，若您发现当前已开启该功能，可以编辑nginx.conf文件，删除如下两行：autoindex on;autoindex\_exact\_size on;重启Nginx。

**注意事项：**暂无

* + 1. 默认页面泄漏

**漏洞描述：**存在默认安装中间件、插件、框架等会携带示例页面及说明文档。

**测试方法：**

1. 可以利用web漏洞扫描器或目录扫描器扫描web应用进行检测
2. 根据网站使用的第三方组件和框架手工输入对应的示例页面。

**风险分析：**攻击者可利用默认页面提供的功能和信息对服务器进行攻击。

**风险等级：**

【**高危**】：存在可访问默认页面，泄露高风险敏感信息（如：tomcat 的 examples 目录）。

【**中危**】：存在可访问默认页面，泄露于业务、操作和配置相关的敏感信息。

【**低危**】：存在可访问的默认页面，但未泄露敏感信息。

**修复方案：**建议在不影响业务的前提下删除默认页面。

**注意事项：**暂无

* + 1. 存在可访问的管理后台入口

**漏洞描述：**应用存在未限制访问的后台，或者能直接登录管理后台。

**测试方法：**

1. 可以利用web漏洞扫描器或目录扫描器扫描web应用进行检测
2. 识别网站使用的cms框架，判断其默认的管理后台地址。
3. 在网站中寻找管理后台超链接。

**风险分析：**攻击者可通过登录网站管理后台篡改页面，或利用上传功能上传webshell，导致服务器被控制。

**风险等级：**

**【高危**】：可访问默认管理后台，通过后台获取 shell。

【**中危**】：可访问默认管理后台，并成功登录，但无法获取 shell。

【**低危**】：可访问默认管理后台，但无法登录或执行危险操作。

**修复方案：**建议在不影响业务的前提下，将管理后台隐藏在非常规目录下或增加管理后台的访问限制。

**注意事项：**暂无

* + 1. 存在可访问的管理控制台入口

**漏洞描述：**Web 控制台是一种基于 Web 的用户界面， 其常常被用于网站后台或者web容器控制台中，其不仅仅局限于容器或者网站管理后台，还包括一些数据库默认地址等。在web安全中，网站系统在泄漏其web容器（中间件）或者数据库的控制台后，存在增加被入侵的风险。常见的web控制台包括以下多种：tomcat、aria2、weblogic、websphere、oracle、jboss、等。这些web的容器控制台常见访问形式：http://hostname:port/load/，例如：http://x.x.x.x:8080/manage/。

**测试方法：**常见的web控制台检测方法：整体思路为首先需识别网站容器的指纹，判断其所采用的中间件，然后去扫描其所开放的端口，根据开放端口信息和常见固定的路径，去判断其控制台地址。以下列举常见控制台的检测方法：

1. Apache+tomcat：tomcat常见的web控制台地址为：http://x.x.x.x/manager/html或者添加端口：http://x.x.x.x:8080/manager/html，从TOMCAT5（开始默认/admin后台不存在，tomcat5之前的控制台为/admin。
2. Weblogic控制台：http://[weblogic所在机器IP]:[weblogic端口]/console若没有指定端口，且安装在本机上则为：（weblogic默认端口为7001）http://localhost:7001/console。
3. Websphere控制台：websphere的控制台常见有两种，一种是基于http，另一种是基于https的，分别为如下：http://localhost:9060/ibm/console和https://localhost:9043/ibm/console/logon.jsp。
4. Oracle web控制台：一般默认的是http://localhost:5500/em,一般存放于Oracle安装文件夹下的install文件夹下中文本文件，上面有web控制台的地址。
5. Mongodb web控制台：自带了Web控制台：默认和数据服务一同开启。他的端口在Mongodb数据库服务器端口的基础上加1000，如果是默认的Mongodb数据服务端口(Which is 27017)，则相应的Web端口为28017，这个页面可以看到当前Mongodb的所有连接、各个数据库和Collection的访问统计，包括：Reads, Writes, Queries, GetMores ,Inserts, Updates, Removes、写锁的状态、以及日志文件的最后几百行（CentOS+10gen yum 安装的mongodb默认的日志文件位于/var/log/mongo/mongod.log)。
6. HP system managent控制台：该控制台一般默认的端口为2381，可在其后添加路径/cpqlogin.php?errno=100&severity=4，即可访问.https://localhost:2381/cpqlogin.php?errno=100&severity=4
7. Service Registry 3控制台：在 Web 浏览器中键入以下 URL：http://hostname:port/soar/例如：http://localhost:6060/soar/如果系统中安装了 Registry，则 hostname 为 localhost。如果系统中尚未安装 Registry，请使用安装了 Registry 的系统的名称。port 的值通常为 6060，除非发生端口冲突。
8. Tomcat控制台URL：http://www.exmaple.com/manager/html
9. Tomcat控制台默认帐号admin，默认密码admin或空
10. Jboss控制台URL：http://www.exmaple.com/jmx-console/
11. Jboss控制台URL：http://www.exmaple.com/web-console/
12. Jboss控制台默认无须登陆，或者admin/admin
13. WebSphere控制台URL：http://www.exmaple.com/ibm/console/logon.jsp
14. WebSphere默认帐号admin，默认密码admin
15. Apache控制台URL：http://www.exmaple.com/server-status
16. Axis2控制台URL：http://www.exmaple.com/axis2-admin/
17. Axis2控制台默认口令帐户：admin/axis2
18. iSAP控制台URL：http://www.exmaple.com/admin/login.jsp
19. iSAP控制台默认的帐号和密码：admin/admin
20. “普元”管理控制台URL：http://www.exmaple.com/eosmgr/
21. “普元”管理控制台默认的帐号和密码：sysadmin/000000

**风险分析：**攻击者使用弱口令扫描工具或者直接使用常见的弱口令去尝试登录Web中间件的管理控制后台，然后通过部署war包上传webshell，进而控制整个系统。

**风险等级：**

【**高危**】：可访问默认中间件控制台，且能过成功获取 shell。

【**中危**】：可访问默认中间件控制台，并成功登录，但无法获取 shell。

【**低危**】：可访问默认中间件控制台，但无法登录且无法执行危险操作。

**修复方案：**默认的web容器控制台泄漏于网络中，常常可被利用，进行对web系统的攻击，一旦进入这些控制台后，可对网站进行任意的部署，中断服务等危险行为，建议从以下几点出发，修复有关控制台地址泄漏的问题：

1. 对于必须暴露于公网或者其他网络中的控制台地址，则为其地址做访问白名单措施，即只允许白名单以内的用户IP地址可以访问到该控制台，通过过滤器（filter）实现：
2. 创建客户端IP过滤器ClientIpFilter

|  |
| --- |
| package com.huawei.filters;  import java.io.IOException;  import javax.servlet.FilterChain;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.Filter;  import javax.servlet.ServletException;  import javax.servlet.FilterConfig;  public class ClientIpFilter implements Filter  {  protected FilterConfig filterConfig;  private String whiteIPlist[] = null;  public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)  throws IOException, ServletException  {  String clientIP = req.getRemoteAddr();  if(IsIPinWhitelist(clientIP))  {  //验证成功，继续处理  //System.out.println(clientIP+"调用Web Service，IP鉴权通过");  chain.doFilter(req,res);  }  else  {  //验证不成功，禁止用户调用该服务。  System.out.println(clientIP+"调用Web Service，IP鉴权不通过");  //throw new ServletException(clientIP + "无权限访问Web Service！");  }  }  private boolean IsIPinWhitelist(String remoteIPAddress)  {  for(int i = 0; i < this.whiteIPlist.length; i++)  {  String strIP = this.whiteIPlist[i];  if(strIP.equals(remoteIPAddress))  {  return true;  }  }  return false;  }  public void setFilterConfig(FilterConfig filterConfig)  {  this.filterConfig = filterConfig;  }  public void destroy()  {  filterConfig = null;  }  public void init(FilterConfig config)  throws ServletException  {  this.filterConfig = config;  String IPs = config.getInitParameter("IPList");  this.whiteIPlist = IPs.split(",");  }  } |

1. 在web.xml中部署过滤器ClientIpFilter

|  |
| --- |
| 在对应的web.xml文件的</web-app>前增加以下内容，其中蓝色部分为需要根据实际情况配置的IP地址白名单（也就是允许访问相应Web模块的客户端IP地址列表），IP地址间通过逗号隔开，不允许有空格；其中红色部分为需要根据实际情况配置的Web目录。参考代码：  <filter>  <filter-name>ClientIpFilter</filter-name>  <filter-class>com.huawei.filters.ClientIpFilter</filter-class>  <init-param>  <param-name>IPList</param-name>  <param-value>192.168.1.11,192.168.2.22,10.70.107.119</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>ClientIpFilter</filter-name>  <url-pattern>/WebService/\*</url-pattern>  </filter-mapping> |

1. 修改控制台默认的用户名和名吗，并为其控制台设置强壮的口令措施，防止可被恶意或简单猜解得到用户名和密码。
2. 修改web容器控制台的默认端口，默认路径，避免可被直接利用，访问得到地址。

例如：tomcat的修复方式：禁止别人访问tomcat主页，环境：tomcat 6.0 ；步骤1：在server.xml中 <Host name="localhost" appBase="webapps"修改改webapps为xxxxxx（自己设置的路径）；步骤2：在tomcat-users.xml中 ：<user name="admin" password="xxxx" roles="admin,manager" /> 更改用户名和密码(需强壮)。

**注意事项：**暂无

* + 1. 参数溢出

**漏洞描述：**攻击者在参数中输入超长字符串，导致数据溢出，致使应用或者数据库报错引发相关信息泄露，或者引起拒绝服务攻击等问题。

**测试方法：**在前端可控参数中输入超长字符串

**风险分析：**攻击者可通过输入参数溢出触发应用服务器异常或服务器拒绝服务，影响系统可用性。

**风险等级：**

【**高危**】：超长参数导致服务器报错引发崩溃拒绝服务。

**修复方案：**限制输入参数内容的长度。

**注意事项：**暂无

* + 1. 任意文件下载

**漏洞描述：**目录遍历（任意文件下载）漏洞不同于网站目录浏览，此漏洞不仅仅可遍历系统下web中的文件，而且可以浏览或者下载到系统中的文件，攻击人员通过目录遍历攻击可以获取系统文件及服务器的配置文件等等。一般来说，他们利用服务器API、文件标准权限进行攻击。严格来说，目录遍历攻击并不是一种web漏洞，而是网站设计人员的设计“漏洞”。

**测试方法：**

1. 通过web漏洞扫描工具对网站实施扫描可能发现目录遍历或者任意文件下载漏洞，发送一系列”../”字符来遍历高层目录，并且尝试找到系统的配置文件或者系统中存在的敏感文件。
2. 也可通过判断网站语言，并根据其url中部分提供的参数，进行构造相关的路径信息，如收集到网站中间件版本为apache，则想办法构造../../../ WEB-INF/web.xml等，然后查看其是否可被下载出来。随后可构造下载系统文件。

**风险分析：**如果web设计者设计的web内容没有恰当的访问控制，允许http遍历，攻击者就可以访问受限的目录，并可以在web根目录以外执行命令。

**风险等级：**

【**高危**】：各场景通用

**修复方案：**

1. 净化数据：对用户传过来的文件名参数进行硬编码或统一编码，对文件类型进行白名单控制，对包含恶意字符或者空字符的参数进行拒绝。
2. web应用程序可以使用chroot环境包含被访问的web目录，或者使用绝对路径+参数来访问文件目录，时使其即使越权也在访问目录之内。www目录就是一个chroot应用. 由chroot创造出的那个根目录，叫做“chroot监狱”（所谓"监狱"就是指通过chroot机制来更改某个进程所能看到的根目录，即将某进程限制在指定目录中，保证该进程只能对该目录及其子目录的文件有所动作，从而保证整个服务器的安全，详细具体chroot的用法，可参考：http://blog.csdn.net/frozen\_fish/article/details/2244870
3. 任意文件下载漏洞也有可能是web所采用的中间件的版本低而导致问题的产生，例如ibm的websphere的任意文件下载漏洞，需更新其中间件的版本可修复。
4. 要下载的文件地址保存至数据库中。
5. 文件路径保存至数据库，让用户提交文件对应ID下载文件。
6. 用户下载文件之前需要进行权限判断。
7. 文件放在web无法直接访问的目录下。
8. 不允许提供目录遍历服务。
9. 公开文件可放置在web应用程序下载目录中通过链接进行下载。

参考代码：

|  |
| --- |
| public String download() throws Exception {  //获取文件id  String id = Struts2Utils.getRequest().getParameter("id");  try {  //通过id进行文件查询  DownloadFile downFile = fileService.findEntityById(Long.parseLong(id));  // 获取该附件的类型  byte[] bt = null;  bt = downFile.getContent();  HttpServletResponse res =Struts2Utils.getResponse();  res.reset();  res.setContentType("application/x-msdownload");  res.setHeader("Content-Disposition", "attachment;filename="+ URLEncoder.encode(uacFile.getName(), "UTF-8"));  OutputStream out = res.getOutputStream();  out.write(bt);  out.flush();  out.close();  } catch (Exception e1) {  e1.printStackTrace();  }  return null;  } |

* 1. 信息猜解
     1. 邮件内容中请求链接可预测

**漏洞描述：**邮件中的重置密码等链接可预测，导致链接可以直接被猜解访问。

**测试方法：**

1. 先按照正常流程重置密码，接收重置密码邮件，分析重置链接的构造。
2. 通常情况下链接中会使用token参数使得链接具有唯一性，判断该参数是否可预测。如用户名的md5值，用户名+时间戳的md5值等。

**风险分析：**攻击者通过猜测重置密码链接可重置他人账户的密码。

**风险等级：**

【**高危**】：存在需要收件人点击确认的链接中，无安全随机数，或 token 简单可预测。

**修复方案：**重置密码链接中的token使用安全随机数

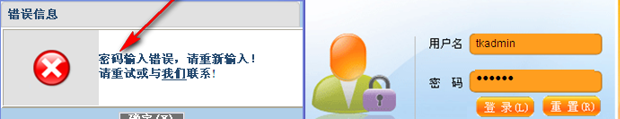
**注意事项：**暂无

* 1. 数据猜解
     1. 账号枚举

**漏洞描述：**存在于系统登录页面，利用登陆时输入系统存在的用户名错误密码和不存在的用户名错误密码，返回不同的出错信息可枚举出系统中存在的账号信息。

**测试方法：**

1. 找到网站或者web系统登录页面。
2. 在web系统登录页面，通过手工方式，利用系统中存在的用户名和不存在的用户名，密码随意，尝试登录，查看其回显内容。例如：输入存在的用户名admin，回显如下：密码错误；输入不存在的用户名test，回显如下：用户不存在。如图所示：





**风险分析：**攻击者根据Web应用程序返回的上述提示信息即可枚举系统中存在的登录用户名，然后针对枚举出来的登录用户名，对其密码进行暴力破解。

**风险等级：**

【**低危**】：可通过返回关键字对系统可用账号进行枚举。

**修复方案：**建议对网站登录页面的判断回显信息修改为一致：用户名或密码错误。

**注意事项：**暂无

* + 1. 账号密码共用

**漏洞描述：**不同的业务系统存在相同的用户名密码，或者不同用户名使用相同的初始密码。

**测试方法：**使用同一个用户名密码登录不同业务系统，看是否均可成功登录。

**风险分析：**攻击者在得到一个业务系统的用户名密码后可尝试登录其他业务系统，造成其他业务系统信息泄漏。或者使用初始密码遍历用户名，批量获取可登录系统的用户名密码。

**风险等级：**

【**高危**】：多台服务器的后台或其他服务口令相同。

【**高危**】：不同用户名使用相同初始密码，且第一次登陆未强制密码修改或强制修改机制可绕过继续使用初始密码。

【**低危**】：同样的账户名密码可以在多个系统上登录。

**修复方案：**

1. 设置每个账号的初始密码均不同，且不可预测。
2. 不同业务系统采用不同的账号密码体系。

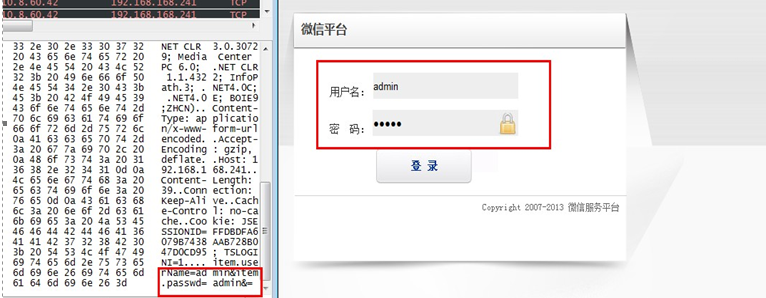
**注意事项：**暂无

* 1. 认证信息泄漏
     1. 传输过程泄漏

**漏洞描述：**认证过程中传输未加密（用户名密码等敏感数据明文传输）。

**测试方法：**

1. 找到网站或者Web系统登录页面
2. 通过对网站登录页面的请求进行抓包，工具可用burp、wireshark、filder、等等，分析其数据包中相关password（密码）参数的值是否为明文。如图利用wireshark抓包分析的密码：



**风险分析：**攻击者通过在局域网中嗅探网络流量，获取明文传输的认证凭证，如用户名密码、SESSIONID等敏感信息。

**风险等级：**

【**中危**】：传输数据包含明文密码、链接、明文身份证、明文地址等其他敏感信息。

【**中危**】：GET方式明文传输用户名密码。

【**中危**】：Token或者用户身份标识，以GET方式显示在URL中。

**修复方案：**建议按照网站的密级要求，需要对密码传输过程中进行加密得使用加密的方式传输，如使用HTTPS， 但加密的方式增加成本，或许会影响用户体验。如果不用 HTTPS，可以在网站前端用 Javascript 做密码加密，加密后再进行传输。

**注意事项：**暂无

* + 1. 会话变量泄漏

**漏洞描述：**登录、验证等页面的隐藏域中存在密码信息。

**测试方法：**查看网页源代码，寻找隐藏域中是否存在密码等信息。

**风险分析：**攻击者通过在局域网中嗅探网络流量，获取明文传输的认证凭证，如用户名密码。

**风险等级：**

【**高危**】：隐藏域中存在密码等信息

**修复方案：**禁止在前端页面中保存密码等敏感信息。

**注意事项：**暂无

* 1. 认证信息猜解
     1. 存在弱口令

**漏洞描述：**认证登录环节存在弱口令

**测试方法：**

1. 找到网站登录页面，尝试输入常见弱口令；
2. 根据网站所使用的第三方组件，寻找特定的弱口令或默认口令进行登录。

**风险分析：**攻击者可利用互联网公开的常见弱口令尝试登录管理后台，对网站造成一定的影响。

**风险等级：**

【**高危**】：存在弱口令

**修复方案：**禁止使用弱口令，口令应满足一定的复杂度。

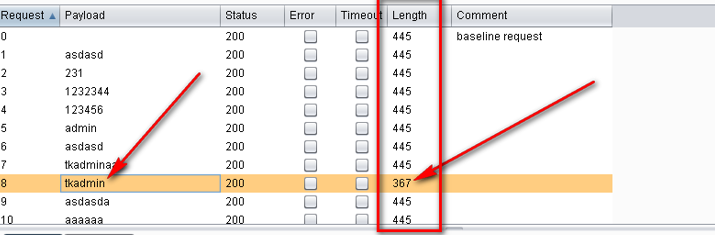
**注意事项：**暂无

* + 1. 存在暴力破解

**漏洞描述：**暴力破解的基本思想是根据题目的部分条件确定答案的大致范围，并在此范围内对所有可能的情况逐一验证，直到全部情况验证完毕。若某个情况验证符合题目的全部条件，则为本问题的一个解；若全部情况验证后都不符合题目的全部条件，则本题无解。常常存在于网站的登录系统中，通过对已知的管理员用户名，进行对其登录口令的大量尝试。

**测试方法：**

1. 找到网站登录页面。
2. 利用burp对登录页面进行抓包，将其发送到Intruder,并设置其密码参数，如pwd=为变量，添加payload（字典），进行攻击，攻击过程中查看其返回的字节长度，来判断是否成功。攻击效果如图所示：



一般情况下，暴力破解有三种形式：

1) 固定账号对密码暴力破解。

2) 在得知账号具有规律性，或者通过某种方式获取到大量账号的前提下，固定密码对账号暴力破解。

3) 使用网上流传的账号密码库进行撞库攻击。

**风险分析：**攻击者一般会使用自动化脚本组合出常见的用户名和密码，即字典，再结合软件burpsuite的intruder功能进行暴力破解。

**风险等级：**

【**中**】：存在暴力破解风险，但未暴破出密码。

**修复方案：**防止暴力攻击的一些方法如下：

1. 账户锁定

账户锁定是很有效的方法，因为暴力破解程序在5-6次的探测中猜出密码的可能性很小。但是同时也拒绝了正常用户的使用。如果攻击者的探测是建立在用户名探测成功之后的行为，那么会造成严重的拒绝服务攻击。对于对大量用户名只用一个密码的探测攻击账户锁定无效。如果对已经锁定的账户并不返回任何信息，可能迷惑攻击者。

1. 返回信息

如果不管结果如何都返回成功的信息，破解软件就会停止攻击。但是对人来说很快就会被识破。

1. 页面跳转

产生登录错的的时候就跳到另一个页面要求重新登录。比如126和校内网都是这样做的。局限性在于不能总是跳转页面，一般只在第一次错误的时候跳转，但是第一次之后又可以继续暴力探测了。

1. 适当的延时

检查密码的时候适当的插入一些暂停，可以减缓攻击，但是可能对用户造成一定的影响。

1. 封锁多次登录的IP地址

这种方法也是有缺点的，因为攻击者可以定时更换自己的IP。

1. 验证码

验证码是阻止暴力攻击的好方法，但设计不好的验证码是可以绕过的，而且对于特定目标的手工探测来说验证码是没有作用的。

**注意事项：**暂无

* 1. 认证功能失效
     1. 存在空口令

**漏洞描述：**认证登录环节允许空口令

**测试方法：**

找到网站登录页面，尝试输入用用户名，密码为空进行登录。

**风险分析：**攻击者可利用该漏洞登录网站后台，操作敏感数据，甚至上传webshell，从而控制服务器。

**风险等级：**

【**高危**】：存在空口令

**修复方案：**判断输入密码是否为空，禁止空口令登录。

**注意事项：**暂无

* + 1. 认证绕过

**漏洞描述：**能够绕过应用认证，直接登录系统。

**测试方法：**绕过认证主要有几下几种途径：

1. 网络嗅探。通过网络嗅探工具探测局域网中传输的明文用户名和密码。有些应用程序采用GET方式发送登录请求，可能导致GET的URL请求内容被缓存在代理服务器活着Web服务器端，导致用户名和密码泄漏。
2. 默认或可猜测的用户账号。大多数开源软件或商业软件提供的基于网络配置和管理的接口，通常都会有一些默认的用户名和密码。例如，一般默认的用户名是：admin，administrator、root、system、guest等，而默认的秘密吗也根据硬件和软件的不同而不同，可尝试一下这些密码：password、admin、guest、12345等。
3. 直接访问内部URL。使用Spider工具找到含有admin、manager、administrator、login等词语的路径，尝试使用普通的登录用户访问这些URL。从而获得管理员的权限。
4. 修改参数绕过认证。应用程序可能会会使用一个参数或一个隐藏的域表示一个用户是否经过验证了，通过修改这些参数，从而被认为是已经认证过的用户。例如：http://www.xxx.xom/userinfo.jsp?authenticated=no，通过修改authenticated参数为yes，http://www.xxx.xom/userinfo.jsp?authenticated=yes，然后就可以通过认证，直接访问内部页面。
5. 可猜测的SessionID。利用规律，猜测到一个有效的SessionID，然后通过修改请求中的SessionID为一个预测到的有效的SessionID，从而冒充会话的真正拥有着，绕过认证环节。
6. 注入问题。利用万能密码登录系统，绕过认证环节。
7. CSRF。利用CSRF漏洞在用户不知情的情况下，利用用户的会话进行敏感操作，从而绕过认证。

