X X X X

信 息 技 术 标 准 规 范

WEB安全开发规范

WEB information security development standard

（V1.0）

2021-04-17 发布　　2021-04-17 实施

XXXX 发布

目 录

1. WEB 安全编码规范

1.1. 安全编码

1.1.1.适用范围

本规范的制定考虑了 xxxx 各种 Web 应用的共性, 适合于 xxxx 绝大部分 Web 应用系统，要求 Web 应用系统开发必须遵循。

1.1.2.用词约定

规则：强制必须遵守的原则;

建议：需要加以考虑的原则;

说明：对此规则或建议进行相应的解释;

实施指导：对此规则或建议的实施进行相应的指导;

1.1.3.规范评审

根据研发安全流程控制活动要求，项目立项后，开发团队项目经理或指定接口人根据《需求分析说明书》编写《概要设计说明书》，再根据《概要设计说明书》编写《详细设计说明书》，同时，信息安全工作组对安全需求项和安全设计项进行评审，并建立和完善安全设计规范和安全编码规范。

1.1.4.编码规范

安全编码规范建立：安全编码规范由信息安全工作组提供咨询支持、开发团队安全接口人提供技术和编码支持，结合 xxxx 具体业务系统特性，建立 Web 应用、移动 APP 两种架构，主要 Java 语言的安全编码规范，规范由信息安全工作主要维护并提供版本更新。

1.1.5.规范分类

xxxx 研发安全流程所提安全编码规范主要包括针对：监管合规需求、安全功能需求、安全技术需求、最低需求等的安全建设对策；具体包括开发语言和组件、开发工具要求、编译器版本要求、以及针对安全漏洞缓解措施等在业务系统中的落地进行具体要求和指引。

1.2. 安全漏洞

1.2.1.Web 安全漏洞

针对众多的 Web 漏洞，OWASP 的专家们结合各自在各领域的应用安全工作经验及智慧，提出了十大 Web 应用程序安全风险，帮助人们关注最严重的漏洞。（OWASP 即开放 Web 应用安全项目，是一个旨在帮助人们理解和提高 Web 应用及服务安全性的项目组织。）xxxx 维护的威胁风险库中，Web 安全漏洞基于 OWASP 1 0 大应用风险的细分项：

序号	风险名称	风险描述
1	SQL 注入	攻击者利用注入漏洞，通过构造特殊的语句，可进行后台数据库操作并插入木马，以获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
2	跨站脚本	攻击者利用此漏洞，可以获取其他合法用户的 Cookie 身份信息、访问地址等，通过获取到的 Cookie 信息，即可以被攻击者的身份访问 Web 应用，如获取到管理员的 Cookie，就可以以管理员的身份访问应用系统。
3	任意文件上传	攻击者利用此漏洞，可直接上传木马文件，以获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。

4	XML 实体注入	攻击者利用此漏洞，通过构造特殊的引用或请求可以读取服务器文件或执行操作系统命令，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
5	EL 表达式注入	攻击者利用 EL 表达式的方法，EL 表达式语法允许开发人员开发自定义函数，以调用 Java 类的方法。语法：${prefix：method(params)}。在 EL 表达式中调用的只能是 Java 类的静态方法，这个 Java 类的静态方法需要在 TLD 文件中描述，才可以被 EL 表达式调用。EL 自定义函数用于扩展 EL 表达式的功能，可以让 EL 表达式完成普通 Java 程序代码所能完成的功能。从而攻击者可以构造代码执行，获取系统的执行命令权限。如果这些信息有助于攻击者进一步攻击利用，可能导致更严重的危害。（存在风险点的位置：用于表达式的运算。如：加、减、乘、除,用于从作用域中取出数据）
6	账号弱口令	攻击者利用弱口令，可以获取特定账户或应用的访问控制权限，如果进一步攻击利用可能获取服务器控制权限。
7	跨站请求伪造	攻击者利用此漏洞，可以以受害者的身份发起 HTTP 请求，访问 Web 应用的相应功能，如果利用成功，那么攻击者就可以更新用户的相关数据（添加、删除、修改），如添加管理员账户、向指定账户转账。
8	文件包含	攻击者利用此漏洞，可以执行任意代码，获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
9	目录遍历	攻击者利用此漏洞，可以操作服务器指定文件，特别是一些敏感文件，如程序配置文件、数据库配置文件、程序代码文件、服务器账户文件等；如果进一步利用可获取服务器敏感数据或获取服务器控制权限。
10	服务端请求伪造	攻击者利用此漏洞，可以利用 Web 服务器发起请求（HTTP、FTP、HTTPS 等等），可以突破内外网访问控制、部分安全控制措施，如果进一步利用可获取内网服务器权限及敏感数据。
11	Java 反序列化	攻击者利用此漏洞，通过构造特殊的引用或请求可以读取服务器文件或执行操作系统命令，

		包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。（导入模版文件、网络通信、数据传输、日志格式化存储、对象数据落磁盘或 DB 存储等业务场景等）。
12	有漏洞的组件	攻击者利用此漏洞，可执行恶意操作系统命令，获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
13	敏感信息泄漏	攻击者利用此漏洞，可以访问到程序备份文件，可能导致源代码或者数据泄露，如果进一步被利用可能导致更严重的危害。
14	不安全加密算法	攻击者利用此漏洞，可以获取到对应的铭感信息进行下一步的攻击。
15	任意文件下载	攻击者利用此漏洞，可以下载服务器任意文件，如脚本代码，服务及系统配置文件等；可用得到的代码进一步代码审计，得到更多可利用漏洞。
16	任意密码找回	攻击者利用此漏洞，可以找回任意账户的密码。
17	HTML 注入	攻击者利用此漏洞，可以访问引导用户访问恶意的网页，如果这个网页是钓鱼、挂马的网页，可能导致更严重的危害。
18	隐藏暗链	暗链对政府网站来讲危害更大，当用户通过搜索引擎搜索某地政府网站时，可能从搜索引擎的描述中看到一些非法的关键词，从而影响政府的形象。境外敌对势力甚至可以根据政府网站是否存在暗链来判断一个政府网站是否存在安全漏洞，并决定是否发起攻击。成功入侵后，可以发布虚假政策信息造成群众对政府的不信任，制造政府与群众之间的矛盾，引发社会事件。
19	未授权访问	攻击者利用此漏洞，可以直接访问相应的页面，如果此页面存在敏感数据或信息，那么这些敏感数据或信息可能发生泄露，如果这些信息有助于攻击者进一步攻击利用，可能导致更严重的危害。
20	越权访问	攻击者利用此漏洞，可以访问其他用户的数据，可能导致数据泄露，如果进一步被利用可能导致更严重的危害。
21	固定 SessionID 漏洞	攻击者可以简单的伪造一个 SessionID 诱使用户使用该 SessionID 登录，即可获取登录权限。

22	URL 重定向	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。
23	Flash 访问	攻击者利用此漏洞，可以以受害者的身份发起 HTTP 请求，访问 Web 应用的相应功能，如果利用成功，那么攻击者就可以更新用户的相关数据（添加、删除、修改），如添加管理员账户、向指定账户转账。

1.2.2.业务场景漏洞

业务场景漏洞基于业务模块功能可能产生的漏洞细分项：

序号	风险名称	风险描述
1	用户登录	攻击者通过用户登录模块漏洞，可以获取特定账户或应用的访问控制权限，如果进一步攻击利用可能获取服务器控制权限。
2	用户注册	攻击者通过用户注册模块漏洞，可以通过恶意注册，结合其他攻击手段进一步利用并扩大漏洞危害。
3	密码找回	攻击者利用找回密码用其他攻击进行下一步的攻击操作，甚至攻击成功的话，可能导致获取网站用户登录的部分权限，包括修改删除用户信息、窃取用户敏感信息。
4	后台管理	攻击者利用后台管理漏洞，可以通过登录后台或者 url 跳转等操作，对服务器指定文件，特别是一些敏感文件，如程序配置文件、数据库配置文件、程序代码文件、服务器账户文件等，从而达到下一步的攻击。