**风险分析：**如果应用程序在认证上没有做好，可以导致恶意用户或者攻击者绕过认证，访问内部资源，这类漏洞通过防火墙和入侵检测系统很难预防。

**风险等级：**

【**高**】：能够登录进入应用系统，且可以进行相关操作

【**中**】：能够登录访问进入应用系统，但无法进行相关操作

**修复方案：**可从以下几个方面预防认证绕过：

1. 对于每一个访问的URL都首先检查是否已经登录（不需要认证的URL除外，例如，帮助页面、免费下载页面等），如果没有登录，则跳转到登录页面。对于已经登录的用户，在退出的时候或者在会话很长时间处于idle状态的时候，需要保证原来的会话被正确的销毁并且不会再被重利用。
2. 规定密码强度要求，防止密码被猜测到。
3. 对于用户是否已经认证，禁止依赖客户端传过来的参数标识，而应将是否登录的标识保存在服务器端的会话中，当接收到该会话的请求时，从会话保存的状态判断是否登录。
4. 对于SessionID一定要使用安全的随机数生成算法，使得SessionID不可预测。
5. 对于暴力破解攻击，建议在尝试3次左右失败之后，使用图形验证码。

**注意事项：**暂无

* + 1. Oauth认证缺陷

**漏洞描述：**OAuth是一个在不提供用户名和密码的情况下，授权第三方应用访问Web资源的安全协议。Oauth认证不完全，可越权登录他人账户。

**测试方法：**OAuth认证缺陷具体测试场景如下：

1. 开始认证过程，从提供商那里得到回调，获取code，但暂不访问带有code的URL，如：http://www.xxx.com/connect/facebook/?code=AQCOtAVov1Cu316rpqPfs-8nDb-jJEiF7aex9n05e2dq3oiXlDwubVoC8VEGNq10rSkyyFb3wKbtZh6xpgG59FsAMMSjIAr613Ly1usZ47jPqADzbDyVuotFaRiQux3g6Ut84nmAf9j-KEvsX0bEPH\_aCekLNJ1QAnjpls0SL9ZSK-yw1wPQWQsBhbfMPNJ\_LqI
2. 然后把它放在<img src=”URL”>或<iframe>标签下保存起来。

http://image.3001.net/2012/10/13.png

http://image.3001.net/2012/10/Screenshot-2.png

http://image.3001.net/2012/10/Screenshot-3.png

1. 让用户(某一特定的用户或者target.com上随机的用户)发送HTTP请求到你的callback  
   URL。例如，通过钓鱼等手段诱骗用户访问example.com/somepage.html，其中包含<iframe src=URL>，且用户在发送HTTP请求时处于登录状态。
2. 现在，按下“用facebook账号登录”即可以登录用户账户。

**风险分析：**攻击者结合OAuth认证缺陷和钓鱼攻击可定向的登录某个用户的账户。

**风险等级：**

【**高危**】：可导致用户资源任意访问或任意账户登陆或用户密码获取。

**修复方案：**各应用除了验证access token之外，还必须辅助其他参数进行判断（比如自行加入其它认证参数进行双重认证）；另一种方法则是验证access token背后所属的应用app key的唯一性和对应性（无论是自行验证还是开放平台通过签名等形式帮助验证），确保该access token是该应用的。

**注意事项：**暂无

* + 1. IP地址伪造

**漏洞描述：**通过伪造IP地址能够绕过应用IP地址限制，访问和执行系统相关功能。

**测试方法：**使用代理软件拦截请求包，修改HTTP头中的Host字段，伪造IP地址绕过限制。

**风险分析：**攻击者可利用该漏洞访问受限系统，造成应用系统数据泄漏。

**风险等级：**

【**高危**】：访问重要系统/执行重要功能

【**中危**】：访问非重要系统/执行非重要的功能

**修复方案：**

1. 使用getServerName()代替getHeader(“Host”)；
2. 在Apache和Nginx里可以通过设置一个虚拟机来记录所有的非法Host header，或者在Apache和Nginx里指定一个ServerName名单；同时，Apache开启UseCanonicalName选项。

**注意事项：**暂无

* 1. 认证功能滥用
     1. 多点认证缺陷

**漏洞描述：**系统允许多点认证

**测试方法：**

1. 在浏览器A中使用测试账号登录系统；
2. 同时在浏览器B中使用同一个账号登录系统；
3. 若在多个浏览器中均可登录同一各账号，说明存在多点认证缺陷。

**风险分析：**攻击者在获取到其他用户的账号密码后，可利用该缺陷在用户已登录且未知的情况下进行登录，操作用户账户。

**风险等级：**

【**高危**】：多点登录架构可被绕过

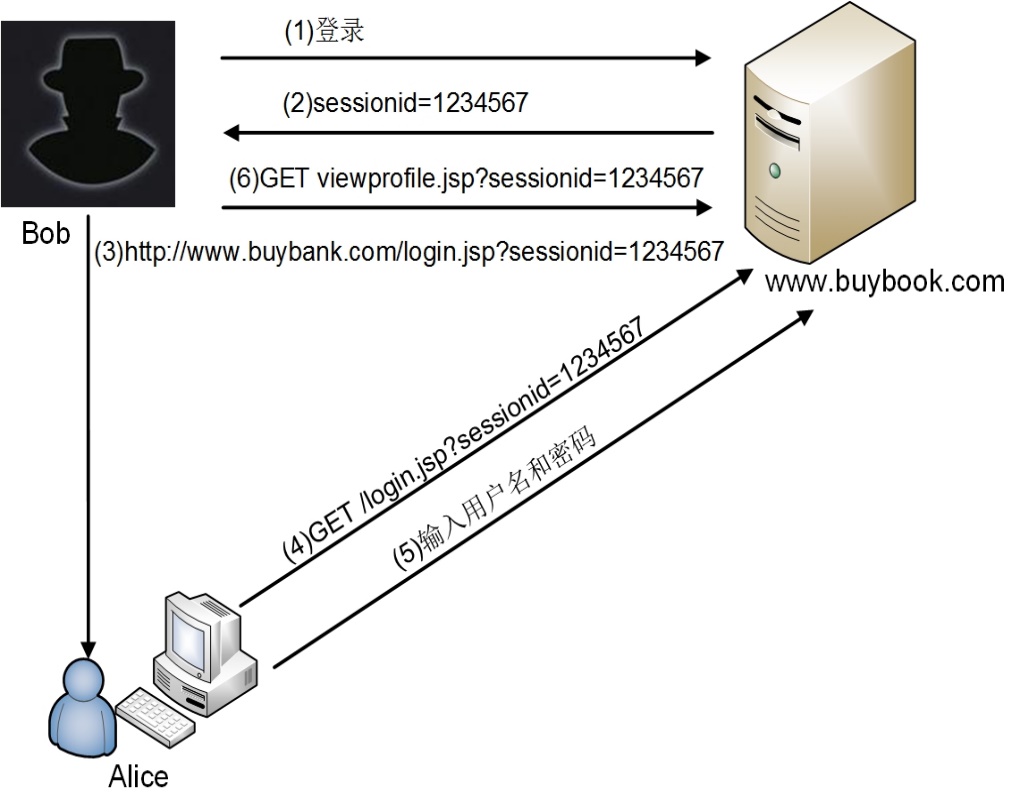
【**中危**】：核心系统允许多点登录

**修复方案：**建议在不影响业务的前提下，关键业务系统应禁止多点认证。当同一账号在其他地方登录时已登录的账号应退出会话，并提示用户账户在其他地区登录，可能存在账号被盗风险。

**注意事项：**暂无

* + 1. 会话固定

**漏洞描述：**在用户进入登录页面，但还未登录时，就已经产生了一个session，用户输入信息，登录以后，session的id不会改变，也就是说没有建立新session，原来的session也没有被销毁）。攻击者事先访问系统并建立一个会话，诱使受害者使用此会话登录系统，然后攻击者再使用该会话访问系统即可登录受害者的账户。会话固定攻击的原理及流程如下图所示：



1. 攻击者Bob匿名访问www.buybook.com。
2. 服务器与Bob建立了一个会话，比如sessionid为1234567。
3. Bob构造了一个URL：http://www.buybook.com/login.jsp?sessionid=1234567，发给了受害者Alice。
4. Alice直接打开此链接，输入自己的用户名和密码登录系统。
5. 此时Bob再次访问http://www.buybook.com/viewprofile.jsp?sessionid=1234567，即可进入Alice的账户。

**测试方法：**

1、 打开网站登录页面。

2、 登陆前通过软件工具抓取到的cookie信息值与在登录后抓取到的cookie进行对比，如果其值一样，则可判断其会话的cookies或者sessions未进行更新。

**风险分析：**攻击者可能会窃取或操纵客户会话和cookie，用于模仿合法用户，从而以该用户身份查看或变更用户记录以及执行事务。

**风险等级：**

【**中危**】：会话可由攻击者建立后发给受害者使用该会话登录系统，然后攻击者利用该会话即可登录受害者账户。

**修复方案：**在用户提供的认证信息（例如用户名和密码）、相应的权限级别发生变化时，服务器端应重新生成SessionID，并强制失效之前的会话，JAVA示例代码如下：

|  |
| --- |
| request.getSession().invalidate();//清空session  Cookie cookie = request.getCookies()[0];//获取cookie  cookie.setMaxAge(0);//让cookie过期 ； |

注意：这段代码需要在页面的最后部分加上才可以，否则将报错。

**注意事项：**暂无

* 1. 业务逻辑篡改
     1. 密码修改/重置流程跨越

**漏洞描述：**密码修改功能常采用分步骤方式来实现，攻击者在未知原始密码的情况下绕过某些检验步骤修改用户密码。

**测试方法：**

1. 完成修改／重置密码的正常流程；
2. 绕过检验原密码等步骤，直接访问输入新密码页面，输入新密码，修改／重置密码。

**风险分析：**有些密码修改／重置流程采用step=1、step=2类似的方式实现，如果应用校验不全面，攻击者可绕过前面的步骤，直接访问最后一步，输入新密码进行修改／重置。

**风险等级：**

【**高危**】：绕过原密码验证或绕过验证码

**修复方案：**一次性填写校验信息（原始密码、新密码等）后再提交修改／重置密码请求。

**注意事项：**暂无

* + 1. 负值反冲

**漏洞描述：**应用程序未校验订单数据的取值范围，交易存在负值反冲。

**测试方法：**提交订单时拦截请求，修改订单参数为负数，如商品单价、数量、总价等。

**风险分析：**通过篡改订单参数，使得订单金额为负值，在使用余额支付时余额反而增加。

**风险等级：**

【**高危**】：未对数据进行校验，导致业务数据被污染。

**修复方案：**

1. 服务器端在生成交易订单时，商品的价格从数据库中取出，禁止使用客户端发送的商品价格。
2. 服务器端对客户端提交的交易数据（如商品ID、商品数量、商品价格等）的取值范围进行校验，将商品ID和商品价格与数据库中的数据对比校验，商品数量为大于零的整型数。
3. 服务器端在生成支付订单时，对支付订单中影响支付金额的所有因素（比如商品ID、商品数量、商品价格、订单编号等）进行签名，对客户端提交的支付订单进行校验。

**注意事项：**暂无

* + 1. 正负值对冲

**漏洞描述：**应用程序未校验订单数据的取值范围，交易存在正负值对冲。

**测试方法：**提交订单（包含多种商品）时拦截请求，修改部分商品的单价或数量，保证订单总金额为正数。

**风险分析：**由于应用会校验订单总金额的取值范围，所以在保证该条件满足的前提下，修改个别商品的数量，达到正负值对冲。

**风险等级：**

【**高危**】：未对数据进行校验，导致业务数据被污染。

**修复方案：**

1. 服务器端在生成交易订单时，商品的价格从数据库中取出，禁止使用客户端发送的商品价格。
2. 服务器端对客户端提交的交易数据（如商品ID、商品数量、商品价格等）的取值范围进行校验，将商品ID和商品价格与数据库中的数据对比校验，商品数量为大于零的整型数。
3. 服务器端在生成支付订单时，对支付订单中影响支付金额的所有因素（比如商品ID、商品数量、商品价格、订单编号等）进行签名，对客户端提交的支付订单进行校验。

**注意事项：**暂无

* + 1. 业务流程跳跃

**漏洞描述：**业务逻辑流程分步骤进行且能越过中间校验步骤直接进行后续操作，导致中间校验等步骤失效。

**测试方法：**

1. 首先完成正常的业务逻辑步骤，获取每一个步骤的请求；
2. 绕过中间步骤，直接访问最后一个或几个验证请求，看是否可绕过。

**风险分析：**攻击者可利用该漏洞绕过业务流程检测，进行非法修改他人密码等危险操作。

**风险等级：**

【**高危**】：绕过前面的校验步骤，直接跳转至后面的校验步骤。

**修复方案：**建议在不影响业务的前提下，在Session中添加对每一步流程页面的校验标志位，在新步骤页面浏览过程中，仅能够顺序执行页面校验，不可进行跳步操作，例如：页面二完成后，应更新Flag=3，则仅能够打开页面三。

**注意事项：**暂无

* 1. 业务功能失效
     1. 通配符注入

**漏洞描述：**允许使用通配符构造语句查询数据库，导致拒绝服务攻击问题。

**测试方法：**模糊查询时输入第一个字符'%'或'\_'，sql会遍历全表，导致应用访问缓慢。

**风险分析：**SQL中通配符的使用如下：

%包含零个或多个字符的任意字符串。

\_任何单个字符。

[]指定范围（例如 [a-f]）或集合（例如 [abcdef]）内的任何单个字符。

[^]不在指定范围（例如 [^a - f]）或集合（例如 [^abcdef]）内的任何单个字符。

在模糊查询LIKE中，对于输入数据中的通配符必须转义，否则会造成客户想查询包含这些特殊字符的数据时，这些特殊字符却被解析为通配符，数据库响应缓慢，导致拒绝服务攻击。

**风险等级：**

【**中危**】：通配符注入引发数据库响应缓慢

**修复方案：**有两种将通配符转义为普通字符的方法：

1) 使用ESCAPE关键字定义转义符（通用）

在模式中，当转义符置于通配符之前时，该通配符就解释为普通字符。例如，要搜索在任意位置包含字符串 5% 的字符串，请使用：

WHERE ColumnA LIKE '%5/%%' ESCAPE '/'

2) 在方括号 ([ ]) 中只包含通配符本身，或要搜索破折号 (-) 而不是用它指定搜索范围，请将破折号指定为方括号内的第一个字符。例如：

|  |  |
| --- | --- |
| **符号** | **含义** |
| LIKE '5[%]' | 5% |
| LIKE '5%' | 5 后跟 0 个或多个字符的字符串 |
| LIKE '[\_]n' | \_n |
| LIKE '[ [ ]' | [ |
| LIKE ']' | ] （右括号不需要转义） |

所以，对输入参数的关键字过滤后，还需要做下面转换确保LIKE的正确执行

|  |
| --- |
| private static string ConvertSqlForLike(string sql)  {  sql = sql.Replace("[",  "[[]");  // 这句话一定要在下面两个语句之前，否则作为转义符的方括号会被当作数据被再次处理  sql = sql.Replace("\_",  "[\_]");  sql = sql.Replace("%",  "[%]");  returnsql;  } |

**注意事项：**暂无

* 1. 业务功能滥用
     1. 短信定向转发

**漏洞描述：**短信接收人可任意指定

**测试方法：**拦截发送短信的请求，将手机号改为测试人员的手机号，测试是否可接收短信验证码。

**风险分析：**攻击者可利用该漏洞将验证码发送到自己的手机号上，重置他人密码或转账。

**风险等级：**

【**高危**】：短信接收人可任意指定

**修复方案：**发送短信时手机号从当前会话中获取，避免从前端传入。

**注意事项：**暂无

* + 1. 邮件可定向转发

**漏洞描述：**应用系统发送邮件的接收人可由客户端任意指定

**测试方法：**拦截发送邮件的请求，将接收人邮箱改为测试人员的邮箱地址，测试是否可接收邮件。

**风险分析：**攻击者可利用该漏洞将邮件发送到自己的邮箱中，重置他人密码。

**风险等级：**

【**高危**】：邮件接收人可任意指定

**修复方案：**发送邮件时邮箱从当前会话中获取，避免从前端传入。

**注意事项：**暂无

* + 1. 业务接口调用缺陷

**漏洞描述：**应用的业务接口存在缺陷，可以通过业务接口直接进行恶意操作。

**测试方法：**把业务逻辑和功能模块呈现的内容结合，推测出隐藏的API接口。如用户信息的接口是http://www.xxx.com/api/user/userInfo，推测重置密码接口可能是http://www.xxx.com/api/user/passReset，文件上传接口是http://www.xxx.com/api/user/uploadFile等。如果这些接口没有限制访问，则可以直接调用并提交数据。

针对开源或商业软件的业务接口调用缺陷可参考《通用系统安全测试指导文档》

**风险分析：**攻击者可通过编写API枚举脚本，恶意调用敏感接口，从而进行提交数据等操作。

**风险等级：**

【**高危**】：通过业务接口能够操作核心业务内容，进行越权

【**高危**】：通过业务接口能够获得重要敏感数据

【**中危**】：通过业务接口能够获得中等程度敏感数据

**修复方案：**对于每一个访问的接口都首先检查当前用户是否已经登录并授权（不需要认证的接口除外，例如，免费下载接口等）。

**注意事项：**暂无

* + 1. IMAP/SMTP注入

**漏洞描述：**和广为人知的诸如SQL注入、XPath注入等技术类似，邮件服务器注入技术也是通过一个对用户提供的数据没有严格检查的webmail应用程序将IMAP命令或者SMTP命令注入到邮件服务器。要向邮件服务器注入命令，前提条件是允许用户通过webmail应用程序访问其端口25（SMTP）和143（IMAP）。

**测试方法：**要利用SMTP注射，用户必须事先通过认证，所以用户必须有一个有效的webmail帐户。通过SMTP发送电子邮件消息的格式如下：

* 发送方的e-mail地址
* 接收方的e-mail地址
* 主题
* 消息主体
* 附件

1. CC/BCC注入  
   在发送者字段(sender)后注入CC和BCC参数  
   From:sender@domain.com%0ACc:recipient@domain.com%0ABcc:recipient1@domain.com  
   所以现在,消息将被发送到recipient和recipient1账户。
2. 参数注射  
   From:sender@domain.com%0ATo:attacker@domain.com  
   现在消息将被发送到原来的收件人和攻击者帐户。注意，这里的攻击者的账户是我们通过注入额外传入的。
3. 邮件主题注入  
   From:sender@domain.com%0ASubject:This’s%20Fake%20Subject  
   攻击者注入的假的主题subject将被添加到原来的主题中并且在某些情况下将取代原本的主题subject。这取决于邮件服务行为。即代码编写的容错性，当参数中出现两个subject的时候代码是选择丢弃还是后者覆盖。
4. 改变消息的主体body  
   要注意SMTP的Mail格式，消息主题和头部Header之间有两个换行符(和HTTP是一样的)。  
   From:sender@domain.com%0A%0AMy%20New%20%0Fake%20Message.  
   假消息将被添加到原始消息中。

**风险分析：**电子邮件注入允许恶意攻击者注入任意邮件头字段,BCC、CC、主题等,它允许黑客通过注入手段从受害者的邮件服务器发送垃圾邮件。

**风险等级：**

【**高危**】：可成功劫持密码找回等信息

【**高危**】：可成功发送垃圾邮件

**修复方案：**建议从以下几个方面进行防御：

1. 所有用户输入应该被认为是不可信的。使用正则表达式来过滤用用户提交的数据。例如，可以在输入字符串中搜索(r 或 n)。
2. 使用外部组件和库，例如ZEND mail、PEAR mail和swift mailer。
3. ModSecurity可以阻止服务器级别的电子邮件注入。利用ModSecurity，可以检测通过POST或GET提交的CC, BCC或目的地址，并且拒绝任何包含这些字母请求。

**注意事项：**暂无

* + 1. 引用第三方不可控脚本/URL

**漏洞描述：**页面中引用了不可控的脚本或超链接

**测试方法：**检查页面内容，是否引用了第三方未知脚本或超链接，如恶意的js脚本或病毒木马的下载链接等。

**风险分析：**攻击者可在网站中插入恶意链接或脚本，导致正常用户浏览时cookie被窃取或被种植病毒木马。

**风险等级：**

【**高危**】：存在不可控外链或脚本，且未经过审批

【**中危**】：存在不可控外链或脚本，但可提供审批记录

**修复方案：**建议在不影响业务的前提下，对网站引用的文件和源代码进行审查，一经发现有未审批的外链或脚本，应立即删除。

**注意事项：**暂无

* + 1. 开启危险接口

**漏洞描述：**开启可利用的危险接口，如获取订单信息、用户身份信息等。

**测试方法：**

1. 使用扫描器扫描特殊目录和链接
2. 根据正常接口的命名特征猜测隐藏的危险接口，如获取个人信息接口是getUserInfo，猜测获取订单信息接口getOrderDetail。

**风险分析：**开启了危险接口，如订单信息打印、上传、web管理控制台等，可能被攻击者发现并利用，直接操作应用数据，造成数据泄漏等风险。

**风险等级：**

【**高危**】：正常情况接口是在认证之后被调用，如果可公网直接无认证使用

【**中危**】：存在特权、非正常用户不可知但可利用接口

**修复方案：**

1. 敏感接口增加访问控制，避免未授权访问；
2. 用户访问接口需校验权限，避免越权访问。

**注意事项：**暂无

* + 1. 未验证的URL跳转

**漏洞描述：**服务端未对传入的跳转url变量进行检查和控制，可能导致可恶意构造任意一个恶意地址，诱导用户跳转到恶意网站。由于是从可信的站点跳转出去的，用户会比较信任，所以跳转漏洞一般用于钓鱼攻击，通过转到恶意网站欺骗用户输入用户名和密码盗取用户信息，或欺骗用户进行金钱交易；也可能引发的XSS漏洞（主要是跳转常常使用302跳转，即设置HTTP响应头，Locatioin: url，如果url包含了CRLF，则可能隔断了http响应头，使得后面部分落到了http body，从而导致xss漏洞）。另外在struts2 中存在重定向的漏洞，是因为struts2由于缩写的导航和重定向前缀“action:”、 “redirect:”、 “redirectAction:” 等参数前缀的内容没有被正确过滤导致的开放式重定向漏洞。