5	接口调用	可以通过攻击接口，获取敏感的数据或者能访问一些访问权限。
6	会员系统	通过攻击会员系统，可以越权访问权限，个人信息，图片上传，从而获取下一步的攻击，结合其他安全漏洞可能造成更严重的危害。
7	业务办理	攻击者通过攻击业务办理模块，可能存在替代办理、业务绕过、篡改他人的信息。
8	业务查询	攻击者利用此漏洞，可能会造成恶意查询或者可能办理人的信息泄露，如果进一步被利用可能导致更严重的危害。
9	业务传输	攻击者通过攻击 cookie，劫持 cookie，替换会话等攻击手段，从而进一步攻击利用可能获取服务器控制权限。
10	发表评论	攻击者利用发表评论漏洞，可能访问其他用户的数据，可能导致数据泄露，如果进一步被利用可能导致更严重的危害。
11	购买支付	攻击者利用此漏洞，可以篡改商品的订单，金额，遍历交易信息。如果进一步篡改可导致平台上损失。
12	账号充值	攻击者通过账号充值，对充值功能模块进行虚拟充值，篡改充值，篡改账号等，从而导致充值业务被破坏。
13	抽奖活动	攻击者可以通过抽奖活动，进行薅羊毛，盗刷积分，刷取奖品等攻击行为，从而导致正常的抽奖活动被破坏。
14	代金优惠	攻击者通过攻击代金优惠，可以进行对代金业务进行盗刷，篡改等，甚至会给平台带来一定的损失。
15	订单信息	如果订单泄露，可能会造成个人的敏感信息泄露。如果进一步被利用可能导致更严重的危害。

16	运费账单	攻击者利用运费账单漏洞,通过运费的漏洞利用，篡改或者构造假账单，从而让平台造成一定的损失。
17	第三方商家	攻击者利用平台上的第三方商家,可能会造成第三方商家被盗号，商家账号被遍历，或者越权访问其他商家的用户信息,甚至可能窃取用户敏感信息。

1.2.3.攻击行为风险

业务攻击风险基于业务场景功能可能产生的风险细分项:

序号	风险名称	风险描述
1	短信验证	攻击者利用验证码爆破漏洞,通过构造的密码字典，对用户的验证码进行暴力猜测，可以获取网站用户登录的部分权限,包括修改删除用户信息、窃取用户敏感信息。(存在风险点的位置：登录处、注册处)。
2	用户撞库	攻击者撞库成功后,可以获取网站用户登录的部分权限，包括修改删除用户信息、窃取用户敏感信息。(存在风险点的位置：登录处、注册处)
3	爆破密码	攻击者利用弱口令漏洞，通过进行暴力猜解,获取网站管理权限，包括修改删除网站页面、窃取数据库敏感信息、植入恶意木马；甚至以网站为跳板，获取整个内网服务器控制权限。
4	ID 遍历	攻击者利用该漏洞，通过遍历个人信息，可能获取网站帐户等敏感信息;通过结合其他漏洞可能造成更严重的危害。（存在风险点位置:用户个人信息，订单信息等。）
5	风控缺失	攻击者利用此漏洞,通过脚本和程序进行批量的用户注册和登录操作,从而进行下一步的攻击，如薅羊毛等攻击。（存在分析点位置：登录处，密码找回处，用户注册处。）

1.3. 开发构建

1.3.1.语言和组件

xxxx 业务系统采用的主要开发语言和调用的框架或组件如下, 如有增加新的框架需更新注明:

序号	类别	详细描述
1	主要语言	Java,JavaScript
2	附加语言	peoplecode，Objective-C
3	调用框架组件	Spring，Hibernate，JeeSite，MyBatis，FastJson，Shiro，jQuery，SpringMVC，Struts2，supcan，dojo，Struts，freemarker，sonarqube，dubbo，spring-data-jpa，emoss，用友
4	应用服务器	Tomcat，WebLogic，Domino，WebSphere

1.3.2.版本更新要求

调用的开源框架不能使用有漏洞的版本，需到官方下载当前最新的版本，当前最新的版本在业务上线前有漏洞爆出时需更新到最新的无漏洞的版本, 可以使用开发工具插件检查各模块最新版本，各组件官方下载地址如下

序号	组件名称	官网下载地址
1	Spring	http://spring.io/projects/spring-framework https://projects.spring.io/spring-security/
2	Hibernate	http://hibernate.org/orm/downloads/ http://hibernate.org/validator/
3	JeeSite	http://www.jeesite.com/
4	MyBatis	http://www.mybatis.org/mybatis-3/
5	FastJson	https://github.com/alibaba/fastjson
6	Shiro	http://shiro.apache.org/
7	jQuery	http://jquery.com/
8	Struts2	https://struts.apache.org/ https://struts.apache.org/security/

9	Spring boot	http://spring.io/projects/spring-boot
10	dojo	https://dojotoolkit.org/download/
11	freemarker	http://freemarker.org/freemarkerdownload.html
12	sonarqube	https://www.sonarqube.org/
13	dubbo	http://dubbo.apache.org/zh-cn/
14	spring-data-jpa	http://spring.io/projects/spring-data-jpa

1.3.3.开发工具要求

开发人员使用的开发工具需从官方下载，并用杀毒软件做病毒扫描，不建议从其他第三方网站下载，建议下载最新的版本，不要用迅雷等 BT 下载工具下载。

序号	开发工具	官网下载地址
1	Eclipse	http://www.eclipse.org/downloads/
2	IntelliJ IDEA	https://www.jetbrains.com/idea/
3	Application Designer	http://www.oracle.com/us/products/applications/peoplesoft-enterprise/tools-tech/index.html
4	Lotus Designer	https://www.ibm.com/developerworks/cn/downloads/ls/dominodesigner/

1.3.4.应用服务器要求

应用服务器软件不能使用有漏洞的版本，建议更新到最新的无漏洞的版本，各应用服务器软件安全公告官网如下，如有增加新需更新注明:

序号	开发工具	官网下载地址
1	Tomcat	http://tomcat.apache.org/ https://tomcat.apache.org/security.html
2	WebLogic	https://www.oracle.com/middleware/technologies/weblogic.html https://docs.oracle.com/cd/E24329_01/web.1211/e24446/security.htm#INTRO232
3	WebSphere	https://www.ibm.com/developerworks/cn/websphere/ https://www.ibm.com/developerworks/webspher

		e/zones/was/security/index.html
4	Domino	https://www.ibm.com/us-en/marketplace/ibm-domino https://www.ibm.com/security/secure-engineering/bulletins.html

1.3.5.编译器版本要求

开发人员不能使用有漏洞的低版本或老版本编译器和运行库，建议下载最新的无漏洞的版本，各编译器软件官网如下，如有增加新需更新注明:

序号	编译和运行库	官网下载地址
1	JDK 1.8	https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
2	JDK 1.7	https://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html 已停止免费更新（仅对 Oracle Customers 更新），不建议用
3	JDK 1.6	已停止免费更新（仅对 Oracle Customers 更新），不建议用
4	Microsoft cl	https://visualstudio.microsoft.com/
5	Sun cc	https://www.oracle.com/technetwork/server-storage/solarisstudio/downloads/index-jsp-138519.html

1.3.6.依赖库版本扫描

对系统调用的各依赖 jar 包及组件需要通过 OWASP Dependency Check 做版本漏洞扫描，确保当前使用的各依赖包版本最新的无漏洞的版本，OWASP Dependency Check 使用方法如下:

dependency-check.bat --project gbss -s "C:\repository"

然后会在当前目录生成报告文件。

1.4. 通用规范

通用的安全编码规范包括禁止后门、输入验证、输出编码、上传下载、异常处理、代码注释、归档要求等，安全编码规范将针对业务系统安全开发过程中的落地进行具体要求。

1.4.1.禁止后门

规则 4.1.1：禁止在系统（代码）中留有预留任何的后门帐号或特殊的访问机制用以维护、支持或操作的需要。

说明: 例如系统中留了一个特别的帐号和密码、没有公开的通信字（功能码）等、没有公开的参数等。

1.4.2.输入验证

规则 4.2.1：必须对所有用户产生的输入进行校验，一旦数据不合法，应该告知用户输入非法并且建议用户纠正输入。

说明：用户产生的输入是指来自 text、password、textareas 或 file 表单域的数据；必须假定所有用户产生的输入都是不可信的，并对它们进行合法性校验。

规则 4.2.2：必须对所有服务器产生的输入进行校验，一旦数据不合法，必须使会话失效，并记录告警日志。

说明：服务器产生的输入是指除用户产生的输入以外的输入，例如来自 hidden fields、selection boxes、check boxes、radio buttons、cookies、HTTP headers、热点链接包含的 URL 参数的数据或客户端脚本等；必须假定所有服务器产生的输入都是被篡改过的、恶意的，并对它们进行合法性校验，如果不合法，说明有人恶意篡改数据。举例：假如用户资料填写表单中的“性别”为必填项，用 radio button（‘男’和‘女’对应实际值分别为‘1’和‘0’）来限制用户的输入，如果应用程序收到的“性别”值为‘2’，那么可以断定有人恶意篡改数据。