**测试方法：**

1. 首先找到网站相关url中存在跳转链接的参数（如登陆页面）。
2. 在检测的同时，可以修改参数中的合法URL为非法URL，然后查看是否能正常跳转或者通过抓包工具获取其HTTP响应头中Host:的值是否包含了构造的URL。
3. 如果是struts2重定向漏洞，则可通过web扫描工具扫描发现，或者手工验证，直接在URL后添加?redirect:+指定钓鱼链接，例如：10.1.82.53:9098/admin/login.action?redirect:http://diaoyu.com进行验证。

**风险分析：**攻击者利用URL跳转漏洞来欺骗安全意识低的用户，从而导致“中奖”之类的欺诈事件发生。同时利用URL跳转，也可以突破常见的基于“白名单方式”的一些安全限制，如传统IM里对于URL的传输会进行安全校验，但是对于知名网站的域名及URL将直接允许通过并且显示为可信的URL，一旦该可信的URL里包含URL跳转漏洞将导致安全限制被绕过。URL跳转最典型的例子就是登录跳转，示例代码如下：

|  |
| --- |
| public void doRedirect(HttpServletRequest req, HttpServletResponse res)  {  String jumpURL=request.getParameter(“jumptoURL”);  response.setHeader("Location",jumpURL);  } |

若程序未过滤jumptoURL参数，攻击者将恶意链接：http://www.xxx.com/login.jsp?jumptoURL=http://www.evil.com发给其他用户，安全意识较低的用户会认为该链接展现的内容是www.xxx.com，从而产生欺诈行为。同时由于QQ、淘宝旺旺等在线IM都是基于URL的过滤，对部分站点会通过白名单的方式直接通过，因此恶意URL可以在IM里传播。例如IM认为www.xxx.com是可信的，但是通过在IM里点击上述链接将导致用户最终访问http://www.evil.com。

**风险等级：**

【**高危**】：URL 跳转参数可控，且未对参数做白名单限制导致任意地址跳转可被利用钓鱼。

**修复方案：**对传入的URL做有效性的认证，保证该URL来自于信任域。限制的方式可参考以下两种：

1. 限制Referer（Referer是HTTP header中的字段，当浏览器向web服务器发送请求时，一般会带上Referer，告诉服务器是从哪个页面链接过来的），通过限制Referer保证将要跳转URL的有效性，避免攻击者生成自己的恶意跳转链接；
2. 加入有效性验证Token，保证所有生成的链接都来自于可信域，通过在生成的链接里加入用户不可控的Token对生成的链接进行校验。

**注意事项：**暂无

* + 1. 服务器端请求伪造（SSRF）

**漏洞描述：**服务端请求伪造攻击（Server-side Request Forgery）: 很多web应用都提供了从其他的服务器上获取数据的功能。使用用户指定的URL，web应用可以获取图片，下载文件，读取文件内容等。这个功能如果被恶意使用，可以利用存在缺陷的web应用作为代理攻击远程和本地的服务器。这种形式的攻击称为服务端请求伪造攻击（Server-side Request Forgery）。一般情况下， SSRF攻击的目标是从外网无法访问的内部系统 。（ 正是因为它是由服务端发起的，所以它能够请求到与它相连而与外网隔离的内部系统 ）.SSRF 形成的原因大都是由于 服务端提供了从其他服务器应用获取数据的功能且没有对目标地址做过滤与限制 。比如从指定URL地址获取网页文本内容，加载指定地址的图片，下载等等。攻击者利用ssrf可以实现的攻击主要有5种:

1. 可以对外网、服务器所在内网、本地进行端口扫描，获取一些服务的banner信息;
2. 攻击运行在内网或本地的应用程序（比如溢出）;
3. 对内网web应用进行指纹识别，通过访问默认文件实现;
4. 攻击内外网的web应用，主要是使用get参数就可以实现的攻击（比如struts2，sqli等）;
5. 利用file协议读取本地文件等。

**测试方法：**从WEB功能上寻找：我们从上面的概述可以看出，SSRF是由于服务端获取其他服务器的相关信息的功能中形成的，因此我们大可以列举几种在web 应用中常见的从服务端获取其他服务器信息的的功能。

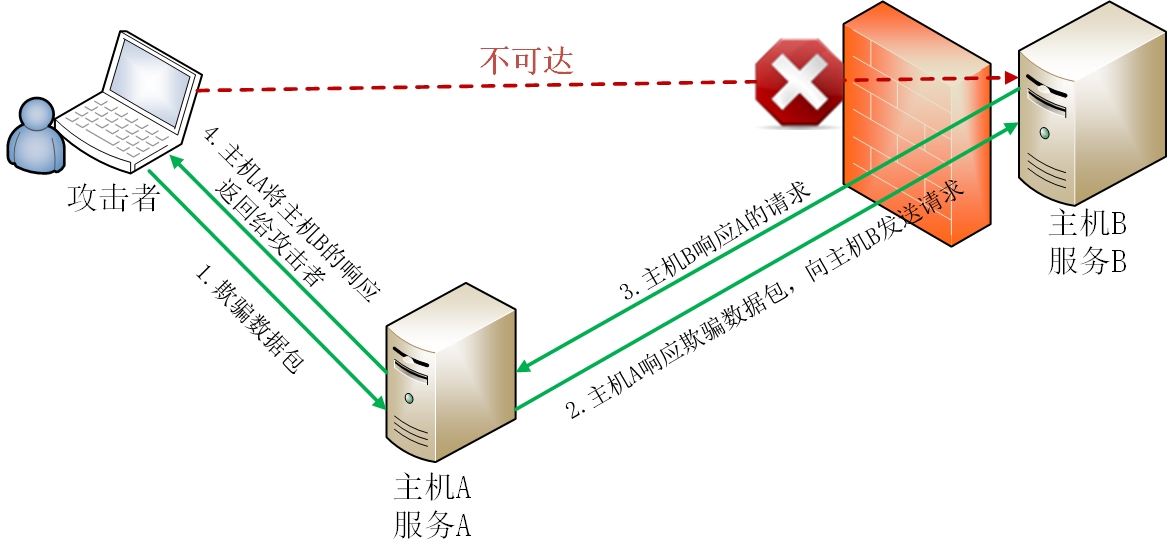
1. 通过分享功能：通过URL地址分享网页内容：

早期分享应用中，为了更好的提供用户体验，WEB应用在分享功能中，通常会获取目标URL地址网页内容中的<tilte></title>标签或者<meta name="description" content=“”/>标签中content的文本内容作为显示以提供更好的用户体验。例如人人网分享功能中：http://widget.renren.com/\*\*\*\*\*?resourceUrl=https://www.nsfocus.com

通过目标URL地址获取了title标签和相关文本内容。而如果在此功能中没有对目标地址的范围做过滤与限制则就存在着SSRF漏洞.

1. 转码服务：通过URL地址把原地址的网页内容调优使其适合手机屏幕浏览：由于手机屏幕大小的关系，直接浏览网页内容的时候会造成许多不便，因此有些公司提供了转码功能，把网页内容通过相关手段转为适合手机屏幕浏览的样式。例如百度、腾讯、搜狗等公司都有提供在线转码服务。
2. 在线翻译：通过URL地址翻译对应文本的内容。提供此功能的国内公司有百度、有道等。
3. 图片加载与下载，通过URL地址加载或下载图片：图片加载远程图片地址此功能用到的地方很多，但大多都是比较隐秘，比如在有些公司中的加载自家图片服务器上的图片用于展示。（此处可能会有人有疑问，为什么加载图片服务器上的图片也会有问题，直接使用img标签不就好了？ ，没错是这样，但是开发者为了有更好的用户体验通常对图片做些微小调整例如加水印、压缩等，所以就可能造成SSRF问题）。
4. 图片、文章收藏功能：此处的图片、文章收藏中的文章收藏就类似于功能一、分享功能中获取URL地址中title以及文本的内容作为显示，目的还是为了更好的用户体验，而图片收藏就类似于功能四、图片加载。
5. 未公开的api实现以及其他调用URL的功能：此处类似的功能有360提供的网站评分，以及有些网站通过api获取远程地址xml文件来加载内容.

**风险分析：**如果应用程序对用户提供的URL和远端服务器返回的信息没有进行合适的验证和过滤，则可能存在服务器端请求伪造攻击。服务器端请求伪造攻击场景如下图所示：

****

攻击者想要访问主机B上的服务，由于存在防火墙的原因无法直接访问，这时可以借助主机A来发起服务器端请求伪造攻击，通过主机A向主机B发起请求，从而完成攻击。SSRF攻击方式主要有以下5种：

1. 对外网、服务器所在内网、本地进行端口扫描，获取一些服务的banner信息（一般包括服务器的类型、版本，服务器上启动的服务信息）；
2. 攻击运行在内网或本地的应用程序（比如堆栈溢出）；
3. 通过访问默认文件实现对内网Web应用的指纹识别；
4. 攻击内外网的Web应用，主要是使用GET参数就可以实现的攻击（比如Struts2，Sqli等）；
5. 利用file协议读取本地文件等。

**风险等级：**

【**高危**】：有回显，可探测内网，文件访问

【**中危**】：无回显，但可访问外网

**修复方案：**通常从以下几点来防御服务器端请求伪造攻击：

1. 过滤内网服务器对公网服务器请求的响应。如果Web应用是获取某一类型的文件，在把返回结果展示给用户之前应先验证返回的信息是否符合文件类型标准，比如返回信息应为图片，如果返回信息是HTML，则停止将返回信息返回客户端。
2. 统一错误提示信息，避免用户可以根据错误信息来判断远端服务器的端口状态。
3. 在内网服务器的防火墙上限制公网服务器的请求端口为HTTP等协议常用端口，如：80、443、8080、8090。
4. 若公网服务器的内网IP与内网无业务通信，建议将公网服务器对应的内网IP列入黑名单，避免应用被用来获取内网数据。
5. 内网服务器禁用不必要的协议，仅允许HTTP和HTTPS请求，防止类似于file:///、gopher://、ftp:// 等协议引起的安全问题。

**注意事项：**暂无

* + 1. 短信内容可控

**漏洞描述：**短信内容可从客户端指定

**测试方法：**在客户端编辑任意短信内容，测试是否过滤特殊内容。

**风险分析：**攻击者可利用该漏洞，借助短信平台，编辑钓鱼短信发送给其他用户。

**风险等级：**

【**高危**】：短信内容可控，且接收人可任意指定

【**中危**】：短信内容可控，但接受人不可控

**修复方案：**建议在不影响业务的前提下，禁止客户端编辑短信内容。

**注意事项：**暂无

* + 1. 邮件内容可控

**漏洞描述：**应用系统发送邮件的邮件内容可由客户端任意指定

**测试方法：**在客户端编辑任意邮件内容，测试是否过滤特殊内容。

**风险分析：**攻击者可利用该漏洞，借助邮件平台，编辑钓鱼邮件发送给其他用户。

**风险等级：**

【**高危**】：邮件内容可控，且收件人可任意指定

【**中危**】：邮件内容可控，但收件人地址不可控

**修复方案：**建议在不影响业务的前提下，禁止客户端编辑邮件内容。

**注意事项：**暂无

* + 1. 请求重放攻击

**漏洞描述：**关键业务操作未作token或者唯一标识码，导致攻击者可以重复多次进行请求，导致恶意操作。如重复交易等问题。

**测试方法：**重放http/tcp请求，查看第二次请求是否被正常处理。

**风险分析：**攻击者恶意或欺诈性地重复发送一个有效的数据包，如果服务器未检查此类重复提交的请求数据的有效性，那么转账充值等敏感操作将有可能会被重复执行。

**风险等级：**

【**高危**】：关键业务操作请求未设置 token 或标识码，导致业务数据越界或错误

**修复方案：**服务端应用程序应检查客户端提交的数据的唯一性，如使用流水号、时间戳、token等，并将流水号、时间戳等进行签名。

**注意事项：**暂无

* + 1. 批量提交

**漏洞描述：**应用程序未使用验证码等防自动化操作的方法，可批量提交请求，如批量注册。

**测试方法：**

1. 编写自动提交HTTP数据包的脚本；
2. 或使用burpsuite的intruder功能批量提交请求。

**风险分析：**注册不需要验证码时，攻击者通过编写自动化脚本，实现程序自动提交注册信息；若注册需要验证码，但验证码位数不多于4位且为纯数字时，通过使用软件burpsuite的intruder功能穷举得到正确的验证码后，再结合自动化脚本工具即可实现批量注册垃圾账号。

**风险等级：**

【**高危**】：正常业务为单条数据请求，且不存在防自动化批量操作措施，可批量自动化提交。

【**低危**】：正常业务为单条数据请求，且存在防自动化批量操作措施，但在单个数据包中写入批量数据，可成功提交，并且服务器能批量执行。

**修复方案：**

1. 使用安全性强的验证码，验证码应从以下方面保证其安全性：验证码长度不低于4位，避免使用容易被程序自动识别的验证码，验证码不应返回客户端。
2. 用户注册功能处，提交注册信息后进行邮箱或短信验证通过后再将注册信息写入数据库。
3. 对同一IP地址短时间内提交请求的次数进行限制。

**注意事项：**暂无

* 1. 防护功能失效
     1. 账号弱锁定机制

**漏洞描述：**系统帐号锁定时间太短

**测试方法：**登录时多次输入错误密码，触发账户锁定机制，查看锁定时间是否低于3分钟。

**风险分析：**若账户锁定时间过短，攻击者仍可利用其进行暴力破解。

**风险等级：**

【**低危**】：账户在多次尝试失败后锁定时间低于 3 分钟

**修复方案：**建议在不影响业务的前提下，账户在多次尝试失败后锁定时间不低于3分钟。

**注意事项：**暂无

* + 1. 图形验证码可自动获取

**漏洞描述：**图形验证码过于简单，可使用工具自动化识别。

**测试方法：**利用Python Image Library、tesseract-ocr、pytesser等python第三方库，经过二值化、文字分割等操作识别验证码。

**风险分析：**验证码通常使用一些线条和一些不规则的字符组成，这些字符通常同时包含字母和数字。但有些Web程序设计的验证码较简单，仅由数字或字母组成，且生成的验证码字符排列规整，很容易被程序自动识别。

**风险等级：**

【**中危**】：图形验证码可被自动化识别，且成功率高于95%。

**修复方案：**使用安全性强的验证码，验证码应从以下方面保证其安全性：验证码长度不低于4位，至少同时包含数字、字母或汉字，增加干扰因素比如线条，避免使用容易被程序自动识别的验证码，验证码不应返回到客户端。

**注意事项：**暂无

* + 1. 图形验证码绕过

**漏洞描述：**图形验证码可被绕过，执行暴力破解等操作。

**测试方法：**

1. 验证码和用户名、密码是否一次性、同时提交给服务器验证，如果是分开提交、分开验证，则存在漏洞。
2. 在服务器端，是否只有在验证码检验通过后才进行用户名和密码的检验，如果不是说明存在漏洞。（检测方法：输入错误的用户名或密码、错误的验证码。观察返回信息，是否只提示验证码错误，也就是说当验证码错误时，禁止再判断用户名和密码。）
3. 验证码是否为图片形式且在一张图片中，不为图片形式或不在一张图片中，说明存在漏洞，完成检测。
4. 生成的验证码是否可以通过html源代码查看到，如果可以说明存在漏洞，完成检测。
5. 生成验证码的模块是否根据提供的参数生成验证码，如果是说明存在漏洞，完成检测。
6. 请求10次观察验证码是否随机生成，如果存在一定的规律（例如5次后出现同一验证码）说明存在漏洞，完成检测。
7. 验证码在认证一次后是否立即失效。

**风险分析：**图形验证码一般是防止使用程序恶意注册、暴力破解用户名密码或者批量发帖而设置的。在页面初始化时服务器向页面发送一个随机字符串，同时在Session里也保存一份，当用户提交时将随机数一起post到后台，通过与Session中保存的值对比，如果不相同，则有可能是恶意攻击。

**风险等级：**

【**中危**】：验证码可多次使用、发现特权验证码、前端校验、返回验证码、返回带有验证码信息的图片。

**修复方案：**

1. 系统在开发时注意验证识别后销毁session中的验证码。
2. 限制用户提交的验证码不能为空
3. 判断提交的验证码与服务器上存储的是否一致
4. 禁止将验证码明文信息发送至客户端

**注意事项：**暂无

* + 1. 短信验证码绕过

**漏洞描述：**短信验证码可被绕过，执行其他操作。

**测试方法：**

1. 请求发送短信，填写任意验证码，然后提交其他操作请求，将验证码参数置空或删除，测试是否可绕过检测；
2. 尝试特权验证码，如000000、111111等；
3. 同一个短信验证码是否能使用多次；

**风险分析：**修改／重置密码、交易操作等功能通常需要短信验证码，若验证码可绕过，攻击者可利用该漏洞进行重置他人密码或转账等危险操作。

**风险等级：**

【**中危**】：验证码可多次使用或发现特权验证码

**修复方案：**

1. 若存在特权验证码，建议将其删除；
2. 应用服务端应严格校验验证码参数是否为空，格式是否正确；
3. 关键操作每提交一次请求，应发送新的短信验证码，并且不可继续使用旧的验证码。

**注意事项：**暂无

* + 1. 短信验证码可暴力破解

**漏洞描述：**短信验证码位数太短或有效期太长导致可暴力破解

**测试方法：**点击发送短信验证码，输入任意验证码，提交请求，使用burpsuite拦截请求，在intruder模块设置验证码参数为枚举变量，这是payload类型为numbers，对验证码进行暴力破解。

**风险分析：**修改／重置密码、交易操作等功能通常需要短信验证码，若验证码可暴力破解，攻击者可利用该漏洞进行重置他人密码或转账等危险操作。

**风险等级：**

【**中危**】：手机短信验证码小于 6 位、有效期大于 1 分钟、验证错误次数无限制

**修复方案：**

1. 短信验证码不少于6位；
2. 有效期不超过1分钟；
3. 验证码错误次数超过上限应采取账户锁定策略。

**注意事项：**暂无

* + 1. 参数覆盖

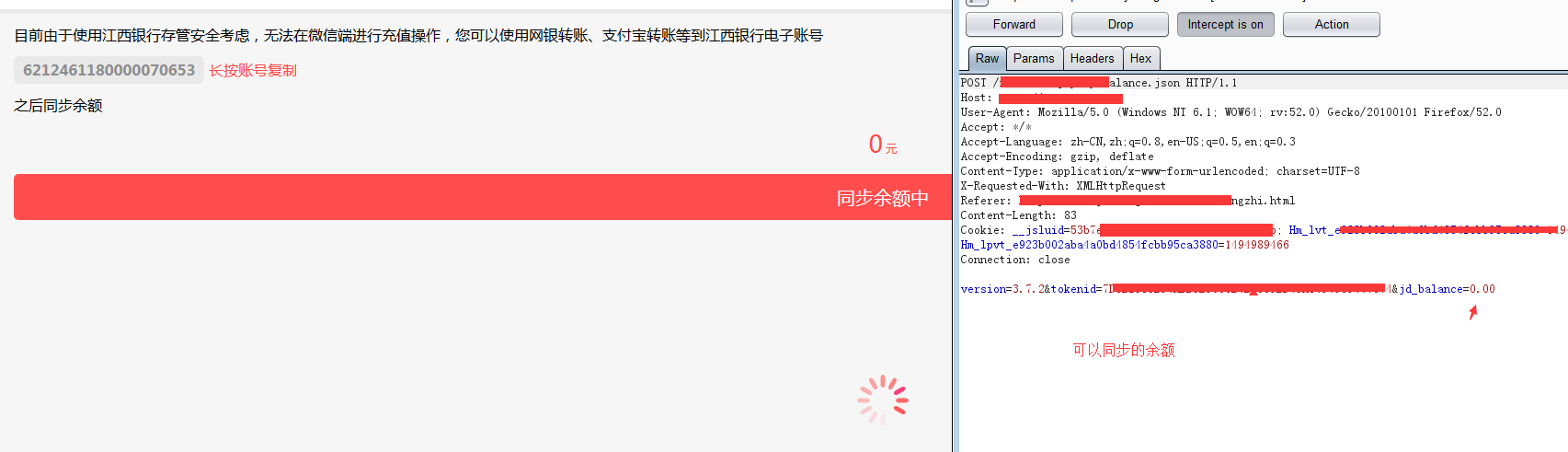
**漏洞描述：**参数覆盖，也叫对象自动绑定漏洞。许多框架均支持对象自动绑定，比如Spring MCV, 它允许将HTTP请求参数自动的绑定到对象，攻击者可以添加额外的HTTP请求参数，如果开发人员在处理业务逻辑时缺少安全校验就会导致参数覆盖问题。

**测试方法：**下面举一个具体的攻击场景：

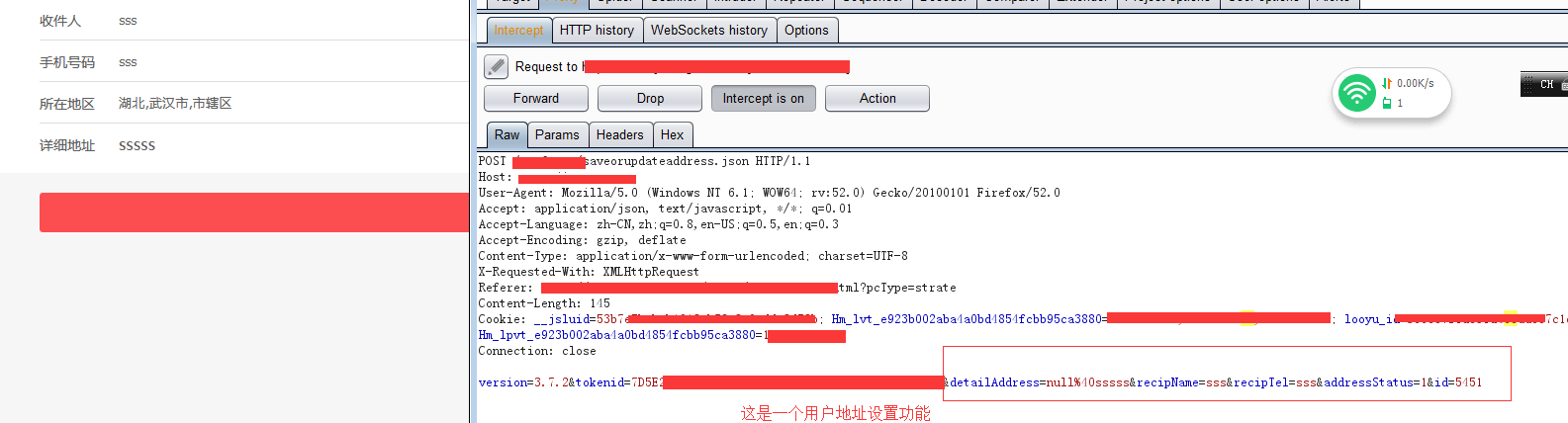
这是国内某p2p平台账户余额功能

****

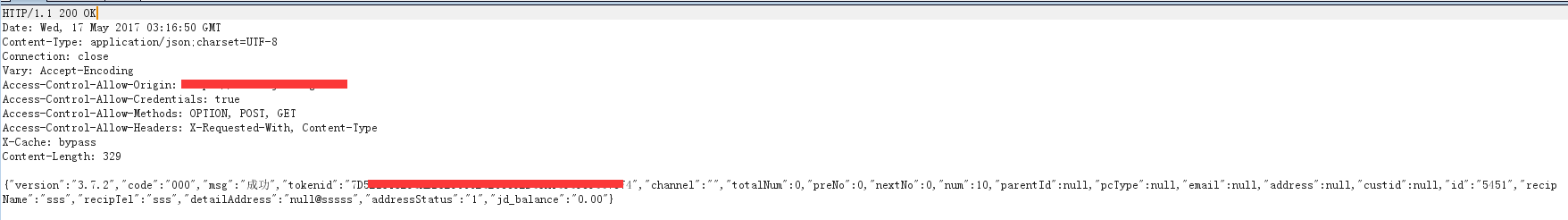
在账户余额的首页我们注册了账号进去以后发现余额是0元，需要充值才能进行投资理财然后提现。

****

在充值的页面可以同步余额，经过分析发现这个功能就是把平台上的钱和银行里账户的钱同步一致。这个p2p平台使用的是江西银行来保存客户资金所以最终同步的结果就是江西银行账号的钱和平台的钱一样。经过长期分析发现这个地方不存任何漏洞，然后继续往下看。

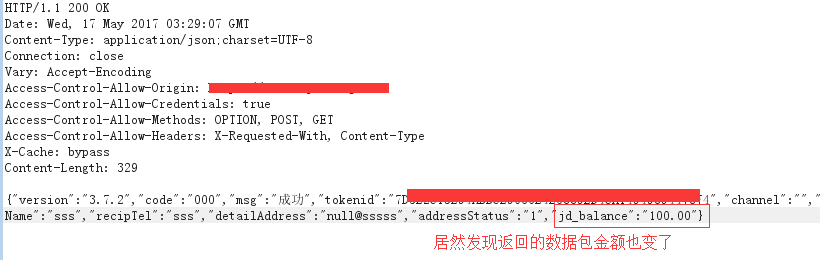
****

找到了一个用户地址的设置功能。

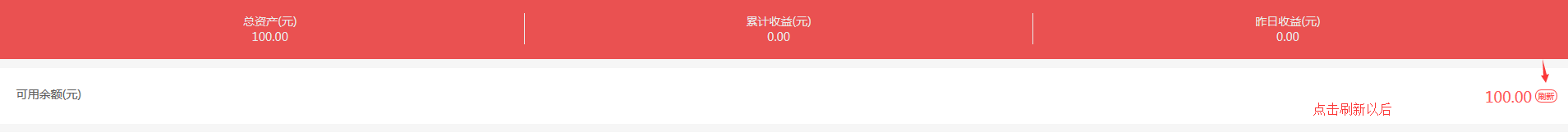
****

这是从burp response回来的数据包，通过对这个数据包的分析，发现请求的数据包和返回的数据包参数不一致。换句话说返回的数据包里面参数比请求的数据包多了几个字段其中有email字段和jdbalance字段。具体的测试操作就是在请求的数据包中把多出来的字段加入进去然后观察。

****

上面是请求的数据包，用burp截断返回数据包的方式看下返回的数据包 ****

返回的金额变成了100.00。 这个时候并不代表真的达到了数据污染的目的。然后去余额账户里面看一下。

****

点击刷新以后我们发现余额和总资产已经变成了100。后面我们重复了前面开始的操作同步平台上账户里面的钱到银行然后在通过平台成功提现100。

**风险分析：**常见的所有输入的地方都会出现这个漏洞，攻击者可利用该漏洞修改任意用户信息，篡改金额等。

**风险等级：**

**【高危】**

**修复方案：**建议从以下三个方面进行修复：

1. DTO

|  |
| --- |
| public class companyInfo {  private String xxx;  private String xxx;  //private String companyName;    //Getters & Setters  } |

1. 白名单

|  |
| --- |
| @Controller  public class companyInfoController  {  @InitBinder  public void initBinder(WebDataBinder binder, WebRequest request)  {  binder.setAllowedFields(["xxx","xxx","xxx"]);  }    ...  } |

1. 黑名单

|  |
| --- |
| @Controller  public class companyInfoController  {  @InitBinder  public void initBinder(WebDataBinder binder, WebRequest request)  {  binder.setDisallowedFields(["companyName"]);  }    ...  } |

**注意事项：**暂无

* + 1. 关键逻辑判断前端验证

**漏洞描述：**关键逻辑仅在前端Javascript处进行验证，导致攻击者可以绕过前端验证直接向服务端提交数据。

**测试方法：**

1. 开启代理软件拦截请求功能，查看关键逻辑判断是发生在前端还是向服务端发送请求去验证；
2. 若在前端验证，则可通过禁用javascript绕过前端检测，或先在前端输入符合条件的数据，然后拦截请求包，修改参数。

**风险分析：**关键逻辑判断前端验证通常发生在上传文件时校验文件格式、验证表单输入内容是否合法、验证码校验等场景。攻击者可利用该漏洞上传任意文件、插入跨站脚本等。

**风险等级：**

【**高危**】：前端进行校验，绕过之后可执行后续业务操作/获取服务器权限/修改用户信息/修改业务数据/验证码本地校验。

【**中危**】：前端进行校验，绕过用户名密码组成规则或对业务影响不大。

**修复方案：**建议在不影响业务的前提下，关键业务逻辑放在服务端判断。

**注意事项：**暂无

* 1. 防护功能缺失
     1. Cookie属性问题

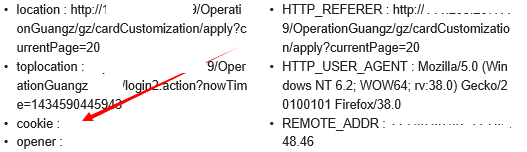
**漏洞描述：**Cookie属性缺乏相关的安全属性，如Secure属性、HttpOnly属性、Domain属性、Path属性、Expires属性等。

**测试方法：**

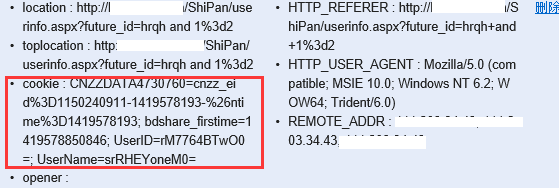
1. 通过用web扫描工具进行对网站的扫描，如果存在相关cookies的安全性问题，则一般工具都会检测出来，误报率小。
2. 或在浏览器调试窗口的网络请求处查看HTTP Header，判断是否设置Cookie属性。

**风险分析：**cookie的属性设置不当可能会造成系统用户安全隐患，Cookie信息泄露是Cookie http only配置缺陷引起的，在设置Cookie时，可以设置的一个属性，如果Cookie没有设置这个属性，该Cookie值可以被页面脚本读取。 例如：当攻击者发现一个XSS漏洞时，通常会写一段页面脚本，窃取用户的Cookie，如果未设置http only属性，则可能导致用户Cookie信息泄露，攻击者能够利用该用户的身份进行系统资源访问及操作。如图是设置了cookies属性和没有设置属性，被XSS跨站截获的cookies对比：

设置了httponly属性：

****

未设置httponly属性：

****

**风险等级：**

【**低危**】：存在未设置 HttpOnly 或 Secure 属性；Domain 或者 Path 设置路径不合理。

**修复方案：**如果网站基于cookie而非服务器端的验证，建议设置Cookie的一些安全属性，jsp参考代码如下：

|  |
| --- |
| response.setHeader("SET-COOKIE",  "user=" + request.getParameter("cookie") + "; HttpOnly"); |

PHP中的设置如下：

PHP5.2以上版本已支持HttpOnly参数的设置，同样也支持全局的HttpOnly的设置，在php.ini中设置session.cookie\_httponly，设置其值为1或者TRUE，来开启全局的Cookie的HttpOnly属性，当然也支持在代码中来开启：

|  |
| --- |
| <?php ini\_set("session.cookie\_httponly", 1);  // or session\_set\_cookie\_params(0, NULL, NULL, NULL, TRUE);  ?> |

Cookie操作函数setcookie函数和setrawcookie函数也专门添加了第7个参数来做为HttpOnly的选项，开启方法为：

|  |
| --- |
| setcookie("abc", "test", NULL, NULL, NULL, NULL, TRUE);  setrawcookie("abc", "test", NULL, NULL, NULL, NULL, TRUE); |

对于PHP5.1以前版本以及PHP4版本的话，则需要通过header函数来设置：

|  |
| --- |
| <?php header("Set-Cookie: hidden=value; httpOnly"); ?> |

**注意事项：**暂无

* + 1. 会话重用

**漏洞描述：**用户退出系统后，服务器端Session未失效，攻击者可利用此Session向服务器继续发送服务请求。

**测试方法：**通过客户端提供的注销功能退出客户端登录，利用登录时的会话再次向服务器发出操作请求，判断服务器是否返回操作结果。

**风险分析：**攻击者通过网络嗅探或者钓鱼攻击窃取用户的Session信息，在用户注销系统后，如服务器端未直接清理此Session，则攻击者仍可利用窃取到的Session成功访问系统直到Session过期。

**风险等级：**

【**高危**】：关闭浏览器后或退出后返回之前页面会话依然有效

**修复方案：**用户退出系统后，服务器端应清空此用户的Session信息。

**注意事项：**暂无

* + 1. 会话失效时间过长

**漏洞描述：**应用系统的会话失效时间过长。导致服务器性能受损，且由于过长的失效时间会导致可以被多次利用。

**测试方法：**登录目标系统获得会话，几个小时之后再次利用该会话操作目标系统，测试会话是否失效。

**风险分析：**实施会话保持攻击的前提是攻击者已通过网络中间人攻击窃取用户的SessionID，并能修改用户的Cookie的Expire时间为长久有效，用户不退出系统的前提下，攻击者可长时间利用用户的SessionID登录系统。

**风险等级：**

【**低危**】：公网系统会话失效时间超过 30 分钟

**修复方案：**

1. 服务器端设置Session的存活时间，超过存活时间强制销毁Session；
2. 当客户端发生变化时（比如IP、UserAgent等信息），要求用户重新登录；
3. 限制每个用户只允许拥有一个有效Session，当用户再次登录时，攻击者所保持的Session便会失效。

**注意事项：**暂无

* 1. 防护功能滥用
     1. 恶意锁定问题

**漏洞描述：**通过不断的输入错误的密码可恶意锁定任意账号

**测试方法：**针对测试账户，不断输入错误的密码，直至将其锁定。

**风险分析：**若系统认证功能无防自动化处理模块，攻击者可编写脚本批量锁定系统账号。

**风险等级：**

【**中危**】：锁定账户之后，可继续使用认证功能，导致可批量自动化账户锁定。

【**低危**】：锁定账户之后，可继续使用认证功能，但认证存在防自动化功能。

**修复方案：**

1. 账户锁定之后应不能继续使用认证功能
2. 认证功能防自动化操作，如添加图形验证码。

**注意事项：**暂无

* + 1. 短信炸弹

**漏洞描述：**短信轰炸攻击时常见的一种攻击，攻击者通过网站页面中所提供的发送短信验证码的功能处，通过对其发送数据包的获取后，进行重放，如果服务器短信平台未做校验的情况时，系统会一直去发送短信，这样就造成了短信轰炸的漏洞。

**测试方法：**

1. 手工找到有关网站注册页面，认证页面，是否具有短信发送页面，如果有，则进行下一步。
2. 通过利用burp或者其它抓包截断工具，抓取发送验证码的数据包，并且进行重放攻击，查看手机是否在短时间内连续收到10条以上短信，如果收到大量短信，则说明存在该漏洞。

**风险分析：**攻击者通过填写他人的手机号，使用软件burpsuite的intruder功能重复提交发送短信的请求包，达到短时间内向他人的手机上发送大量垃圾短信的目的。

**风险等级：**

【**高危**】：可对任意手机号轰炸

【**中危**】：只可对当前手机号轰炸

**修复方案：**

1. 合理配置后台短信服务器的功能，对于同一手机号码，发送次数不超过3-5次，并且可对发送的时间间隔做限制。
2. 页面前台代码编写时，加入禁止针对同一手机号进行的次数大于N次的发送，或者在页面中加入验证码功能，并且限制发送的时间间隔。

**注意事项：**暂无

* + 1. 邮件炸弹

**漏洞描述：**应用系统未限制邮件的发送次数和频率，造成短时间内大量邮件发送至接收者邮箱，造成大量垃圾邮件。

**测试方法：**

1. 手工找到有关网站注册页面，认证页面，是否具有邮件发送页面，如果有，则进行下一步。
2. 通过利用burp或者其它抓包截断工具，抓取发送邮件的数据包，并且进行重放攻击，查看邮箱是否在短时间内连续收到10封以上邮件，如果收到大量邮件，则说明存在该漏洞。

**风险分析：**攻击者通过填写他人的邮箱地址，使用软件burpsuite的intruder功能重复提交发送邮件的请求包，达到短时间内向他人的邮箱中发送大量垃圾邮件的目的。

**风险等级：**

【**高危**】：可对任意邮箱轰炸

【**中危**】：只可对当前邮箱轰炸

**修复方案：**

1. 合理配置后台邮件服务器的功能，对于同一邮箱，发送次数不超过3-5次，并且可对发送的时间间隔做限制。
2. 页面前台代码编写时，加入禁止针对同一邮箱进行的次数大于N次的发送，或者在页面中加入验证码功能，并且限制发送的时间间隔。

**注意事项：**暂无

* 1. 权限缺失
     1. Flash跨域访问

**漏洞描述：**flash跨域通信，依据的是crossdomain.xml文件。该文件配置在服务端，一般为根目录下，限制了flash是否可以跨域获取数据以及允许从什么地方跨域获取数据。举个例子：

1、www.a.com域下不存在crossdomain.xml文件，则不允许除了www.a.com域之外的其他任何域下的flash进行跨域请求。

2、www.a.com域下存在crossdomain.xml文件，如若配置 allow-access-from 为www.b.com，则只允许www.b.com域下的flash进行跨域请求，以及来自自身域www.a.com的网络请求。 crossdomain.xml需严格遵守XML语法，有且仅有一个根节点cross-domain-policy，且不包含任何属性。在此根节点下只能包含如下的子节点：

site-control

allow-access-from

allow-access-from-identity

allow-http-request-headers-from

如果crossdomain.xml文件配置allow-access-from为\*，将允许任何域下的flash进行跨域请求。

**测试方法：** 下面列举一个具体的利用场景：

http://www.xxx.com/crossdomain.xml，文件内容如下图所示：

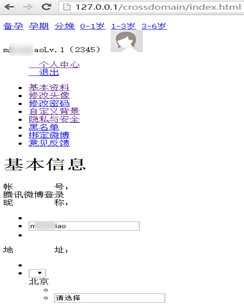


利用开源的FlashHTTPRequest，下载地址：https://github.com/mandatoryprogrammer/FlashHTTPRequest，FlashHTTPRequest，使用方法参考README。

在本地服务器编写一个HTML页面，嵌入FlashHTTPRequest.swf，此页面的目的是获取受害者的账号设置页面：http://www.xxx.com/user-setting，源代码如下：

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="utf-8"/>  <script src="flashhttprequest.js"></script>  <script>  var res="";  function setres(data){ //将返回结果打印至页面中  window.res=data;  //console.log(data);  document.write(data);  }  function onhook() {  FlashHTTPRequest.open('GET', 'http://www.xxx.com/user-setting', '', 'setres' );  }  </script>  </head>  <body>  <div id="flashBridge"></div>  </body>  </html> |

登录该网站后，访问构造好的HTML页面：http://127.0.0.1/crossdomain/index.html，页面内容是该网站的账号设置页面，如下图所示：



由此也可以构造CSRF攻击……

**风险分析：**攻击者通过如下攻击过程可获取用户敏感信息：

1. 攻击者构造一个恶意的flash文件 ，在attacker.com中嵌入该flash文件
2. 受害者已登录victim.com
3. 受害者浏览含有恶意flash文件的页面
4. flash文件以受害者的session向victim.com发出任意请求，并接收返回数据
5. 攻击者保存受害者的信息

**风险等级：**

【**中危**】：可获取数据

【**低危**】：allow-access-from属性为\*

【**低危**】：allow-http-request-headers-from属性为\*

【**低危**】：site-control标签的permitted-cross-domain-policies属性为 all

**修复方案：**限制跨域请求来源，即设置allow-access-from的值，只允许可信域的flash跨域请求。

**注意事项：**暂无

* + 1. jsonp跨域请求

**漏洞描述：**当某网站通过 JSONP 的方式来传递用户认证后的敏感信息时，攻击者通过构造恶意的 JSONP 调用页面，诱导被攻击者访问来达到截取用户敏感信息的目的。

**测试方法：**一个典型的 JSON劫持攻击代码：  
<script>  
function test(v){  
    alert(v.username);  
}  
</script>  
<script src="http://www.xxx.com/?o=sso&m=info&func=test"></script>

当被攻击者在登陆www.xxx.com网站的情况下访问了该网页时，那么用户的隐私数据（如用户名，邮箱等）可能被攻击者劫持。

**风险分析：**攻击者可利用该漏洞结合社会工程学，诱导用户点击某个精心构造的页面，从而达到窃取用户敏感信息的目的。

**风险等级：**

【**高危**】：可获取敏感数据

【**中危**】：可获取普通数据

**修复方案：**建议从以下几个方面进行防御：

1. 严格安全的实现CSRF方式调用JSON文件：限制Referer、部署一次性Token等。
2. 严格安装JSON格式标准输出Content-Type及编码（Content-Type : application/json; charset=utf-8）。
3. 严格过滤callback函数名及JSON里数据的输出。
4. 严格限制对JSONP输出callback函数名的长度。
5. 在callback输出之前加入其他字符(如：/\*\*/、回车换行)这样既不影响JSON文件加载，又能一定程度预防其他文件格式的输出。

**注意事项：**暂无

* + 1. 未授权访问

**漏洞描述：**未授权访问漏洞，是在攻击者没有获取到登录权限或未授权的情况下，或者不需要输入密码，即可通过直接输入网站控制台主页面地址，或者不允许查看的链接便可进行访问，同时进行操作。

**测试方法：**

1. 通过对登录后的页面进行抓包，将抓取到的功能链接，在其他浏览器进行打开。
2. 也可以通过web扫描工具进行扫描，爬虫得到相关地址链接，进行直接访问，如果未进行跳转到登录页面，则可判断为存在未授权访问漏洞。

**风险分析：**攻击者猜测管理后台地址或利用已知的访问地址，尝试在未登录的情况下直接访问，常见的后台登录地址有：admin.jsp、index.jsp、main.jsp、left.jsp、right.jsp、top.jsp等。攻击者进入管理后台后可修改Web应用程序设置，利用后台对上传文件过滤不严的缺陷，上传webshell等实施恶意操作。

**风险等级：**

【**高危**】：认证模式可绕过，不登录即可通过 URL 或其他方式访问登陆后页面

**修复方案：**常见的修复方法：在系统中，加入用户身份认证机制或者tonken验证，防止可被直接通过连接就可访问到用户的功能进行操作，简而言之，一定对系统重要功能点增加权限控制，对用户操作进行合法性验证，下列为针对常见的JSP语言编写的网站的安全修复方案：

1. 采用JAVA过滤器技术，对/pages下所有URL进行登录状态检查，通过session.getAttribute()方法从session中获取登录成功时存入session中的身份标识，判断客户端传递过来的身份标识是否与session中保存的一致，不一致则跳转到登录页面。关键代码如下：

|  |
| --- |
| //从session里取的用户名信息  String username = (String) session.getAttribute("userID");  //getAttribute中变量根据实际变量传入。  //判断如果没有取到用户信息,就跳转到登陆页面  if ((username == null) || "".equals(username)) {  //跳转到登陆页面  res.sendRedirect("http://" + req.getHeader("Host") +"/login\_oa.jsp");}  else {  //已经登陆,继续此次请求  chain.doFilter(req, res); }} |

1. 进行权限判断，以下代码为过滤器程序，通过会话获取用户身份信息，进行权限判断等操作：

|  |
| --- |
| //在配置文件中设置过滤器  <filter>  <filter-name>SessionFilter</filter-name>  <filter-class>com.nsfocus.frame.filter.SessionFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>SessionFilter</filter-name>  <url-pattern>/pages/\*</url-pattern>  </filter-mapping>  <filter>  ----------------------------------------------------------------------------------------------  //后台过滤程序  public void doFilter(ServletRequest request,  ServletResponse response,FilterChain chain) throws  IOException, ServletException {  HttpServletRequest req = (HttpServletRequest) request;  HttpServletResponse res = (HttpServletResponse) response;  HttpSession session = req.getSession(true);  //从session里取的用户名信息  String username = (String) session.getAttribute("userID");  //getAttribute中变量根据实际变量传入。  //判断如果没有取到用户信息,就跳转到登陆页面  if ((username == null) || "".equals(username)) {  //跳转到登陆页面  res.sendRedirect("http://" + req.getHeader("Host") +"/login\_oa.jsp");}  else {  //已经登陆,继续此次请求  chain.doFilter(req, res); } }  public void destroy() { }  } |

**注意事项：**暂无

* 1. 权限篡改
     1. 任意用户密码修改/重置

**漏洞描述：**可通过篡改用户名或ID、暴力破解验证码等方式修改/重置任意账户的密码。

**测试方法：**

密码修改的步骤一般是先校验用户原始密码是否正确，再让用户输入新密码。修改密码机制绕过方式大概有以下三种：

1. 如果输入新密码的接口可以直接访问，那么在未知原始密码的的情况下即可直接修改密码，通常知道了他人的用户名即可任意修改他人的密码。
2. 如果系统未校验修改密码的用户身份，那么在提交修改密码请求时，攻击者通过输入密码，将用户名或者用户ID修改为其他人的，即可成功修改他人的密码。
3. 当修改密码时系统需要电子邮件或者手机短信确认，而应用程序未校验用户输入的邮箱和手机号，那么攻击者通过填写自己的邮箱或手机号接收修改密码的链接和验证码，以此修改他人的密码。

密码重置机制绕过攻击方式主要有以下两种：

1. 通过正常手段获取重置密码的链接，猜解链接的组成结构和内容（如用户名或者时间戳的MD5值）。在得知他人邮箱的情况下，构造重置他人密码的链接。
2. 在得知他人手机号的情况下，通过穷举手机验证码重置他人的密码。

**风险分析：**

密码修改功能常采用分步骤方式来实现，攻击者在未知原始密码的情况下绕过某些检验步骤修改用户密码。

重置密码过程一般是首先验证注册的邮箱或者手机号，获取重置密码的链接（一般会包含一串唯一的字符串）或者验证码，然后访问重置密码链接或者输入验证码，最后输入新密码。密码重置机制绕过攻击是指在未知他人的重置密码链接或手机验证码的情况下，通过构造重置密码链接或穷举手机验证码的方式直接重置他人的密码。