规则 4.2.3：禁止将 HTTP 标题头中的任何未加密信息作为安全决策依据。

说明: HTTP 标题头是在 HTTP 请求和 HTTP 响应的开始阶段发送的。Web 应用程序必须确保不以 HTTP 标题头中的任何未加密信息作为安全决策依据，因为攻击者要操作这一标题头是很容易的。例如，标题头中的 referer 字段包含来自请求源端的 Web 页面的 URL。不要根据 referer 字段的值做出任何安全决策（如检查请求是否来源于 Web 应用程序生成的页面），因为该字段是很容易被伪造的。

规则 4.2.4：不能依赖于客户端校验，必须使用服务端代码对输入数据进行最终校验。

说明：客户端的校验只能作为辅助手段，减少客户端和服务端的信息交互次数。

规则 4.2.5：对于在客户端已经做了输入校验，在服务器端再次以相同的规则进行校验时，一旦数据不合法，必须使会话失效，并记录告警日志。

说明：肯定存在攻击行为，攻击者绕过了客户端的输入校验，因此必须使会话失效，并记入告警日志。

规则 4.2.6：如果输入为数字参数则必须进行数字型判断。

说明：这里的数字参数指的是完全由数字组成的数据。

实施指导:

```
String mobileno = request.getParameter("mobileno");
String characterPattern = "^\\d+$";    //正则表达式表示是否全为数字
if (!mobileno.matches (characterPattern))
{
out.println (“Invalid Input”);
}
```

规则 4.2.7：如果输入只允许包含某些特定的字符或字符的组合，则使用白名单进行输入校验。

说明：对于一些有规则可循的输入，如 email 地址、日期、小数等，使用正则表达式进行白名单校验，这样比使用黑名单进行校验更有效。

实施指导:

```
email 地址校验的方法:
String emailAddress = request.getParameter("emailAddress");
String characterPattern =
"^([a-z0-9A-Z]+[_-]?)+[a-z0-9A-Z]@(([a-z0-9A-Z]+[_-]?)+(-[a-z0-9A-Z]+)?\\.)
+[a-zA-Z]{2,4}$"; //email 正则表达式
if (!emailAddress.matches(characterPattern))
{
out.println ("Invalid Email Address");
}
```

规则 4.2.8：如果输入为字符串参数则必须进行字符型合法性判断。

说明：可定义一个合法字符集。

实施指导:

```
String text = request.getParameter("text");
String characterPattern = "^[A-Za-z]*$"; //开发者自行定义字符规则(方括号
内的字符集)
if (!text.matches (characterPattern))
{
out.println ("Invalid Input");
}
```

规则 4.2.9：校验输入数据的长度。

说明：如果输入数据是字符串，必须校验字符串的长度是否符合要求，长度校验会加大攻击者实施攻击的难度。

规则 4.2.10：校验输入数据的范围。

说明：如果输入数据是数值，必须校验数值的范围是否正确，如年龄应该为 0 ~ 150 之间的正整数。

规则 4.2.11：禁止通过字符串串联直接使用用户输入构造可执行 SQL 语句。

说明：禁止通过字符串串联直接使用用户输入构造可执行 SQL 语句，如：

```
string sql = "select status from Users where UserName='" + txtUserName.Text +
```

""";这样很容易被 SQL 注入攻击。

规则 4.2.12：对于 java/JSP 语言，使用预编译语句 PreparedStatement 代替直接的语句执行 Statement。

说明：使用预编译语句 PreparedStatement，类型化 SQL 参数将检查输入的类型，确保输入值在数据库中当作字符串、数字、日期或 boolean 等值而不是可执行代码进行处理，从而防止 SQL 注入攻击。而且，由于 PreparedStatement 对象已预编译过，所以其执行速度要快于 Statement 对象。因此，多次执行的 SQL 语句经常创建为 PreparedStatement 对象，还可以提高效率。

实施指导：

参考如下代码：

```
String inssql = "insert into buy(empid, name, age, birthday) values (?,?,?,?)";
PreparedStatement stmt = null;
stmt = conn.prepareStatement(inssql);
stmt.setString(1, empid);
stmt.setString(2, name);
stmt.setInt(3, age);
stmt.setDate(4, birthday);
stmt.execute();
```

备注：使用 like 进行模糊查询时，如果直接用"select * from table where comment like %?%"，程序会报错，必须采用如下方法

```
String express = "select * from table where comment like ?";
pstmt = con.prepareStatement(express);
String c="hello";
pstmt.setString(1, "%"+c+"%");
//参数自动添加单引号，最后的 SQL 语句为：select * from  table where comment like '%hello%'
pstmt.execute();
```

规则 4.2.13：禁止动态构建 XPath 语句。

说明：和动态构建 SQL 一样，动态构建 XPath 语句也会导致注入漏洞（XPath 注入）。动态构建 XPath 语句的例子：public boolean doLogin(String loginID, String password){.....