**风险等级：**

【**高危**】：其它用户的密码被修改/重置成功

**修复方案：**

1. 一次性填写校验信息（原始密码、新密码等）后再提交修改密码请求。
2. 对客户端提交的修改密码请求，应对请求的用户身份与当前登录的用户身份进行校验，判断是否有权修改用户的密码并对原始密码是否正确也进行判断。
3. 不应将用于接收验证信息的手机、邮箱等信息全部明文传到客户端，应对手机、邮箱等信息进行屏蔽处理，或不将此类信息返回到客户端。
4. 对原始密码进行了验证的情况下，限制输入原始密码的错误次数，防止攻击者暴力破解原始密码。
5. 重置密码链接中的关键信息应随机化，不可预测（例如token机制），且禁止将关键信息返回到客户端。

**注意事项：**暂无

* + 1. SSO认证缺陷

**漏洞描述：**SSO认证存在缺陷，可越权登录他人账户。

**测试方法：**建议从以下两个方向进行测试：

1. 信息传输缺乏安全保证

SSO认证通信过程中大多数采用明文形式传送敏感信息，这些信息很容易被窃取，致使重要信息泄露。另外，在通信过程中大多数方案也没有对关键信息进行签名，容易遭到伪装攻击。

1. Web服务的安全缺陷

由于单点登录基本上是基于Web服务实现的，所以也不可避免的存在Web服务的安全缺陷，如跨站脚本攻击、越权攻击等。

**风险分析：**因为只需要登录一次，所有的授权的应用系统都可以访问，可能导致一些很重要的信息泄露。

**风险等级：**

【**高危**】：可导致用户在单点登录之后任意登录其他系统和其他用户账号信息

**修复方案：**建议从以下几个方面进行防御：

1. 建议在不影响业务的前提下，使用HTTPS协议传输
2. 严格校验SSO认证过程中的用户身份
3. 过滤用户传入的参数，对特殊符号进行转义或屏蔽。

**注意事项：**暂无

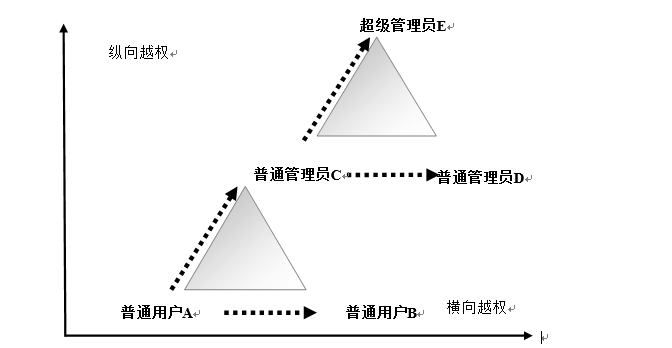
* + 1. 越权

**漏洞描述：**越权访问，这类漏洞是指应用在检查授权（Authorization）时存在纰漏，使得攻击者在获得低权限用户帐后后，可以利用一些方式绕过权限检查，访问或者操作到原本无权访问的高权限功能。在实际的代码安全审查中，这类漏洞往往很难通过工具进行自动化检测，因此在实际应用中危害很大。其与未授权访问有一定差别。

**测试方法：**

1. 以超管 admin（高权限用户） 身份登录系统
2. 找到一个只有超管（高权限）才有的功能的链接，比如:"http://localhost/mywebappname/userManage/userList.do" , 显示出所有的user，并复制此链接。
3. 以普通用户登陆进系统,在地址栏输入: userManage/userList.do，如果可以查看到其所有的user，则就造成了，普通用户的越权访问。
   * 检测说明：权限管理测试更多的是进行人工分析，自动化工具无法了解页面的具体应用场景以及逻辑判断过程。因此这里的测试需要首先测试人员理解测试业务系统的逻辑处理流程，并在此基础上进行如下测试。这里有几个测试的参考依据：
   * 页面是否进行权限判断；
   * 页面提交的资源标志是否与已登陆的用户身份进行匹配比对；
   * 用户登陆后，服务器端不应再以客户端提交的用户身份信息为依据，而应以会话中保存的已登陆的用户身份信息为准；
   * 必须在服务器端对每个请求URL进行鉴权，而不能仅仅通过客户端的菜单屏蔽或者按钮Disable来限制。

**风险分析：**目前存在着两种越权操作类型：横向越权操作和纵向越权操作。前者指的是攻击者尝试访问与他拥有相同权限的用户的资源；而后者指的是一个低级别攻击者尝试访问高级别用户的资源,如图所示的情况。



**风险等级：**

【**高危**】：任意水平或垂直越权

**修复方案：**对用户操作进行权限校验，防止通过修改参数进入未授权页面及进行非法操作，建议在服务端对请求的数据和当前用户身份做校验检查。流程描述：在服务器接收到用户发送的页面访问请求时，根据预设的识别策略，从用户的页面访问请求中提取该用户对应的用户唯一标识信息，同时提取所述页面访问请求对应的应答页面中的表单及该表单中不可修改参数，将所述表单及不可修改参数与所述用户唯一标识信息绑定后记录到参数列表中；检测到用户提交请求页面的表单时，将所述请求页面的表单及不可修改参数与该用户对应的所述参数列表中记录的表单及不可修改参数进行比对，控制该用户的访问。例如：

|  |
| --- |
| 登陆时将用户名存入session  session.setAttribute("username",username);  在相关页面判断  if((String)session.getAttribute("username")!=admin){  (response.sendRedirect("XXX.jsp")); |

注意： xxx.jsp为自定义的错误页面

**注意事项：**暂无

* + 1. Cookies伪造

**漏洞描述：**Cookie伪造攻击指攻击者通过修改cookie（Web网站为了辨别用户身份而储存在用户本地终端上的数据（通常经过加密）），获得用户未授权信息，进而盗用身份的过程，攻击者利用此漏洞可打开新账号或获取用户已存在账号的访问权限。

**测试方法：**使用burpsuite等代理软件或浏览器Cookie修改插件，篡改Cookie信息。

**风险分析：**Cookie中通常使用以下几种方式来确定一个会话：

1. 唯一标识ID
2. 使用token
3. 时间戳

攻击者可以对Cookie中的属性进行逆向分析，当Cookie中的唯一标识ID、token和时间戳可预测（如ID采用顺序增加方式，token值是用户名的MD5值）时，攻击者可通过遍历唯一标识ID、猜测token手段构造Cookie，来模拟一个有效的用户登录系统。

**风险等级：**

【**高危**】：伪造 cookie 后，成功进入业务系统或获取用户权限

**修复方案：**

1. 服务器端在Set-Cookie时在Cookie的值后面加上一段防篡改的验证串，然后再发送到客户端，客户端得到的Cookie形如：user\_name=admin|ab95ef23cc6daecc475de，用|分割的后面部分是防篡改验证串，可以使用以下方式生成：SHA1（cookie内容+密钥）、DES（cookie的内容+密钥）、MD5（cookie内容+密钥），密钥是保存在服务器端的固定字符串，应妥善保管。服务器接收到Cookie后，根据Cookie内容和密钥重新计算防篡改验证串，然后和客户端提交的防篡改验证串比较，若两者一致则认为Cookie未被篡改。
2. 若使用了WebSphere中间件，通过配置LTPA认证方式可防止Cookie被篡改。

**注意事项：**暂无

* + 1. 会话变量可控

**漏洞描述：**会话中的参数可由客户端控制，导致客户端可以控制并修改会话参数。

**测试方法：**查看HTTP请求，分析会话中的可控参数并修改。

**风险分析：**攻击者可通过篡改会话变量，达到欺骗服务器的目的。

**风险等级：**

【**中危**】：会话中存在字段是由可控参数生成

**修复方案：**禁止从客户端传入会话变量

**注意事项：**暂无

* + 1. 跨站请求伪造（CSRF）

**漏洞描述：**跨站请求伪造攻击，Cross-Site Request Forgery（CSRF），攻击者在用户浏览网页时，利用页面元素（例如img的src），强迫受害者的浏览器向Web应用服务器发送一个改变用户信息的HTTP请求。CSRF攻击可以从站外和站内发起。从站内发起CSRF攻击，需要利用网站本身的业务，比如“自定义头像”功能，恶意用户指定自己的头像URL是一个修改用户信息的链接，当其他已登录用户浏览恶意用户头像时，会自动向这个链接发送修改信息请求。从站外发送请求，则需要恶意用户在自己的服务器上，放一个自动提交修改个人信息的htm页面，并把页面地址发给受害者用户，受害者用户打开时，会发起一个请求。威胁描述：攻击者使用CSRF攻击能够强迫用户向服务器发送请求，导致用户信息被迫修改，甚至可引发蠕虫攻击。如果恶意用户能够知道网站管理后台某项功能的URL，就可以直接攻击管理员，强迫管理员执行恶意用户定义的操作。

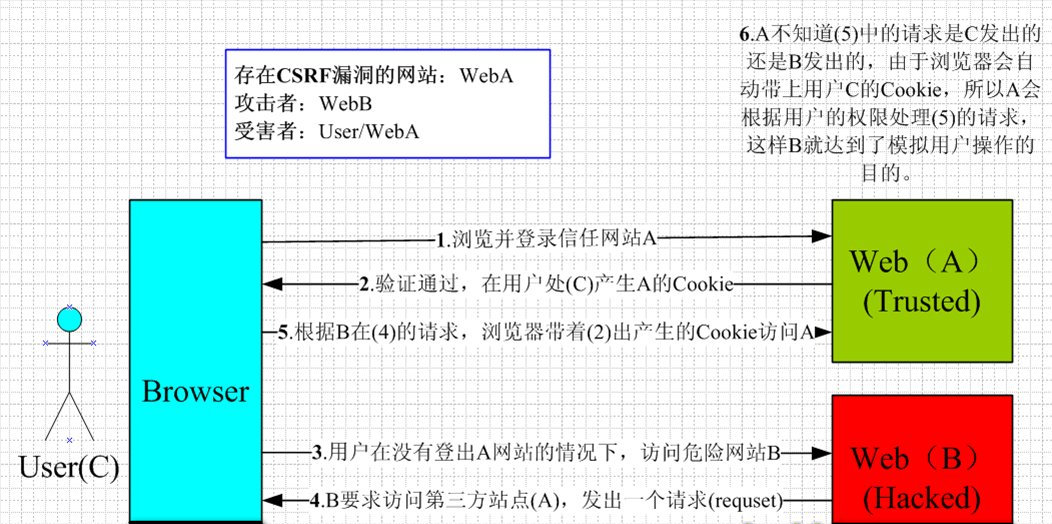
**测试方法：**

1. 检测方式多种多样：工具常常会扫描得到CSRF的漏洞，但是一般常常为误报，重点还是依靠手工来进行检测，以下来举例说明其中一种检测以及攻击方案：
2. 设置页面test.htm中，页面中有一个表单，和一段脚本，脚本的作用是，当页面加载时，浏览器会自动提交请求。页面代码如下：

|  |
| --- |
| <form id="modify" action="http://www.test.com/servlet/modify" method="POST">  <input name="email">  <input name="tel">  <input name="realname">  <input name="userid">  <input type="submit">  </form>  <script>  document.getElementById("modify").submit();  </script> |

1. 诱使用户在登录目标系统后访问URL链接http://xx.x.xx.xxx /test.htm
2. 用户打开test.htm后，会自动提交表单，发送给www.test.com下的那个存在CSRF漏洞的web应用，用户信息被篡改。
3. 在整个攻击过程中，受害者用户仅看到了一个空白页面（可以伪造成其他无关页面），并且不知道自己的信息已经被修改。

**风险分析：**结合社会工程学（如通过电子邮件或聊天发送的链接），攻击者诱骗受害者点击恶意链接，而恶意链接中包含了诸如转账等敏感操作。下图简单阐述了CSRF攻击的思想：



从上图可以看出，完成一次CSRF攻击，受害者必须依次完成两个步骤：

1. 登录受信任网站A，并在本地生成Cookie。
2. 在不退出A的情况下，访问危险网站B。

CSRF利用方式可分为以下两种：

1. GET类型的CSRF

GET类型的CSRF利用非常简单，只需要一个HTTP请求。攻击者诱骗受害者访问恶意页面，通常这个恶意页面包含标签<img src=http://www.mybank.com/Transfer.jsp?toBankId=hackerID&money=1000 />，受害者一旦访问了恶意页面会向http://www.mybank.com/Transfer.jsp?toBankId=hackerID&money=1000发送HTTP请求，该请求是转账操作，受害者在不知情的情况下向攻击者账户转账了1000元。

1. POST类型的CSRF

为了杜绝上面的问题，银行决定改用POST请求完成转账操作。因此，银行网站A的Web表单如下：

|  |
| --- |
| <form action="Transfer.jsp" method="POST">  　<p>ToBankId: <input type="text" name="toBankId" /></p>  　<p>Money: <input type="text" name="money" /></p>  　<p><input type="submit" value="Transfer" /></p>  </form> |

后台处理页面Transfer.jsp的伪代码如下：

|  |
| --- |
| public void doPost(HttpServletRequest req, HttpServletResponse res)  {  HttpSession session = request.getSession(false);  If(session==null)  {  response.sendRedirect("../SessionLogin.htm");  }  String bankID=request.getParameter(“toBankId”);  String money=request.getParameter(“money”);  BankTools.transferMoney(bankID,double.parse(money));  } |

危险网站B的HTML代码如下：

|  |
| --- |
| <html>  <head>  　<script type="text/javascript">  　　　　function steal()  　　　　　{  　　iframe = document.frames["steal"];  　　 　　 iframe.document.Submit("transfer");  　　　　　}  　</script>  </head>  <body onload="steal()">  　<iframe name="steal" display="none">  　　　<form method="POST" name="transfer"　action="http://www.myBank.com/Transfer.jsp">  　　　　<input type="hidden" name="toBankId" value="11">  　　　　<input type="hidden" name="money" value="1000">  　　　</form>  　</iframe>  </body>  </html> |

从以上代码看出，危险网站B暗地里发送了POST请求到银行，且该POST请求是转账操作，攻击者访问危险网站B时在不知情的情况下向攻击者转账了1000元。

**风险等级：**

【**高危**】：核心系统关键操作（账户操作，审批确认……）

【**中危**】：普通系统关键操作（账户操作，审批确认……）

**修复方案：**

1. 通过referer判断页面来源进行CSRF防护，该方式无法防止站内CSRF攻击及referer字段伪造。
2. 重要功能点使用动态验证码进行CSRF防护。
3. 通过token方式进行CSRF防护：

|  |
| --- |
| 1. 在Session中绑定token。如果不能保存到服务器端Session中，则可以替代为保存到Cookie里。 2. 在form表单中自动填入token字段，比如 <input type=hidden name="anti\_csrf\_token" value="$token" />。 3. 在HTTP请求中自动添加token。   在服务器端对比POST提交参数的token与Session中绑定的token是否一致，以验证CSRF攻击 |

1. 为每个session创建唯一的随机字符串，并在受理请求时验证：

|  |
| --- |
| <form action="/transfer.do" method="post">  <input type="hidden" name="randomStr" value=<%=request.getSession().getAttribute("randomStr")%>>  ......  </form>  //判断客户端提交的随机字符串是否正确  String randomStr = (String)request.getParameter("randomStr");  if(randomStr == null) randomStr="";  if(randomStr.equals(request.getSession().getAttribute("randomStr")))  {//处理请求}  else{  //跨站请求攻击，注销会话  } |

**注意事项：**暂无

* 1. 综合利用
     1. 跨站脚本攻击（XSS）

**漏洞描述：**跨站脚本攻击的英文全称是Cross Site Script，为了和样式表区分，缩写为XSS。发生的原因是网站将用户输入的内容输出到页面上，在这个过程中可能有恶意代码被浏览器执行。跨站脚本攻击,它指的是恶意攻击者往Web页面里插入恶意html代码，当用户浏览该页之时，嵌入其中Web里面的html代码会被执行，从而达到恶意用户的特殊目的。已知的跨站脚本攻击漏洞有三种：1）存储式；2）反射式；3）基于DOM。

1. 存储型跨站脚本攻击涉及的功能点：用户输入的文本信息保存到数据库中，并能够在页面展示的功能点，例如用户留言、发送站内消息、个人信息修改等功能点。
2. 反射型跨站脚本攻击涉及的功能点：URL参数需要在页面显示的功能点都可能存在反射型跨站脚本攻击，例如站内搜索、查询功能点。
3. 基于DOM跨站脚本攻击涉及的功能点：涉及DOM对象的页面程序，包括（不限这些）：

|  |
| --- |
| document.URL  document.URLUnencoded  document.location  document.referrer  window.location |

**测试方法：**

1. GET方式跨站脚本：
2. 在输入的参数后逐条添加以下语句，以第一条为例，输入http://www.exmaple.com/page.xxx?name=<script>alert(123456)</script>只要其中一条弹出显示123456的告警框，就说明存在跨站漏洞，记录漏洞，停止测试。
3. 如果没有弹出显示123456的告警框，则在返回的页面上单击鼠标右键，选择“查看源文件”。
4. 查找网页源文件中是否包含完整的字符串<script>alert(123456)</script>，则不管有没有弹出显示123456的告警框，都表明存在跨站脚本漏洞。
5. 由于有些HTML元素（比如<textarea>或”）会影响脚本的执行，所以不一定能够正确弹出123456告警框，需要根据返回网页源文件的内容，构造value的值，比如

|  |
| --- |
| 多行文本输入框：  </textarea><script>alert(123456)</script>  文本输入框：  </td><script>alert(123456)</script>  '><script>alert(123456)</script>  "><script>alert(123456)</script>  </title><script>alert(123456)</script>  --><script>alert(123456)</script>  [img]javascript:alert(123456)[/img]  <scrip<script>t>alert(123456)</scrip</script>t>  </div><Script>alert(123456)</script>  需要对页面上所有可以提交参数的地方进行测试。具体跨站脚本的测试语句根据实际情况的不同而不同，可自行构造，以及触发事件等切换，这里只列出了一些最常见构造语句。 |

1. Post方式跨站脚本：
2. 在POST表单中逐条输入以下语句，只要其中一条弹出显示123456的对话框，就说明存在跨站漏洞，记录漏洞，停止测试。
3. <script>alert(123456)</script>
4. [img]javascript:alert(123456);[/img]
5. 如果没有弹出显示123456的告警框，则在返回的页面上单击鼠标右键，选择“查看源文件”
6. 查找网页源文件中是否包含完整的字符串<script>alert(123456)</script>，则不管有没有弹出显示123456的告警框，都表明存在跨站脚本漏洞。
7. 由于有些HTML元素（比如<textarea>或”）会影响脚本的执行，所以不一定能够正确弹出123456告警框，需要根据返回网页源文件的内容，构造value的值，比如

|  |
| --- |
| 多行文本输入框：  </textarea><script>alert(123456)</script>  文本输入框：  </td><script>alert(123456)</script>  '><script>alert(123456)</script>  "><script>alert(123456)</script>  </title><script>alert(123456)</script>  --><script>alert(123456)</script>  [img]javascript:alert(123456)[/img]  <scrip<script>t>alert(123456)</scrip</script>t>  </div><Script>alert(123456)</script>  需要对页面上所有可以提交参数的地方进行测试。具体跨站脚本的测试语句根据实际情况的不同而不同，可自行构造，以及触发事件等切换，这里只列出了一些最常见构造语句。 |

**风险分析：**

常见的反射型跨站脚本攻击步骤如下：

1. 攻击者创建并测试恶意URL；
2. 攻击者确信受害者在浏览器中加载了恶意URL；
3. 攻击者采用反射型跨站脚本攻击方式安装键盘记录器、窃取受害者的cookie 、窃取剪贴板内容、改变网页内容（例如下载链接）。

存储型跨站脚本攻击最为常见的场景是将跨站脚本写入文本输入域中，如留言板、博客或新闻发布系统的评论框。当用户浏览留言和评论时，浏览器执行跨站脚本代码。

**风险等级：**

【**高危**】：应用中存在存储型跨站

【**中危**】：应用中存在反射型跨站

**修复方案：**对于XSS跨站漏洞，可以采用以下修复方式：

1. 总体修复方式：验证所有输入数据，有效检测攻击；对所有输出数据进行适当的编码，以防止任何已成功注入的脚本在浏览器端运行。具体如下 ：
2. 输入验证：某个数据被接受为可被显示或存储之前，使用标准输入验证机制，验证所有输入数据的长度、类型、语法以及业务规则。
3. 输出编码：数据输出前，确保用户提交的数据已被正确进行entity编码，建议对所有字符进行编码而不仅局限于某个子集。
4. 明确指定输出的编码方式：不要允许攻击者为你的用户选择编码方式(如ISO 8859-1或 UTF 8)。
5. 注意黑名单验证方式的局限性：仅仅查找或替换一些字符(如"<" ">"或类似"script"的关键字)，很容易被XSS变种攻击绕过验证机制。
6. 警惕规范化错误：验证输入之前，必须进行解码及规范化以符合应用程序当前的内部表示方法。请确定应用程序对同一输入不做两次解码。对客户端提交的数据进行过滤，一般建议过滤掉双引号（”）、尖括号（<、>）等特殊字符，或者对客户端提交的数据中包含的特殊字符进行实体转换，比如将双引号（”）转换成其实体形式&quot;，<对应的实体形式是&lt;，<对应的实体形式是&gt;以下为需过滤的常见字符：

|  |
| --- |
| [1] |（竖线符号）  [2] & （& 符号）  [3];（分号）  [4] $（美元符号）  [5] %（百分比符号）  [6] @（at 符号）  [7] '（单引号）  [8] "（引号）  [9] \'（反斜杠转义单引号）  [10] \"（反斜杠转义引号）  [11] <>（尖括号）  [12] ()（括号）  [13] +（加号）  [14] CR（回车符，ASCII 0x0d）  [15] LF（换行，ASCII 0x0a）  [16] ,（逗号）  [17] \（反斜杠）  2、在请求返回页面关键字符进行转义；  [1] “（双引号）：&quot  [2] ’ （单引号）：&apos  [3] &（&符号）：&amp  [4] <（左尖括号）：&lt  [5] >（右尖括号）：&gt  在不影响应用的前提下，建议将cookie标记为httpOnly，同时禁用TRACE方法。 |

1. 对于java进行的web业务开发的过滤器如下：

在用java进行web业务开发的时候，对于页面上接收到的参数，除了极少数是步可预知的内容外，大量的参数名和参数值都是不会出现触发Xss漏洞的字符。而通常为了避免Xss漏洞，都是开发人员各自在页面输出和数据入库等地方加上各种各样的encode方法来避免Xss问题。而由于开发人员的水平不一，加上在编写代码的过程中安全意识的差异，可能会粗心漏掉对用户输入内容进行encode处理。针对这种大量参数是不可能出现引起Xss和SQL注入漏洞的业务场景下，因此可以使用一个适用大多数业务场景的通用处理方法，牺牲少量用户体验，来避免Xss漏洞和SQL注入。