XPathExpression expr = xpath.compile("//users/user[loginID/text()='"+loginID+"' and password/text()='"+password+"']/firstname/text()");

.....}

规则 4.2.14：在 JavaBean 中禁止使用 property="*"进行参数赋值。

说明：property="*"这表明用户在可见的 JSP 页面中输入的，或是直接通过 Query String 提交的参数值，将存储到与参数名相匹配的 bean 属性中。例如，网上购物程序，一般，用户是这样提交请求的：http://www.somesite.com /addToBasket.jsp?newItem=ITEM0105342 ，如果用户提交：http://www.somesite.com /addToBasket.jsp?newItem=ITEM0105342&balance=0，这样，balance=0 的信息就被在存储到了 JavaBean 中了，而 balance 是整个会话中用来存储总费用的，当他们这时点击“chekout”结账的时候，费用就全免了。

规则 4.2.15：用于重定向的输入参数不能包含回车和换行字符，以防止 HTTP 响应拆分攻击。

说明：注意，“回车”字符有多种表示方式（CR = %0d = \r ），“换行”字符有多种表示方式（LF = %0a = \n）。

规则 4.2.16：如果服务端代码执行操作系统命令，禁止从客户端获取命令。

说明：如果服务端代码中使用 Runtime.getRuntime().exec(cmd) 或 ProcessBuilder 等执行操作系统命令，那么禁止从客户端获取命令；而且最好不要从客户端获取命令的参数，如果必须从客户获取命令的参数，那么必须采用正则表达式对命令参数进行严格的校验，以防止命令注入（因为，一旦从客户端获取命令或参数，通过;&|<>符号，非常容易构造命令注入，危害系统）。

1.4.3.输出编码

规则 4.3.1：对于不可信的数据，输出到客户端前必须先进行 HTML 编码。

说明：不可信的数据（也就是其他业务系统生成的未经本应用程序验证的表数据或文件数据），通过对输出到客户端的数据进行编码，可以防止浏览器将 HTML 视为可执行脚本，从而防止跨站脚本攻击。

实施指导：

JSP 语言可以通过替换输出数据的特殊字符【& < > ” ’ ( )%+-】为其他表示形式后再输出给客户端，例如：

```
<%
String OutStr = "<script>alert('XSS')</script>";
OutStr = OutStr.replaceAll("&","&amp;");
OutStr = OutStr.replaceAll("<","&lt;");
OutStr = OutStr.replaceAll(">","&gt;");
OutStr = OutStr.replaceAll("\"","&quot;");
OutStr = OutStr.replaceAll("\'","&#39;");
OutStr = OutStr.replaceAll("\\(","&#40;");
OutStr = OutStr.replaceAll("\\)","&#41;");
out.println(OutStr);
%>
```

ASP.NET 语言可以通过 HtmlEncode 方法对 HTML 的输出进行编码。

PHP 语言可以通过 htmlentities 或 htmlspecialchars 方法对 HTML 输出进行编码。

1.4.4.上传下载

规则 4.4.1：必须在服务器端采用白名单方式对上传或下载的文件类型、大

小进行严格的限制。

规则 4.4.2：禁止以用户提交的数据作为读/写/上传/下载文件的路径或文件名，以防止目录跨越和不安全直接对象引用攻击。

说明：建议对写/上传文件的路径或文件名采用随机方式生成，或将写/上传文件放置在有适当访问许可的专门目录。对读/下载文件采用映射表（例如，用户提交的读文件参数为 1，则读取 file1，参数为 2，则读取 file2）。防止恶意用户构造路径和文件名，实施目录跨越和不安全直接对象引用攻击。