利用Servlet的过滤器机制，编写定制的XssFilter，将request请求代理，覆盖getParameter和getHeader方法将参数名和参数值里的指定半角字符，强制替换成全角字符。使得在业务层的处理时不用担心会有异常输入内容。

|  |
| --- |
| XssFilter.java  package filter;  import java.io.IOException;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.http.HttpServletRequest;  public class XssFilter implements Filter {  public void init(FilterConfig config) throws ServletException {  }  public void doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) throws IOException, ServletException  {  XssHttpServletRequestWrapper xssRequest = new XssHttpServletRequestWrapper(  (HttpServletRequest) request);  chain.doFilter(xssRequest, response);  }  public void destroy() {  }  XssHttpServletRequestWrapper.java  package filter;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletRequestWrapper;  public class XssHttpServletRequestWrapper extends HttpServletRequestWrapper {  HttpServletRequest orgRequest = null;  public XssHttpServletRequestWrapper(HttpServletRequest request) {  super(request);  orgRequest = request;  }  /\*\*  \* 覆盖getParameter方法，将参数名和参数值都做xss过滤。<br/>  \* 如果需要获得原始的值，则通过super.getParameterValues(name)来获取<br/>  \* getParameterNames,getParameterValues和getParameterMap也可能需要覆盖  \*/  @Override  public String getParameter(String name) {  String value = super.getParameter(xssEncode(name));  if (value != null) {  value = xssEncode(value);  }  return value;  }  /\*\*  \* 覆盖getHeader方法，将参数名和参数值都做xss过滤。<br/>  \* 如果需要获得原始的值，则通过super.getHeaders(name)来获取<br/>  \* getHeaderNames 也可能需要覆盖  \*/  @Override  public String getHeader(String name) {  String value = super.getHeader(xssEncode(name));  if (value != null) {  value = xssEncode(value);  }  return value;  }  /\*\*  \* 将容易引起xss漏洞的半角字符直接替换成全角字符  \*  \* @param s  \* @return  \*/  private static String xssEncode(String s) {  if (s == null || s.isEmpty()) {  return s;  }  StringBuilder sb = new StringBuilder(s.length() + 16);  for (int i = 0; i < s.length(); i++) {  char c = s.charAt(i);  switch (c) {  case '>':  sb.append('＞');//全角大于号  break;  case '<':  sb.append('＜');//全角小于号  break;  case '\'':  sb.append('‘');//全角单引号  break;  case '\"':  sb.append('“');//全角双引号  break;  case '&':  sb.append('＆');//全角  break;  case '\\':  sb.append('＼');//全角斜线  break;  case '#':  sb.append('＃');//全角井号  break;  default:  sb.append(c);  break;  }  }  return sb.toString();  }  /\*\*  \* 获取最原始的request  \*  \* @return  \*/  public HttpServletRequest getOrgRequest() {  return orgRequest;  }  /\*\*  \* 获取最原始的request的静态方法  \*  \* @return  \*/  public static HttpServletRequest getOrgRequest(HttpServletRequest req) {  if(req instanceof XssHttpServletRequestWrapper){  return ((XssHttpServletRequestWrapper)req).getOrgRequest();  }  return req;  }  在web.xml中添加  <filter>  <filter-name>xssFilter</filter-name>  <filter-class>filter.XssFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>xssFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

1. 对于PHP语言所编写的网站系统的XSS过滤器：

|  |
| --- |
| <?php  /\*\*  \* @去除XSS（跨站脚本攻击）的函数  \* @par $val 字符串参数，可能包含恶意的脚本代码如<script language="javascript">alert("hello world");</script>  \* @return 处理后的字符串  \* @Recoded By Androidyue  \*\*/  function RemoveXSS($val) {  // remove all non-printable characters. CR(0a) and LF(0b) and TAB(9) are allowed  // this prevents some character re-spacing such as <java\0script>  // note that you have to handle splits with \n, \r, and \t later since they \*are\* allowed in some inputs  $val = preg\_replace('/([\x00-\x08,\x0b-\x0c,\x0e-\x19])/', '', $val);    // straight replacements, the user should never need these since they're normal characters  // this prevents like <IMG SRC=@avascript:alert('XSS')>  $search = 'abcdefghijklmnopqrstuvwxyz';  $search .= 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';  $search .= '1234567890!@#$%^&\*()';  $search .= '~`";:?+/={}[]-\_|\'\\';  for ($i = 0; $i < strlen($search); $i++) {  // ;? matches the ;, which is optional  // 0{0,7} matches any padded zeros, which are optional and go up to 8 chars    // @ @ search for the hex values  $val = preg\_replace('/(&#[xX]0{0,8}'.dechex(ord($search[$i])).';?)/i', $search[$i], $val); // with a ;  // @ @ 0{0,7} matches '0' zero to seven times  $val = preg\_replace('/(&#0{0,8}'.ord($search[$i]).';?)/', $search[$i], $val); // with a ;  }    // now the only remaining whitespace attacks are \t, \n, and \r  $ra1 = Array('javascript', 'vbscript', 'expression', 'applet', 'meta', 'xml', 'blink', 'link', 'style', 'script', 'embed', 'object', 'iframe', 'frame', 'frameset', 'ilayer', 'layer', 'bgsound', 'title', 'base');  $ra2 = Array('onabort', 'onactivate', 'onafterprint', 'onafterupdate', 'onbeforeactivate', 'onbeforecopy', 'onbeforecut', 'onbeforedeactivate', 'onbeforeeditfocus', 'onbeforepaste', 'onbeforeprint', 'onbeforeunload', 'onbeforeupdate', 'onblur', 'onbounce', 'oncellchange', 'onchange', 'onclick', 'oncontextmenu', 'oncontrolselect', 'oncopy', 'oncut', 'ondataavailable', 'ondatasetchanged', 'ondatasetcomplete', 'ondblclick', 'ondeactivate', 'ondrag', 'ondragend', 'ondragenter', 'ondragleave', 'ondragover', 'ondragstart', 'ondrop', 'onerror', 'onerrorupdate', 'onfilterchange', 'onfinish', 'onfocus', 'onfocusin', 'onfocusout', 'onhelp', 'onkeydown', 'onkeypress', 'onkeyup', 'onlayoutcomplete', 'onload', 'onlosecapture', 'onmousedown', 'onmouseenter', 'onmouseleave', 'onmousemove', 'onmouseout', 'onmouseover', 'onmouseup', 'onmousewheel', 'onmove', 'onmoveend', 'onmovestart', 'onpaste', 'onpropertychange', 'onreadystatechange', 'onreset', 'onresize', 'onresizeend', 'onresizestart', 'onrowenter', 'onrowexit', 'onrowsdelete', 'onrowsinserted', 'onscroll', 'onselect', 'onselectionchange', 'onselectstart', 'onstart', 'onstop', 'onsubmit', 'onunload');  $ra = array\_merge($ra1, $ra2);  $found = true; // keep replacing as long as the previous round replaced something  while ($found == true) {  $val\_before = $val;  for ($i = 0; $i < sizeof($ra); $i++) {  $pattern = '/';  for ($j = 0; $j < strlen($ra[$i]); $j++) {  if ($j > 0) {  $pattern .= '(';  $pattern .= '(&#[xX]0{0,8}([9ab]);)';  $pattern .= '|';  $pattern .= '|(&#0{0,8}([9|10|13]);)';  $pattern .= ')\*';  }  $pattern .= $ra[$i][$j];  }  $pattern .= '/i';  $replacement = substr($ra[$i], 0, 2).'<x>'.substr($ra[$i], 2); // add in <> to nerf the tag  $val = preg\_replace($pattern, $replacement, $val); // filter out the hex tags  if ($val\_before == $val) {  // no replacements were made, so exit the loop  $found = false;  }  }  }  return $val;  }  //测试一下效果  //echo RemoveXSS("<script language='javascript'>alert('hello world');</script>") ;  ?> |

1. asp程序网站出现SQL注入 跨站脚本攻击（XSS）漏洞的时候可以使用

|  |
| --- |
| 源码如下：url=trim(request("url"))  修改如下：url=Server.HTMLEncode(trim(request("url")))  使用Server.HTMLEncode它来解决脚本跨站攻击漏洞  或者：  对用户输入区域只允许适应应用功能所必需的规定字符输入，,对于合法的“>”、“<”“&”这三个符号进行替换：  replace(str,"<","&#x3C;")  replace(str,">","&#x3E;")  replace(str,"&","&#x26;") |

**注意事项：**暂无

* + 1. FLASH跨站脚本攻击

**漏洞描述：在**Flash中可以嵌入ActionScript，ActionScript是一种非常强大和灵活的脚本，通过构造恶意的ActionScript脚本，可获取用户的敏感信息或者诱骗用户打开具有网页木马的URL页面。

**测试方法：**

1. 构造含有xss代码的flash文件，如一个常见的flash xss可以这样写：

getURL(“javascript:alert(document.cookie)”)

1. 将flash文件插入到HTML页面中，如：<embed src="helloworld.swf" />
2. 使用SWFIntruder工具检测产生在flash中的xss。

flash xss利用总结请参考https://www.secpulse.com/archives/44299.html。

**风险分析：**攻击者利用该漏洞可实施窃取用户信息、钓鱼等攻击。

**风险等级：**

【**高危**】：手工测试浏览器弹窗

**修复方案：**

1. 尽可能地禁止用户能够上传或加载自定义的flash文件；
2. 通过定义allowScriptAccess参数限制flash动态脚本与HTML页面的通信，该参数有三个可选值，建议设置为never：

always，对与HTML的通信也就是执行javascript不做任何限制；

sameDomain，只允许来自于本域的Flash与Html通信，这是默认值；如果使用默认值的话，应确保flash文件不是用户上传来的；

never，绝对禁止flash与页面通信。

1. 通过定义allowNetworking参数限制flash与外部网络进行通信，该参数有三个可选值，建议设置为none或internal：

all，允许使用所有的网络通信，也是默认值；

internal，flash不能与浏览器通信如navigateToURL，但是可以调用其他的API；

none，禁止任何的网络通信。

**注意事项：**暂无

* + 1. HTTP响应分割

**漏洞描述：**HTTP响应头拆分攻击是一种新型的Web攻击方式，它产生了很多安全漏洞，如Web缓存感染、用户信息涂改、窃取敏感用户页面、跨站脚本漏洞等。这种攻击方式与其衍生的一系列技术的产生，是由于Web应用程序未对用户提交的数据进行严格过滤和检查，导致攻击者可以提交一些恶意字符，如对用户输入的CR 和LF字符没有进行严格的过滤。

**测试方法：**

CR和LF两个字符的编码如下：

CR = %0d = \r

LF = %0a = \n

下面以java为例分析HTTP响应头拆分攻击的原理，如jsp页面(/test.jsp)，内容如下：

<%

response.sendRedirect("index.jsp?lang="+request.getParameter("lang"));

%>

定义lang=en，访问test.jsp，服务器返回302，重定向到另一个地址，HTTP响应内容如下：

|  |
| --- |
| HTTP/1.1 302 Moved Temporarily\r\n  Date: Wed, 1 Mar 2005 12:53:28 GMT\r\n  Location: http://192.168.0.1/index.jsp?lang=en\r\n  Content-Type: text/html\r\n  Connection: Close\r\n  <html><head><title>302 Moved Temporarily</title></head>\r\n  <body>\r\n  <p>This document you requested has moved temporarily.</p>\r\n  </body></html>\r\n |

lang所赋的值被嵌入在Location响应头中。如果服务器未限制lang参数或者校验，攻击者可以修改en参数来迫使服务器返回两个响应，将lang赋值成以下内容：

|  |
| --- |
| en%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2024%0d%0a%0d%0a<html>Test page</html> |

再次访问test.jsp页面时，HTTP响应内容如下：

|  |
| --- |
| HTTP/1.1 302 Moved Temporarily\r\n  Date: Wed, 1 Mar 2005 15:26:41 GMT\r\n  Location: http://192.168.0.1/index.jsp?lang=en\r\n  Content-Length: 0\r\n  HTTP/1.1 200 OK\r\n  Content-Type: text/html\r\n  Content-Length: 24\r\n  <html>Test page</html>\r\n  Content-Type: text/html\r\n  Connection: Close\r\n  <html><head><title>302 Moved Temporarily</title></head>\r\n  <body bgcolor="#FFFFFF">\r\n  <p>This document you requested has moved temporarily.</p>\r\n  </body></html>\r\n |

标红部分是lang参数的值，从上述HTTP响应中可以看出，该响应包含了两条HTTP响应，一个是302，一个是200，请求test.jsp页面，浏览器匹配HTTP/1.1 302响应。

浏览器接收到HTTP响应后解析到Content-Length: 0，认为响应内容已经结束，然后请求重定向URL地址http://192.168.0.1/index.jsp?lang=en，HTTP请求内容如下：

|  |
| --- |
| GET http://192.168.0.1/index.jsp?lang=en HTTP/1.1  Host: 192.168.0.1  User-Agent: Mozilla/5.0 (Windows NT 6.2; rv:20.0) Gecko/20100101 Firefox/20.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Referer: http://192.168.0.1/index.jsp  Connection: keep-alive |

针对上述的HTTP请求，浏览器会自动匹配之前收到的第二个响应HTTP/1.1 200，攻击者利用此漏洞可控制第二条响应的内容，从而达到欺骗用户的目的。

**风险分析：**HTTP响应头拆分攻击本质是：攻击者可以发送一个或几个HTTP指令使服务器产生攻击者想得到的输出。攻击者构造的HTTP指令使服务器误把几条HTTP请求看成一条完整的HTTP请求来解释。如果攻击者可以控制第一条请求的部分内容，而且完全控制着第二条及后面的HTTP请求，即从HTTP状态行一直到HTTP请求的尾部，攻击者会发送多个HTTP请求指令到目标系统，例如第一条请求指令使得服务器完全接受两个HTTP请求，第二条请求通常是在服务器上请求一些非法资源，而服务器将会自动匹配到第二条请求的响应，并输出攻击者想要请求的资源，从而达到攻击目的。

**风险等级：**

【**中危**】：可成功实现 XSS，会话固定

**修复方案：**

1. 限制用户输入的CR和LF，对CR和LF字符正确编码后再输出，以防止攻击者注入自定义的HTTP头；
2. 限制输入URL的长度。

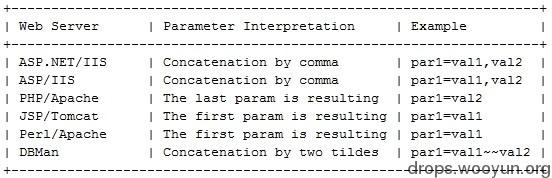
**注意事项：**暂无

* + 1. HTTP参数污染

**漏洞描述：**使用同名参数对原参数进行污染，应用在检测时只检测第一个参数是否符号要求，因此可利用同名参数构造攻击脚本，参数污染通常用于sql注入绕过WAF检测。

**测试方法：**在请求中添加同名参数，并在添加的参数中输入恶意构造的语句。例如：http://www.xx.com/index.php?id=1&id=2%20union%20select%201,2,3,current\_user，利用该漏洞进行sql注入绕过可参考80sec发布的《浅谈绕过waf的数种方法》。

**风险分析：**不同Web服务器对同名参数的处理方式如下图所示：



攻击者可根据其特性对参数进行覆盖，绕过应用检测。

**风险等级：**

【**高危**】：同名参数绕过 filter 或其他规则

**修复方案：**应用程序应正确过滤用户输入（过滤所有参数），以防止此漏洞。

**注意事项：**暂无

* + 1. Host头攻击

**漏洞描述：**Web应用程序获取网站域名一般是依赖HTTP Host header（比如在JSP里通过request.getHeader()获取），这里的header很多情况下是不可靠的。攻击者恶意利用HTTP Host header会导致HTTP Host头攻击发生。

**测试方法：**

1. 密码重置污染攻击，点击重置密码的链接时，url::abs\_site 这一部分使用的Host header是来自用户重置密码的请求，那么可以通过一个受他控制的链接来污染密码重置的邮件，例如替换host：当然这种攻击方式一定要能骗取用户点击访问这个受污染的链接，如果用户警觉了没有点击，那么攻击就会失败

|  |
| --- |
| > POST /password/reset HTTP/1.1  > Host: evil.com  > ...  > csrf=1e8d5c9bceb16667b1b330cc5fd48663&name=admin |

1. 通过Host header来污染缓存的攻击方法:

因此为了能使缓存能将污染后的response返回给用户，我们还必须让缓存服务器看到的host header 和应用看到的host header 不一样, 比如说对于Varnish（一个很有名的缓存服务软件），可以使用一个复制的Host header。Varnish是通过最先到达的请求的host header来辨别host的，而Apache则是看所有请求的host，Nginx则只是看最后一个请求的host。这就意味着你可以通过下面这个请求来欺骗Varnish达到污染的目的

|  |
| --- |
| > GET / HTTP/1.1  > Host: example.com  > Host: evil.com |

1. 利用web漏洞扫描工具进行检测。

**风险分析：**大部分的Web应用程序未经html编码直接把header输出到了页面中。比如：<link href=http://request.getHeader(“Host”) >。有些URL中还包含有secret key和token，如：<a href="http://request.getHeader(“Host”)?token=topsecret">。这样处理header一般会产生两种常见的攻击：缓存污染和密码重置。缓存污染是指攻击者通过控制一个缓存系统来将一个恶意站点的页面返回给用户。密码重置这种攻击主要是因为发送给用户的内容是被污染的，即间接的劫持邮件。

**风险等级：**

【**中危**】：请求数据包包中 host 值直接拼接到生成链接中。

**修复方案：**

* + 1. 服务器方面：
  1. 由于http请求的特点，host header的值其实是不可信的。唯一可信的只有SERVER\_NAME，这个在Apache和Nginx里可以通过设置一个虚拟机来记录所有的非法host header。在Nginx里还可以通过指定一个SERVER\_NAME名单，Apache也可以通过指定一个SERVER\_NAME名单并开启UseCanonicalName选项。建议两种方法同时使用。
  2. Varnish很快会发布一个补丁。在官方补丁出来前，可以通过在配置文件里加入：

|  |
| --- |
| import std;  sub vcl\_recv {  std.collect(req.http.host);  } |

2、应用方面：

在网站安装和初始化的时候，要求管理员提供一个可信任的域名白名单。如果这个实现起来比较困难，那至少也要保证使用使用getServerName()代替getHeader(“Host”)。

**注意事项：**暂无

* + 1. SQL注入

**漏洞描述：**所谓SQL注入，就是通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。具体来说，它是利用现有应用程序，将（恶意）SQL命令注入到后台数据库引擎执行的能力，它可以通过在Web表单中输入（恶意）SQL语句得到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行SQL语句。 造成SQL注入漏洞原因有两个：一个是没有对输入的数据进行过滤（过滤输入），还有一个是没有对发送到数据库的数据进行转义（转义输出）。

**测试方法：**

1. 通过web漏洞扫描工具进行对网站爬虫后得到的所有链接进行检测，或者手工判断是否存在注入点，一旦确认存在漏洞，可利用自动化工具sqlmap去尝试注入。几种常见的判断方法：
2. 数字型。

http://host/test.php?id=100 and 1=1 返回成功

http://host/test.php?id=100 and 1=2 返回失败

1. 字符型。

http://host/test.php?name=rainman ’ and ‘1’=‘1 返回成功

http://host/test.php?name=rainman ’ and ‘1’=‘2 返回失败

1. 搜索型。

搜索型注入：简单的判断搜索型注入漏洞是否存在的办法是：

* 先搜索（'），如果出错，说明90%存在这个漏洞。
* 然后搜索（%），如果正常返回，说明95%有洞了。
* 然后再搜索一个关键字，比如（2006）吧，正常返回所有2006相关的信息。
* 再搜索（2006%'and 1=1 and '%'='）和（2006%'and 1=2 and '%'='）

1. 绕过验证（常见的为管理登陆）也称万能密码

(1) 用户名输入： ‘ or 1=1 or ‘ 密码：任意

(2)Admin’ - -（或‘ or 1=1 or ‘ - -）(admin or 1=1 --) (MS SQL)(直接输入用户名，不进行密码验证)

(3)用户名输入：admin 密码输入：’ or ‘1’=’1 也可以

(4) 用户名输入：admin' or 'a'='a 密码输入：任意

(5) 用户名输入：‘ or 1=1 - -

(6) 用户名输入：admin‘ or 1=1 - - 密码输入：任意

(7) 用户名输入：1'or'1'='1'or'1'='1 密码输入：任意

1. 不同的SQL服务器连结字符串的语法不同，比如MS SQL Server使用符号+来连结字符串，而Oracle使用符号||来连结：

http://host/test.jsp?ProdName=Book’ 返回错误

http://host/test.jsp?ProdName=B’+’ook 返回正常

http://host/test.jsp?ProdName=B’||’ook 返回正常说明有SQL注入

1. 如果应用程序已经过滤了’和+等特殊字符，我们仍然可以在输入时过把字符转换成URL编码（即字符ASCII码的16进制）来绕过检查。

**风险分析：**在输入URL和表单处，攻击者通过输入精心构造的SQL语句，对数据库记录进行增删改查，或直接获取服务器权限。攻击者实施SQL注入攻击时大多借助自动化注入工具，如穿山甲、明小子、sqlmap、Havij等。

**风险等级：**

【**高危**】：存在SQL注入，无论能否爆出数据

**修复方案：**SQL注入的主要原因是程序没有严格过滤用户输入的数据，导致非法数据侵入系统。

1. 对用户输入的特殊字符进行严格过滤，如’、”、<、>、/、\*、;、+、-、&、|、(、)、and、or、select、union。
2. 使用参数化查询（PreparedStatement），避免将未经过滤的输入直接拼接到SQL查询语句中。
3. Web应用中用于连接数据库的用户与数据库的系统管理员用户的权限有严格的区分（如不能执行drop等），并设置Web应用中用于连接数据库的用户不允许操作其他数据库。
4. 设置Web应用中用于连接数据库的用户对Web目录不允许有写权限。
5. 使用Web应用防火墙。

以下给出sql注入防范编码供开发者参考：

方法一：参数化查询，利用PreparedStatement对象的set方法给参数赋值。参数化查询强制要求给每个参数定义类型，这种方式使得数据库能够区分出哪些属于代码段哪些属于数据段。

|  |
| --- |
| String custname = request.getParameter("customerName");  String query = "SELECT account\_balance FROM user\_data WHERE user\_name = ? ";  PreparedStatement pstmt = connection.prepareStatement( query );  pstmt.setString( 1, custname);  ResultSet results = pstmt.executeQuery( ); |

方法二：输入合法性验证。检查输入字符串中是否包含敏感的SQL字符，是否包含SQL关键字、是否是典型的SQL注入语句。检查到非法字符之后，可以直接结束数据库查询并返回告警，也可以把非法字符替换为空或进行其他形式的修正。需要注意的是这种防御方式的可靠性建立在目前情况下攻击者无法构造绕过合法性验证的恶意SQL语句。

通用的SQL语句合法性验证如下：

|  |
| --- |
| public static boolean sql\_inj(String str)  {  String inj\_str = "'|and|exec|insert|select|delete|update|  count|\*|%|chr|mid|master|truncate|char|declare|;|or|-|+|,";  String inj\_stra[] = split(inj\_str,"|");  for (int i=0 ; i < inj\_stra.length ; i++ )  {  if (str.indexOf(inj\_stra)>=0)  {  return true;  }  }  return false;  } |

方法三：使用OWASP提供的ESAPI，ESAPI (OWASP企业安全应用程序接口)是一个免费、开源的、网页应用程序安全控件库，它使程序员能够更容易写出更低风险的程序。ESAPI接口库被设计来使程序员能够更容易的在现有的程序中引入安全因素。ESAPI库也可以成为作为新程序开发的基础。ESAPI详细介绍请参考：http://www.owasp.org.cn/owasp-project/ESAPI/。