规则 4.4.3：禁止将敏感文件（如日志文件、配置文件、数据库文件等）存放在 Web 内容目录下。

说明：Web 内容目录指的是：通过 Web 可以直接浏览、访问的目录，存放在 Web 内容目录下的文件容易被攻击者直接下载。

1.4.5.异常处理

规则 4.5.1：应用程序出现异常时，禁止向客户端暴露不必要的信息，只能向客户端返回一般性的错误提示消息。

说明：应用程序出现异常时，禁止将数据库版本、数据库结构、操作系统版本、堆栈跟踪、文件名和路径信息、SQL 查询字符串等对攻击者有用的信息返回给客户端。建议重定向到一个统一、默认的错误提示页面，进行信息过滤。

规则 4.5.2：应用程序捕获异常，并在日志中记录详细的错误信息。

说明：记录详细的错误消息，可供入侵检测及问题定位。

1.4.6.代码注释

规则 4.6.1：在注释信息中禁止包含物理路径信息。

规则 4.6.2：在注释信息中禁止包含数据库连接信息。

规则 4.6.3：在注释信息中禁止包含 SQL 语句信息。

规则 4.6.4：对于静态页面，在注释信息中禁止包含源代码信息。

规则 4.6.5：对于动态页面不使用普通注释，只使用隐藏注释。

说明：动态页面包括 ASP、PHP、JSP、CGI 等由动态语言生成的页面。通过浏览器查看源码的功能，能够查看动态页面中的普通注释信息，但看不到隐藏注释（隐藏注释不会发送给客户端）。因此，为了减少信息泄漏，建议只使用隐藏注释。

实施指导：

```
<form action=h.jsp>
<%--隐藏注释 1--%>
<textarea name=a length=200></textarea>
<input type=submit value=test>
</form>
<%
//隐藏注释 2
java.lang.String str=(String)request.getParameter("a");
/*隐藏注释 3*/
str = str.replaceAll("<","&lt;");
out.println(str);
%>
```

1.4.7.归档要求

规则 4.7.1：版本归档时，必须删除开发过程（包括现场定制）中的临时文件、备份文件、无用目录等。

说明：恶意用户可以通过 URL 请求诸如.bak 之类的文件，Web 服务器会将这些文件以文本方式呈现给恶意用户，造成代码的泄漏，严重威胁 Web 应用的安全。

实施指导：

在 web 应用的根目录下执行以下命令：

find ./ -name "*.old" -o -name "*.OLD" -o -name "*.bak" -o -name "*.BAK" -o -name "*.temp" -o -name "*.tmp" -o -name "*.save" -o -name "*.backup" -o -name "*.orig" -o -name "*.000" -o -name "*~" -o -name "*~1" -o -name "*.dwt" -o -name "*.tpl" -o -name "*.zip" -o -name "*.7z" -o -name "*.rar" -o -name "*.gz" -o -name "*.tgz" -o -name "*.tar" -o -name "*.bz2"

分析查找到的文件是否临时文件、备份文件、无用文件，如果是则删除。

规则 4.7.2：归档的页面程序文件的扩展名必须使用小写字母。

说明：很多 Web server 对大小写是敏感的，但对后缀的大小写映像并没有做正确的处理。攻击者只要在 URL 中将 JSP 文件后缀从小写变成大写，Web 服务器就不能正确处理这个文件后缀，而将其当作纯文本显示。攻击者可以通过查看源码获得这些程序的源代码。因此，归档的页面程序文件的扩展名必须使用小写字母，如 jsp、html、htm、asp 等页面程序文件的扩展名分别为 jsp、html、htm、asp。

规则 4.7.3：归档的程序文件中禁止保留调试用的代码。

说明：这里的“调试用的代码”是指开发过程中进行临时调试所用的、在 Web 应用运行过程中不需要使用到的 Web 页面代码或 servlet 代码。例如：在代码开发过程中为了测试一个添加帐号的功能，开发人员临时编写了一个 JSP 页面进行测试，那么在归档时，该 JSP 页面必须删除，以免被攻击者利用。