**注意事项：**暂无

* + 1. NoSQL注入

**漏洞描述：** 攻击者伪造一个带有注入代码的Web访问请求，当数据库客户端或协议包装器进行处理时，将会执行预期的非法数据库操作。如下图所示：



**测试方法：**NoSQL相关的攻击大致分为以下五类：

* **重言式**。又称为永真式。此类攻击是在条件语句中注入代码，使生成的表达式判定结果永远为真，从而绕过认证或访问机制。例如，MongoDB注入攻击，用$ne（不相等）操作符, 构造请求http://www.xxx.com/index.php?username[$ne]=1&password[$ne]=1，绕过认证进入系统。
* **联合查询**。联合查询最常用的用法是绕过认证页面获取数据。通过增加永真的表达式利用布尔OR运算符进行攻击，从而导致整个语句判定出错，进行非法的数据获取。
* **JavaScript注入**。这是一种新的漏洞，由允许执行数据内容中JavaScript的NoSQL数据库引入的。JavaScript使在数据引擎进行复杂事务和查询成为可能。传递不干净的用户输入到这些查询中可以注入任意JavaScript代码，这会导致非法的数据获取或篡改。
* **背负式查询**。在背负式查询中，攻击者通过利用转义特定字符（比如像回车和换行之类的结束符）插入由数据库额外执行的查询，这样就可以执行任意代码了。
* **跨域违规**。HTTP REST APIs是NoSQL数据库中的一个流行模块，然而，它们引入了一类新的漏洞，它甚至能让攻击者从其他域攻击数据库。在跨域攻击中，攻击者利用合法用户和他们的网页浏览器执行有害的操作。在本文中，我们将展示此类跨站请求伪造（CSRF）攻击形式的违规行为，在此网站信任的用户浏览器将被利用在NoSQL数据库上执行非法操作。通过把HTML格式的代码注入到有漏洞的网站或者欺骗用户进入到攻击者自己的网站上，攻击者可以在目标数据库上执行post动作，从而破坏数据库。

案例详情请参考：http://www.infoq.com/cn/articles/nosql-injections-analysis。

**风险分析：**在输入URL和表单处，攻击者通过输入精心构造的语句，获取数据库内容，或绕过认证。

**风险等级：**

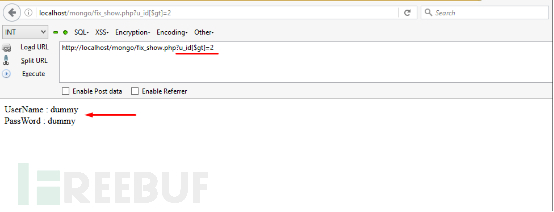
【**高危**】：存在NoSQL注入，无论是否能够爆出数据

**修复方案：**防止数组中的运算操作即可防御该漏洞。可以使用一下两种方式：

* + - 1. 采用implode()方法，如下图：



implode()函数返回由数组元素组合成的字符串。这样就只能得到一个对应的结果。



* + - 1. 使用addslashes()函数，这样攻击者便不能破坏查询语句。同时，也可以用正则表达式把一些特殊符号替换掉，如使用如下正则：

$u\_name =preg\_replace(‘/[^a-z0-9]/i’, ‘\’, $\_GET[‘u\_name’]);



**注意事项：**暂无

* + 1. LDAP注入

**漏洞描述：**LDAP(Lightweight Directory Access Protocol)，轻量级目录访问协议，是一种在线目录访问协议，主要用于目录中资源的搜索和查询。如果在用户可控制的输入中没有对 LDAP 语法进行除去或引用，那么生成的 LDAP 查询可能会导致将这些输入解释为 LDAP 而不是普通用户数据。这可用于修改查询逻辑以绕过安全性检查，或者插入其他用于修改后端数据库的语句，可能包括执行系统命令。

**测试方法：**

可参考http://www.moonsec.com/post-296.html。

**风险分析：**利用LDAP注入技术的关键在于控制用于目录搜索服务的过滤器。使用这些技术，攻击者可能直接访问LDAP目录树下的数据库，及重要的公司信息。

**风险等级：**

【**高危**】

**修复方案：**

1. 使用不允许此弱点出现的经过审核的库或框架，或提供更容易避免此弱点的构造。
2. 使用自动实施数据和代码之间的分离的结构化机制。这些机制也许能够自动提供相关引用、编码和验证，而不是依赖于开发者在生成输出的每一处提供此能力。
3. 使用完成必要任务所需的最低特权来运行代码。
4. 如果在有风险的情况下仍需要使用动态生成的查询字符串或命令，建议对参数正确地加引号并将这些参数中的任何特殊字符转义。
5. 使用“接受已知善意”输入验证策略：严格遵守规范的可接受输入的白名单。拒绝没有严格遵守规范的任何输入，或者将其变换为严格遵守规范的内容。不要完全依赖于通过黑名单检测恶意或格式错误的输入。但是，黑名单可帮助检测潜在攻击，或者确定哪些输入由于格式严重错误而应直接拒绝。

**注意事项：**暂无

* + 1. XML注入

**漏洞描述：**可扩展标记语言 (Extensible Markup Language, XML) ，用于标记电子文件使其具有结构性的标记语言，可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。 XML是标准通用标记语言 (SGML) 的子集，非常适合 Web 传输。XML 提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。发现目前一些普遍使用xml的场景中都存在一种古老的XML实体注入漏洞，这可能导致较为严重的安全问题，使得攻击者可能可以任意访问服务器以及应用所在网络的任何资源。

1. **测试方法：**通过手工篡改网站中xml实体中的头部，加入相关的读取文件或者是链接，或者是命令执行等，如file:///path/to/file.ext；http://url/file.ext；php://filter/read=convert.base64-encode/resource=conf.php，类似如下代码所示：

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <!DOCTYPE nsfocus-sec [  <!ELEMENT methodname ANY >  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>  <methodcall>  <methodname>&xxe;</methodname>  </methodcall> |

篡改以后，如果可读取file文件或者达到植入命令的效果，则说明存在该漏洞。

**风险分析：**XML注入攻击广泛用于Web系统的攻击。攻击方式主要有以下几种：

* 1. 使用超长标签像<AAAAAAA/> ，多达1024个A，或者使用超多的属性像<AA a=‘a’b=‘b’…> ，由于XML文件本身标签所占用的资源通常比内容多，在数据和属性比较多时，XML文件将变得非常庞大，给XML文件的传输和解析带来困难；
  2. 递归负载攻击，注入深层次的循环嵌套的XML数据，造成服务器上XML分析器崩溃，造成系统计算资源耗尽而被Dos ；
  3. 外部实体攻击，利用<!Entityname SYSTEM “URI”>。XML文件的解析依赖libxml库，而libxml2.9以前的版本默认支持并开启了外部实体的引用，服务端解析用户提交的XML文件时，未对XML文件引用的外部实体（含外部普通实体和外部参数实体）做合适的处理，并且实体的URL支持file://和ftp://等协议，攻击者可以在XML文件中声明URI指向服务器本地的实体造成攻击。实体攻击可导致信息泄露、任意文件读取、DOS攻击和代码执行等问题。

**风险等级：**

【**高危**】：额外数据字段注入导致权限提升或原始数据修改

【**中危**】：批量数据提交并成功被解析

**修复方案：**

1. 严格检查用户输入的字符；
2. 检查使用的底层XML解析库，使用JAVA语言提供的禁用外部实体的方法：DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();dbf.setExpandEntityReferences(false)；
3. 操作XML时对格式字符进行转义处理，常见的格式字符如下表：

|  |  |
| --- | --- |
| &lt； | < |
| &gt; | > |
| &amp; | & |
| &apos; | ‘ |
| &quot; | “ |

**注意事项：**暂无

* + 1. XXE

**漏洞描述：**外部实体攻击，利用<!Entityname SYSTEM “URI”>。XML文件的解析依赖libxml库，而libxml2.9以前的版本默认支持并开启了外部实体的引用，服务端解析用户提交的XML文件时，未对XML文件引用的外部实体（含外部普通实体和外部参数实体）做合适的处理，并且实体的URL支持file://和ftp://等协议。

**测试方法：**通过手工篡改网站中xml实体中的头部，加入相关的读取文件或者是链接，或者是命令执行等，如file:///path/to/file.ext；http://url/file.ext；php://filter/read=convert.base64-encode/resource=conf.php，类似如下代码所示：

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <!DOCTYPE nsfocus-sec [  <!ELEMENT methodname ANY >  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>  <methodcall>  <methodname>&xxe;</methodname>  </methodcall> |

**风险分析：**攻击者可以在XML文件中声明URI指向服务器本地的实体造成攻击。实体攻击可导致信息泄露、任意文件读取、DOS攻击和代码执行等问题。

**风险等级：**

【**高危**】：实体注入可读取服务器文件或执行命令

**修复方案：**

1. 严格检查用户输入的字符；
2. 检查使用的底层XML解析库，使用JAVA语言提供的禁用外部实体的方法：DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();dbf.setExpandEntityReferences(false)；
3. 操作XML时对格式字符进行转义处理，常见的格式字符如下表：

|  |  |
| --- | --- |
| &lt； | < |
| &gt; | > |
| &amp; | & |
| &apos; | ‘ |
| &quot; | “ |

**注意事项：**暂无

* + 1. XPATH注入

**漏洞描述：**XPath注入攻击是指利用XPath 解析器的松散输入和容错特性，能够在URL、表单或其它信息上附带恶意的XPath查询代码，以获得权限信息的访问权并更改这些信息。XPath注入攻击是针对Web服务应用新的攻击方法，它允许攻击者在事先不知道XPath查询相关知识的情况下，通过XPath查询得到一个XML文档的完整内容。

**测试方法：**XPath注入攻击主要是通过构建特殊的输入，这些输入往往是XPath语法中的一些组合，这些输入将作为参数传入Web 应用程序，通过执行XPath查询而执行入侵者想要的操作，下面以登录验证中的模块为例，说明XPath注入攻击的实现原理和检测方法，在Web应用程序的登录验证程序中，一般有用户名（username）和密码（password）两个参数，程序会通过用户所提交输入的用户名和密码来执行授权操作。若验证数据存放在XML文件中，其原理是通过查找user表中的用户名（username）和密码（password）的结果来进行授权访问，例存在user.xml文件如下：

|  |
| --- |
| <users>  <user>  <firstname>Ben</firstname>  <lastname>Elmore</lastname>  <loginID>abc</loginID>  <password>test123</password>  </user>  <user>  <firstname>Shlomy</firstname>  <lastname>Gantz</lastname>  <loginID>xyz</loginID>  <password>123test</password>  </user> |

则在XPath中其典型的查询语句如下：

//users/user[loginID/text()='xyz' and password/text()='123test']

但是，可以采用如下的方法实施注入攻击，绕过身份验证。如果用户传入一个login和password，例如loginID = 'xyz' 和password = '123test'，则该查询语句将返回true。但如果用户传入类似'' or 1=1 or ''=''的值，那么该查询语句也会得到true 返回值，因为XPath查询语句最终会变成如下代码：

//users/user[loginID/text()='' or 1=1 or ''='' and password/text()='' or 1=1 or ''='']

这个字符串会在逻辑上使查询一直返回true 并将一直允许攻击者访问系统。攻击者可以利用XPath在应用程序中动态地操作XML文档。攻击完成登录可以再通过XPath盲注技术获取最高权限帐号和其它重要文档信息。

**风险分析：**XPath 注入攻击利用两种技术，即XPath扫描和XPath查询布尔化。通过该攻击，攻击者可以控制用来进行XPath查询的XML数据库。这种攻击可以有效地对付使用XPath查询（和XML数据库）来执行身份验证、查找或者其它操作。XPath注入攻击同SQL注入攻击类似，但和SQL注入攻击相比较，XPath在以下方面具有优势。

（1）广泛性。XPath注入攻击利用的是XPath语法，由于XPath是一种标准语言，因此只要是利用XPath语法的Web应用程序如果未对输入的XPath查询做严格的处理都会存在XPath注入漏洞，所以可能在所有的XPath实现中都包含有该弱点，这和SQL注入攻击有很大区别。在SQL注入攻击过程中根据数据库支持的SQL语言不同，注入攻击的实现可能不同。

（2）危害性大。XPath语言几乎可以引用XML文档的所有部分，而这样的引用一般没有访问控制限制。但在SQL注入攻击中，一个“用户”的权限可能被 限制到某一特定的表、列或者查询，而XPath注入攻击可以保证得到完整的XML文档，即完整的数据库。只要Web服务应用具有基本的安全漏洞，即可构造针对XPath应用的自动攻击。

**风险等级：**

【**高危**】：通过构造Xpath查询语句进行注入，操作XML数据。

**修复方案：**目前专门的 XPath 攻击防御技术还不是太多，但是 SQL 注入攻击防御技术 可以加以改进，应用到 XPath 注入攻击防御。具体技术总结如下：

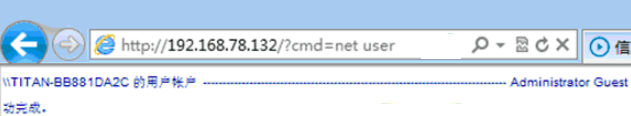
1. 数据提交到服务器上端，在服务端正式处理这批数据之前，对提交数据的合法性进行验证。
2. 检查提交的数据是否包含特殊字符，对特殊字符进行编码转换或替换、删除敏感字符或字符串。
3. 对于系统出现的错误信息，以IE错误编码信息替换，屏蔽系统本身的出错信息。
4. 参数化XPath查询，将需要构建的XPath查询表达式，以变量的形式表示，变量不是可以执行的脚本。如下代码可以通过创建保存查询的外部文件使查询参数化：declare variable $loginID as xs：string external； declare variable $password as xs：string external； //users/user[@loginID=$loginID and @password= $password]。
5. 通过MD5、SSL等加密算法， 对于数据敏感信息和在数据传输过程中加密， 即使某些非法用户通过非法手法获取数据包，看到的也是加密后的信息。

**注意事项：**暂无

* + 1. 命令注入

**漏洞描述：**Command Injection，即命令注入攻击，是指由于Web应用程序对用户提交的数据过滤不严格，导致黑客可以通过构造特殊命令字符串的方式，将数据提交至Web应用程序中，并利用该方式执行外部程序或系统命令实施攻击，非法获取数据或者网络资源等。在命令注入的漏洞中，最为常见的是PHP的命令注入。PHP命令注入攻击存在的主要原因是Web应用程序员在应用PHP语言中一些具有命令执行功能的函数时，对用户提交的数据内容没有进行严格的过滤就带入函数中执行而造成的。例如，当黑客提交的数据内容为向网站目录写入PHP文件时，就可以通过该命令注入攻击漏洞写入一个PHP后门文件，进而实施下一步渗透攻击。

**测试方法：**通过web扫描工具进行扫描，也可以通过手工进行验证：打开网站，在URL中传入命令的参数输入所要执行的命令，如下图所示的http://192.168.1.3/?cmd=net user，可以发现命令能够成功执行：



**风险分析：**Web应用程序没有过滤类似system()，eval()，exec()等函数是恶意命令执行的最主要原因。

**风险等级：**

【**高危**】：成功执行系统命令或脚本命令

**修复方案：**PHP中命令注入攻击漏洞带来的危害和影响很严重。防范命令注入攻击漏洞的存在可以通过以下几种方法：

1. 尽量不去执行外部的应用程序或命令。
2. 使用自定义函数或函数库实现外部应用程序或命令的功能。
3. 在执行system、eval等命令执行功能的函数前，校验参数内容。
4. 使用escapeshellarg函数处理相关参数。Escapeshellarg函数会将任何引起参数或命令结束的字符进行转义，如单引号“’”会被转义为“\’”，双引号“””会被转义为“\””，分号“;”会被转义为“\;”，这样escapeshellarg会将参数内容限制在一对单引号或双引号里面，转义参数中所包含的单引号或双引号，使其无法对当前执行进行截断，实现防范命令注入攻击的目的。
5. 使用safe\_mode\_exec\_dir执行可执行的文件路径。将php.ini文件中的safe\_mode设置为On，然后将允许执行的文件放入一个目录中，并使用safe\_mode\_exec\_dir指定这个可执行的文件路径。在需要执行相应的外部程序时，程序必须在safe\_mode\_exec\_dir指定的目录中才会允许执行，否则执行将失败。

**注意事项：**暂无

* + 1. 任意文件上传

**漏洞描述：**一般情况下文件上传漏洞是指用户上传了一个可执行的脚本文件，并通过此脚本文件获得了执行服务器端命令的能力。文件上传本身是web中最为常见的一种功能需求，关键是文件上传之后服务器端的处理、解释文件的过程是否安全。一般的情况有：

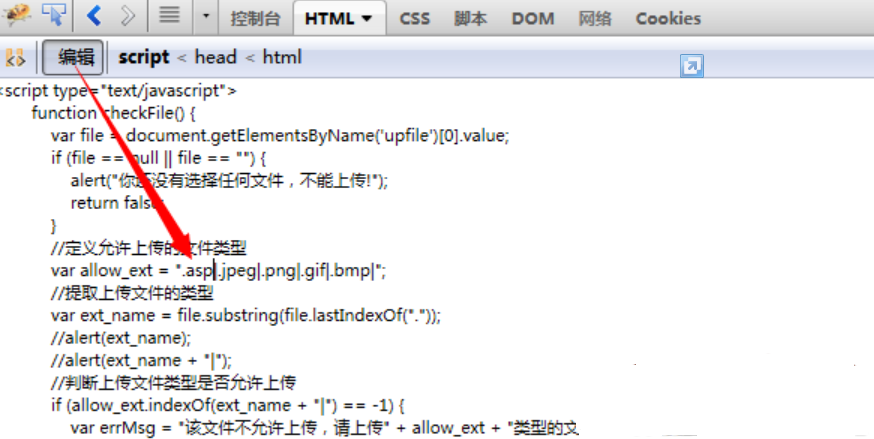
1. 上传文件WEB脚本语言，服务器的WEB容器解释并执行了用户上传的脚本，导致代码执行；
2. 上传文件FLASH策略文件crossdomain.xml，以此来控制Flash在该域下的行为；
3. 上传文件是病毒、木马文件，攻击者用以诱骗用户或管理员下载执行；
4. 上传文件是钓鱼图片或包含了脚本的图片，某些浏览器会作为脚本执行，可用于实施钓鱼或欺诈；

**测试方法：**上传方式根据不同的web语言，检测方法也各式各样，以下列举基于JS验证的上传的几种常见的文件上传绕过方法：

1. 直接删除代码中onsubmit事件中关于文件上传时验证上传文件的相关代码即可。如图：



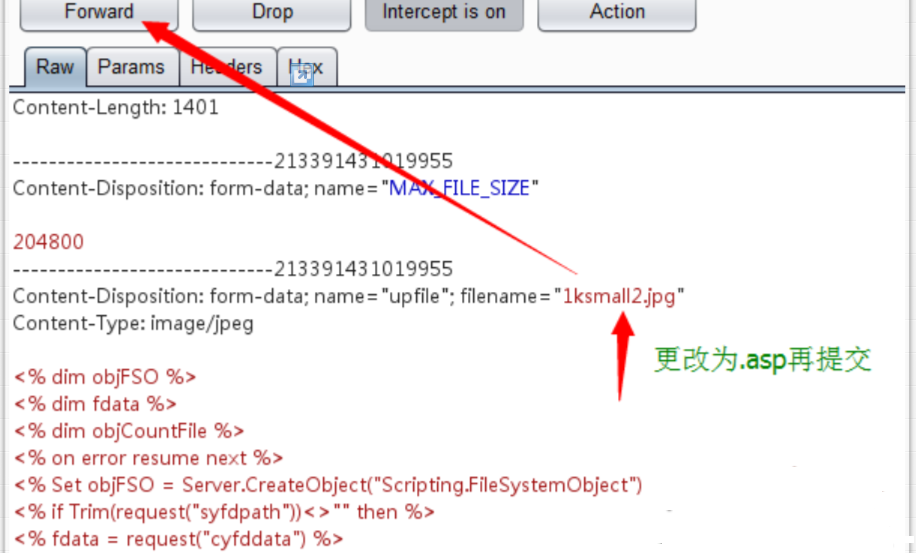
1. 直接更改文件上传JS代码中允许上传的文件扩展名你想要上传的文件扩展名，如图所示：



1. 使用本地提交表单即可，如图所示：



1. 使用burpsuite或者是fiddle等代理工具提交，本地文件先更改为jpg，上传时拦截，再把文件扩展名更改为asp即可，如图所示：



1. 当然也有不是基于JS验证的上传，例如一些中间件IIS，Nginx,，PHP,FCK编辑器等等的解析漏洞，其上传绕过方式也是多种多样。通过对上传页面进行检查，常见的文件上传检查针对文件类型进行，可以使用手动修改POST包然后添加%00字节用于截断某些函数对文件名的判断。除了修改文件名来绕过类型检查外，还可以修改文件头来伪造文件头，欺骗文件上传检查，如图，修改文件头中的类型来进行绕过：



以上为几种常见的上传，更多的还需自行研究，进行上传绕过。以下为总体的测试流程：

1. 登陆网站，并打开文件上传页面
2. 点击“浏览”按钮，并选择本地的一个JSP文件（比如hacker.jsp），确认上传。
3. 如果客户端脚本限制了上传文件的类型（比如允许gif文件），则把hacker.jsp更名为hacker.gif；配置HTTP Proxy（burp）进行http请求拦截；重新点击“浏览”按钮，并选择hacker.gift，确认上传。
4. 在WebScarab拦截的HTTP请求数据中，将hacker.gif修改为hacker.jsp，再发送请求数据。
5. 登陆后台服务器，用命令find / -name hacker.jsp查看hacker.jsp文件存放的路径。如果可以直接以Web方式访问，则构造访问的URL，并通过浏览器访问hacker.jsp，如果可以正常访问，则已经取得WebShell，测试结束。如果hacker.jsp无法通过web方式访问，例如hacker.jsp存放在/home/tmp/目录下，而/home/tomcat/webapps目录对应http://www.example.com/，则进行下一步
6. 重复1～3，在burp拦截的HTTP请求数据中，将hacker.gif修改为../tomcat/webapps/hacker.jsp，再发送请求数据。
7. 在浏览器地址栏输入http://www.example.com/hacker.jsp，访问该后门程序，取得WebShell，结束检测。

**风险分析：**攻击者直接上传webshell到服务器，从而远程控制服务器。

**风险等级：**

【**高危**】：成功上传恶意脚本文件，并解析webshell。

**修复方案：**

1. 最有效的，将文件上传目录直接设置为不可执行，对于Linux而言，撤销其目录的'x'权限；实际中很多大型网站的上传应用都会放置在独立的存储上作为静态文件处理，一是方便使用缓存加速降低能耗，二是杜绝了脚本执行的可能性；
2. 文件类型检查：建议使用白名单方式，结合MIME Type、后缀检查等方式（即只允许允许的文件类型进行上传）；此外对于图片的处理可以使用压缩函数或resize函数，处理图片的同时破坏其包含的HTML代码；
3. 使用随机数改写文件名和文件路径，使得用户不能轻易访问自己上传的文件；
4. 单独设置文件服务器的域名。

**注意事项：**暂无

* + 1. 反序列化漏洞

**漏洞描述：**Java序列化就是把对象转换成字节流，便于保存在内存、文件、数据库中，Java中的ObjectOutputStream类的writeObject()方法可以实现序列化。Java反序列化即逆过程，由字节流还原成对象。ObjectInputStream类的readObject()方法用于反序列化。Apache Commons Collections允许链式的任意的类函数反射调用。攻击者通过允许Java序列化协议的端口，把攻击代码上传到服务器上，再由Apache Commons Collections里的TransformedMap来执行。

**测试方法：**在java编写的web应用与web服务器间java通常会发送大量的序列化对象例，如以下场景：

1. HTTP请求中的参数，cookies以及Parameters。

2. RMI协议，被广泛使用的RMI协议完全基于序列化

3. JMX 同样用于处理序列化对象

自定义协议 用来接收与发送原始的java对象

常见中间件，如JBoss、Weblogic、WebSphere、jenkins等存在java反序列化漏洞，可利用工具进行测试，如DeserializeExploit.jar、WebLogic\_EXP.jar、attackRMI.jar。

**风险分析：** java反序列化漏洞存在于各大业务系统中，影响范围大，且自动化利用工具比较成熟，攻击成本较低。

**风险等级：**

【**高危**】：成功执行系统命令或脚本命令

**修复方案：**Apache官方发布了commons-collections的新版本，下载地址：   
http://commons.apache.org/proper/commons-collections/download\_collections.cgi，修复方法为替换有漏洞的commons-collections组件。

**注意事项：**暂无

* 1. 专项漏洞

专项漏洞部分注重测试方法论，每个专项仅列举一个例子。实际测试过程中，需视情况而定。

* + 1. Web组件（SSL/WebDAV）漏洞

**漏洞描述：**Web组件漏洞泛指Web组件存在的通用漏洞。

**测试方法：**根据目标系统所使用的Web组件版本，查找其公开漏洞及其利用脚本。如目标使用了SSL协议，可测试是否存在SSL v3重协商漏洞、POODLE信息泄漏漏洞等。下面以SSL v3重协商漏洞为例进行说明：

1. 通过web漏洞扫描工具进行检测。

2、通过利用SSLciphercheck软件，通过CMD下运行，检测命令：“sslciphercheck.exe -h ip地址或者域名 -p 443”。

**风险分析：** SSL＼TSL是位于一种可靠地网络层协议TCP协议之上的一个协议 ，该协议是为了在客户端和服务器之间生成一个安全的连接，这种连接是私密的、可靠的并且通信双方可以互相验证双方的身份 。所以SSI＼TSI 协议应该具有机密性 、完整性和确定性。而对于重新协商，一个SSL协议一旦建立起来，客户端 (C )和服务器 (S)都可以再一次重新协商，也称为重新握手，这将会导致在加密的信道上进行一次握手，C 可以通过发送一个新的client Hello消息来创建一次重新协商。同样地，S可以发送一个Hello request消息，从而使C回应一个新的client Hello，以创建一个重新协商，建立重新协商的目的是更新密钥，增强所用密码组的保密性和从C到S的身份认证等等。

在中间人攻击中，攻击者会主动窃听，他将截获客户端和服务器之间的连接，并使他们认为他们是在直接通话，而实际上他们是通过攻击者的转接来通话的。要完成中间人攻击，中间人必须截获客户端和服务器间的所有通信信息，这样中间人可以选择有用的信息并且它可 以将新的信息随意的插入。

**风险等级：**

【**低危**】

**修复方案：**

1. 关闭renegotiation协议或限制SSL重协商数，以防止TLS DOS攻击：在Apache 2.2.15以上版本中，可以通过在配置文件httpd.conf中，添加如下字串：SSLInsecureRenegotiation Off；如使用的Apache版本低于2.2.15，则可以通过限制SSL重协商的次数防止DOS攻击。
2. 使用WEB应用防火墙防止中间人攻击：通过WEB应用防火墙，对HTTP Header请求头进行过滤和监控，舍弃嵌入式的HTTP请求行，以防止中间人攻击。

**注意事项：**暂无

* + 1. 中间件相关漏洞

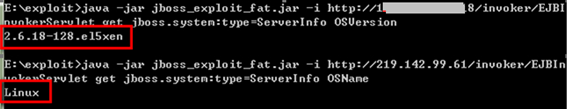
**漏洞描述：**包括Apache、Nginx、IIS、Tomcat、Weblogic、Jboss以及其他的中间件的漏洞。

**测试方法：**根据目标系统所使用的中间件版本，查找其公开漏洞及其利用脚本。如Tomcat/Weblogic/WebSphere管理控制台弱口令、java发序列化、IIS短文件名泄漏、JBoss远程代码执行等。下面以JBoss远程代码执行漏洞为例进行说明：

1. 对于采用JBOSS的网站，首先判断其版本，如果版本为1.0.x版本，则可通过对其控制台的访问来判断，访问http://127.0.0.1:8080/jmx-console/，出现以下页面，则此漏洞可被利用：



1. 或者用工具jboss\_exploit\_fat.jar来判断是否存在漏洞，如图所示：



**风险分析：**JBOSS默认配置会有一个后台漏洞，漏洞发生在jboss.deployment命名空间，中的addURL()函数,该函数可以远程下载一个war压缩包并解压。 如果压缩包里含有webshell文件，是直接可以解析的。

**风险等级：**

【**高危**】

**修复方案：**给jmx-console加上访问密码并且执行以下修复方式：

1. 在 ${jboss.server.home.dir}/deploy下面找到jmx-console.war目录编辑WEB-INF/web.xml文件去掉security-constraint块的注释，使其起作用。
2. 编辑以下文件：

|  |
| --- |
| 编辑  WEB-INF/classes/jmx-console-users.properties  或  server/default/conf/props/jmx-console-users.properties (version >=4.0.2)和  WEB-INF/classes/jmx-console-roles.properties  或server/default/conf/props/jmx-console-roles.properties(version >=4.0.2) 添加用户名密码 |

1. 编辑WEB-INF/jboss-web.xml去掉 security-domain块的注释，security-domain值的映射文件为login-config.xml（该文件定义了登录授权方式）。
2. 对于invoker,采用以下方案：升级相关的JBOSS中间件到最新版本，或者删除 $JBOSS\_HOME/[server]/all/deploy 和$JBOSS\_HOME/[server]/default/deploy下的Jmx-console.war 、Web-console.war这两个.War文件来禁止对Jmx-console和Web-console的访问。

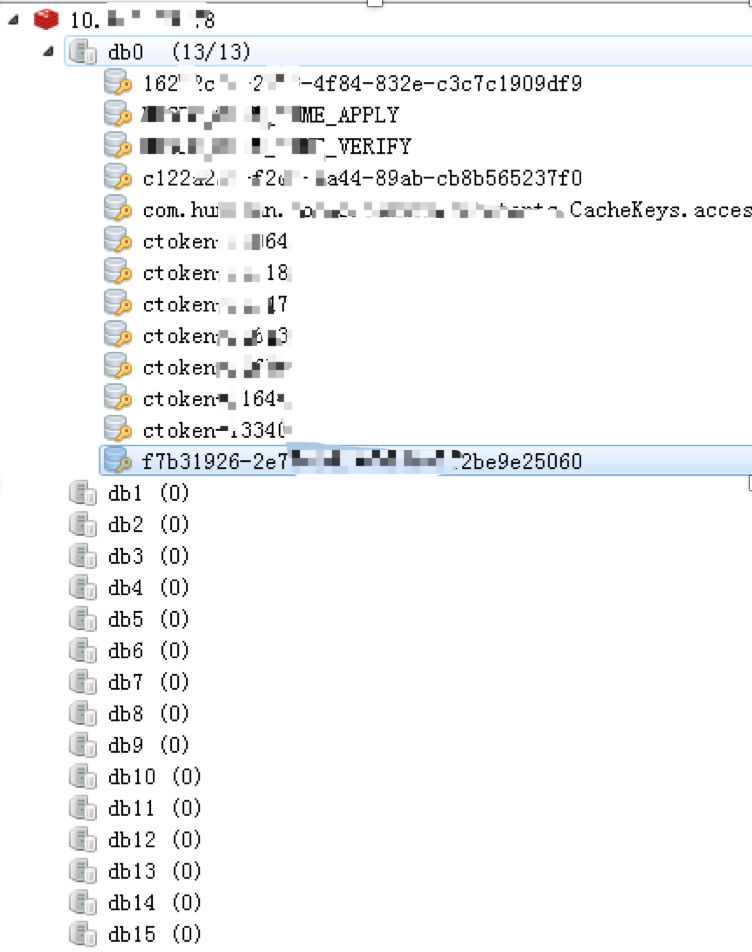
**注意事项：**暂无

* + 1. 第三方应用相关漏洞

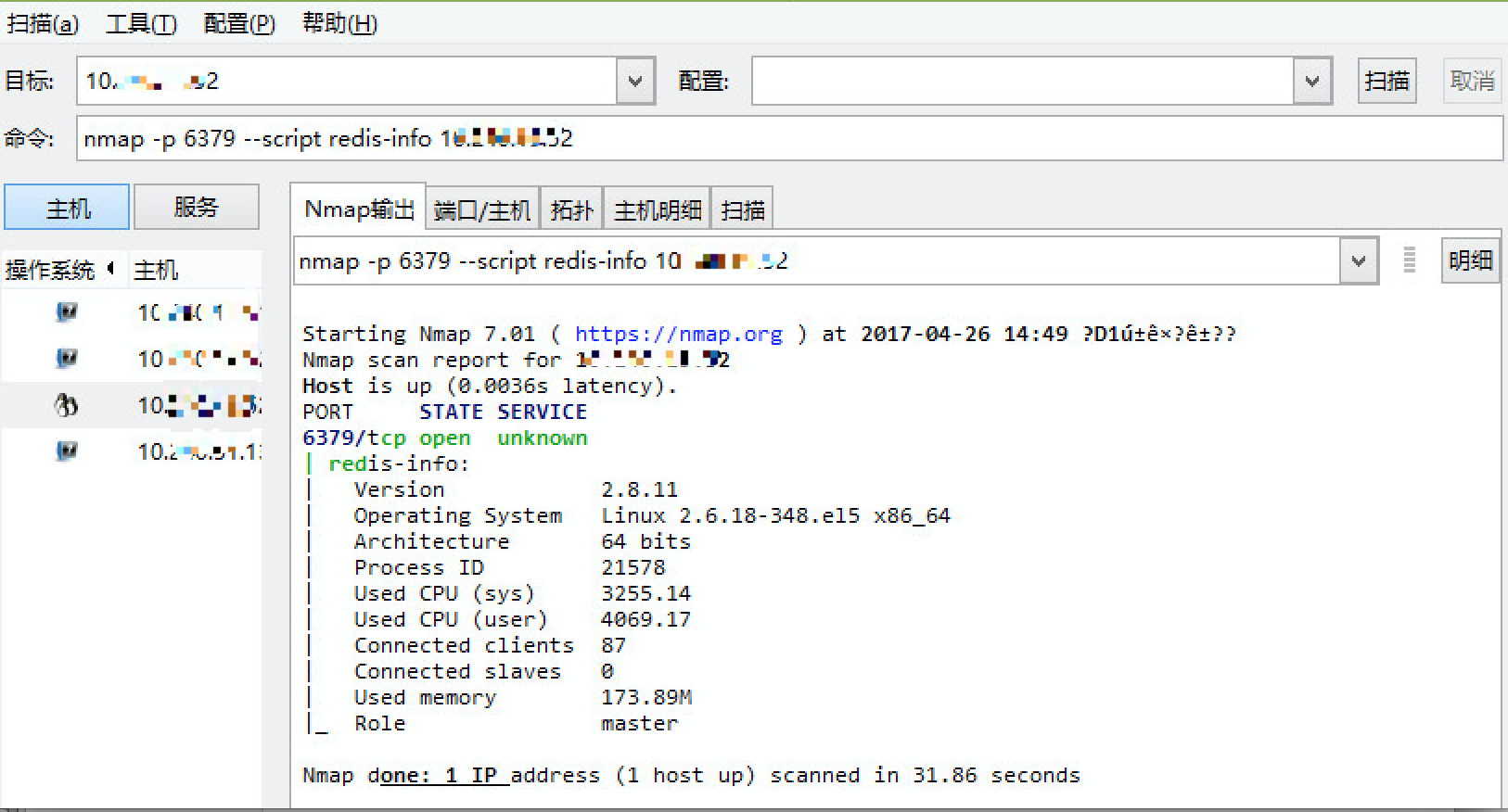
**漏洞描述：**Redis和Memcached等非关系型数据库，Zabbix等其他第三方应用存在的漏洞。

**测试方法：**以Redis未授权访问为例：

1. 使用RedisDesktopManager客户端工具连接Redis，如下图所示：



1. 使用nmap中script扫描，如下图所示：



**风险分析：**Redis 默认情况下，会绑定在 0.0.0.0:6379，导致Redis服务暴露到公网上。

如果在没有开启认证并且在任意用户可以访问目标服务器的情况下，从而可以未授权访问Redis服务，进一步可进行数据增删改查，甚至获取服务器权限等恶意操作。

**风险等级：**

【**高危**】

**修复方案：**

1. 禁止Redis服务对公网开放，可通过修改redis.conf配置文件中的"#bind 127.0.0.1" ，去掉前面的"#"即可（Redis本来就是作为内存数据库，只要监听在本机即可）；

2. 设置密码访问认证，可通过修改redis.conf配置文件中的"requirepass" 设置复杂密码 （需要重启Redis服务才能生效）；  
3. 对访问源IP进行访问控制，可在防火墙限定指定源ip才可以连接Redis服务器；

4. 修改Redis默认端口，将默认的6379端口修改为其他端口；

5. 禁用config指令避免恶意操作，在Redis配置文件redis.conf中配置rename-command项"RENAME\_CONFIG"，这样即使存在未授权访问，也能够给攻击者使用config 指令加大难度；

6. Redis使用普通用户权限，禁止使用 root 权限启动Redis 服务，这样可以保证在存在漏洞的情况下攻击者也只能获取到普通用户权限，无法获取root权限；

**注意事项：**暂无

* + 1. 第三方插件相关漏洞

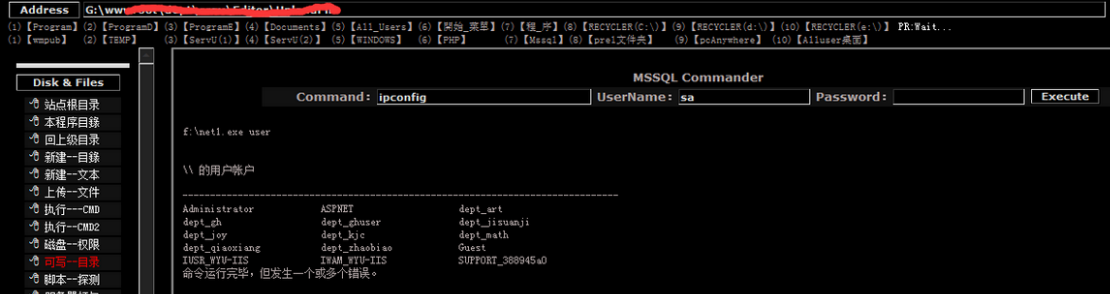
**漏洞描述：**第三方插件，如编辑器fckeditor、ewebeditor等存在的漏洞。

**测试方法：**以ewebeditor上传漏洞为例：

通过网站目录扫描工具对网站进行扫描，扫到有ewebeditor的后台，尝试默认口令admin,admin进行登录，如下图所示：



修改样式管理添加媒体后缀asaspsp，然后上传asp webshell，如下图所示：



**风险分析：**由于开发阶段使用了弱口令，导致正式上线后可通过扫描等方式获取到编辑器后台管理路径 ，通过弱口令进入后台管理，为攻击者提供进一步的攻击途径，可篡改网站页面内容甚至通过上传webshell获取服务器权限。

**风险等级：**

【**高危**】：编辑器存在默认口令，可上传webshell。

**修复方案：**

1. 修改编辑器后台默认路径；
2. 为编辑器后台设置高复杂度口令。

**注意事项：**暂无

* + 1. 开发框架

**漏洞描述：**开发框架存在的漏洞，如Struts2框架漏洞。

**测试方法：**以Struts2远程命令执行为例：

1. 在了解网站所采用的结构框架后，除去伪静态页面，抓包或者读取页面源代码方式，查找到网站系统url为.do和.action结尾类型后，添加相应的远程命令执行代码进行判断。
2. 例如用户可以在http://host.com/X.action?后添加相对应struts2 漏洞的远程命令执行代码，或者直接利用工具K8 Struts2 Exploit.exe进行检测，以下列举几种常见的执行命令exp：

**风险分析：**Struts 2是在struts 和WebWork的技术基础上进行了合并的全新的框架。Struts2漏洞类型分为两种，一种是使用缩写的导航参数前缀时的远程代码执行漏洞，另一种是使用缩写的重定向参数前缀时的开放式重定向漏洞。Struts2远程命令执行，属于高危安全漏洞，可使黑客取得网站服务器的权限。这里我们重点描述相关远程命令执行漏洞。Struts2的DefaultActionMapper支持一种方法，可以使用”action:”, “redirect:”, “redirectAction:”对输入信息进行处理，从而改变前缀参数，这样操作的目的是方便表单中的操作。在2.3.15.1版本以前的struts2中，没有对”action:”, “redirect:” , “redirectAction:”等进行处理，导致ongl表达式可以被执行，如s2-020的漏洞中，利用ognl的class.xx这种方式来遍历属性。

**风险等级：**

【**高危**】

**修复方案：**建议及时更新struts2的版本到最新，如果暂时不方便更新，则提供以下方案进行修复：

1. 除升级外：Struts2远程代码执行漏洞(S2-013) 的临时解决方案：

修改文件：org.apache.struts2.views.util.DefaultUrlHelper 的281–284行：

|  |
| --- |
| private String translateVariable(String input) {  ValueStack valueStack = ServletActionContext.getContext().getValueStack();  return TextParseUtil.translateVariables(input, valueStack);  }  修改成  private String translateVariable(String input) {  return input;  } |

1. Struts2 s2-016/s2-017漏洞官网漏洞说明：

http://struts.apache.org/release/2.3.x/docs/s2-016.html，解决方案(以struts 2.1.8.1版本为例)：

1. 下载struts 2.1.8.1 源码struts-2.1.8.1-src.zip
2. eclipse建立一个war项目
3. 将struts-2.1.8.1-src.zip内的源码拖到项目的src目录下
4. 找到对应struts-2.1.8.1的xwork-core-2.1.6.jar放入war项目的WEB-INF/lib目录下此目的就是为了避免java文件带红叉
5. 修改org.apache.struts2.dispatcher.mapper.DefaultActionMapper.java此时文件应该没有红叉。有红叉主要是缺乏servlet-api.jar和xwork-core-2.1.6.jar在buildpath中改动如下:

找到handleSpecialParameters方法，修改如下:

|  |
| --- |
| public void handleSpecialParameters(HttpServletRequest request,  ActionMapping mapping) {  // handle special parameter prefixes.  Set uniqueParameters = new HashSet();  Map parameterMap = request.getParameterMap();  for (Iterator iterator = parameterMap.keySet().iterator(); iterator  .hasNext();) {  String key = (String) iterator.next();  // Strip off the image button location info, if found  if (key.endsWith(".x") || key.endsWith(".y")) {  key = key.substring(0, key.length() - 2);  }  // -- jason.zhou 20130708 add start -- //  if ((key.contains("redirect:")) || (key.contains("redirectAction:")) || (key.contains("action:"))) {  return;  }  // -- jason.zhou 20130708 add end -- //  // Ensure a parameter doesn't get processed twice  if (!uniqueParameters.contains(key)) {  ParameterAction parameterAction = (ParameterAction) prefixTrie  .get(key);  if (parameterAction != null) {  parameterAction.execute(key, mapping);  uniqueParameters.add(key);  break;  }  }  }  } |

1. 点中org.apache.struts2.dispatcher.mapper.DefaultActionMapper.java，右键，导出(export)，导出为JAR file存盘为test.jar
2. 用winrar打开test.jar，拷贝出org/apache/struts2/dispatcher/mapper/下的DefaultActionMapper类，一共有7个文件，如下：

DefaultActionMapper$1.class

DefaultActionMapper$2$1.class

DefaultActionMapper$2$2.class

DefaultActionMapper$2$3.class

DefaultActionMapper$2$4.class

DefaultActionMapper$2.class

DefaultActionMapper.clas

1. 用winrar打开struts2-core-2.1.8.1.jar进入org/apache/struts2/dispatcher/mapper/目录，将上面拷贝出的7个文件拷贝入，覆盖原来的同名文件。
2. 将struts2-core-2.1.8.1.jar拷贝入你自己的项目，覆盖原来的struts2-core-2.1.8.1.jar
3. 启动项目，重新测试s2-016漏洞，如果发现测试结果正常(不响应.do?action:,.do?redrect:,.do?redirectAction)，则证明补丁已生效。

**注意事项：**暂无

* + 1. 通用型系统

**注意事项：**参考《通用系统测试指南》

1. 致谢

在编写过程中，感谢业务标准化小组成员的大力配合和提出的宝贵建议。

1. 漏洞分类依据

参考《绿盟科技漏洞分类树形和场景.xlsx》



1. 名词解释
2. NoSQL

NoSQL泛指依赖于不同存储机制的非关系型数据库，这些存储机制包括文档存储、键值对存储和图。包括像MongoDB和Cassandra之类的NoSQL数据库，也有像Redis和Memcached这样的内存和缓存存储。

1. JSONP

JSONP全称是JSON with Padding，是基于JSON格式的为解决跨域请求资源而产生的解决方案。他实现的基本原理是利用了 HTML 里 <script></script> 元素标签，远程调用 JSON 文件来实现数据传递。如要在 a.com 域下获取存在 b.com 的 JSON 数据( getUsers.JSON ):

{"id" : "1","name" : "test"}

那么他们可以首先通过 JSONP 的“ Padding ”这个 getUsers.JSON 输出为：

callback({"id" : "1","name" : "test"});

对于实际应用过程中 callback 的名称在后台实现是动态输出的。如上面例子在 PHP 实现：

|  |
| --- |
| <?php  //getUsers.php  $callback = $\_GET['callback'];  print $callback.'({"id" : "1","name" : "test"});';  ?>  然后在 a.com 使用 <script> 进行远程调用,在 Jquery 里可以直接这样调用：  <script type="text/javascript" src="http://www.xxx.com/framework/jquery/1.9.1/jquery-1.9.1.js"></script>  <script type="text/javascript">  $.getJSON("http://www.b.com/getUsers.php?callback=?", function(getUsers){  alert(getUsers.name);  });  </script> |

1. SSO

SSO英文全称Single Sign On，单点登录。它是一种统一认证和授权机制，指访问同一服务器不同应用中的受保护资源的同一用户，只需要登录一次，即通过一个应用中的安全验证后，再访问其他应用中的受保护资源时，不再需要重新登录验证。

1. 风险等级依据

参考《风险评估量化方法论》