1.4.8.其他要求

规则 4.8.1：对于 JSP 语言，所有 servlet 必须进行静态映射，不允许通过绝对路径访问。

说明：在 web.xml 文件中为 servlet 配置 URI 映射，使用 servlet 时，引用它的 URI 映射，而不允许通过绝对路径访问。

规则 4.8.2：对客户端提交的表单请求进行合法性校验，防止跨站请求伪造攻击。

说明：跨站请求伪造（CSRF）是一种挟制终端用户在当前已登录的 Web 应用程序上执行非本意的操作的攻击方法。攻击者可以迫使用户去执行攻击者预先设置的操作，例如，如果用户登录网络银行去查看其存款余额，他没有退出网络银行系统就去了自己喜欢的论坛去灌水，如果攻击者在论坛中精心构造了一个恶意的链接并诱使该用户点击了该链接，那么该用户在网络银行帐户中的资金就有可能被转移到攻击者指定的帐户中。当 CSRF 针对普通用户发动攻击时，将对终端用户的数据和操作指令构成严重的威胁；当受攻击的终端用户具有管理员帐户的时候，CSRF 攻击将危及整个 Web 应用程序。

实施指导：

方法一：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
}
```

方法二：受理重要操作请求时，在相应的表单页面增加图片验证码，用户提交操作请求的同时提交验证码，在服务器端先判断用户提交的验证码是否正确，验证码正确再受理操作请求。

规则 4.8.3：使用.innerHtml 时，如果只是要显示文本内容，必须在 innerHTML 取得内容后，再用正则表达式去除 HTML 标签，以预防跨站脚本。

说明：使用.innerHtml 会将内容以 HTML 显示，容易被利用，导致跨站脚本。

实施指导：

```
<a
href="javascript:alert(document.getElementById('test').innerHTML.replace(/<.+?
>/gim,''))">无 HTML,符合 W3C 标准</a>
```

备注：还可以使用.innerText 代替.innerHtml，.innerText 只显示文本内容不显示 HTML 标签，但.innerText 不是 W3C 标准的属性，不能适用于所有浏览器（但适用于 IE 浏览器）。

规则 4.8.4：禁止使用 eval()函数来处理用户提交的字符串。

说明：eval()函数存在安全隐患，该函数可以把输入的字符串当作 JavaScript 表达式执行，容易被恶意用户利用。

建议 4.8.5：关闭登录窗体表单中的自动填充功能，以防止浏览器记录用户名和口令。

说明：浏览器都具有自动保存用户输入数据和自动填充数据的能力。为了保障用户名和口令的安全，必须关闭自动填充选项，指示浏览器不要存储登录窗口中用户名、口令等敏感信息。

实施指导：

在 form 表单头中增加选项（autocomplete="off"），例如：

```
<form action="Login.jsp" name=login method=post autocomplete="off">
```

建议 4.8.6：防止网页被框架盗链或者点击劫持。

说明：框架盗链和点击劫持（ClickJacking）都利用到框架技术，防范措施就是防止网页被框架。

实施指导：

方法一：在每个网页上增加如下脚本来禁止 iframe 嵌套：

```
< <script>
    if(top != self) top.location.href = location.href;
```

```
</script>
```

1.5. API 安全

针对 API 的安全设计，安全编码要求参考:

1.5.1.身份认证

规则 5.1.1：不要使用 Basic Auth ，使用标准的认证协议 (如 JWT, OAuth)。

规则5.1.2:不要再造 Authentication, token generating, password storing 这些轮子，使用标准的。

规则 5.1.3：在登录中使用 Max Retry 和自动封禁功能。

规则 5.1.4：加密所有的敏感数据。

1.5.2.JWT (JSON Web Token)

规则 5.2.1：使用随机复杂的密钥 (JWT Secret) 以增加暴力破解的难度。

规则 5.2.2：不要在请求体中直接提取数据，要对数据进行加密 (HS256 或 RS256)。

规则 5.2.3：使 token 的过期时间尽量的短 (TTL, RTTL)。

规则 5.2.4：不要在 JWT 的请求体中存放敏感数据，它是可破解的。

1.5.3.OAuth 授权或认证协议

规则 5.3.1：始终在后台验证 redirect_uri，只允许白名单的 URL。

规则 5.3.2：每次交换令牌的时候不要加 token (不允许 response_type=token)。

规则 5.3.3：使用 state 参数并填充随机的哈希数来防止跨站请求伪造

(CSRF)。

规则 5.3.4：对不同的应用分别定义默认的作用域和各自有效的作用域参数。

1.5.4.访问保护

规则 5.4.1：限制流量来防止 DDoS 攻击和暴力攻击。

规则 5.4.2：在服务端使用 HTTPS 协议来防止 MITM 攻击。

规则 5.4.3：使用 HSTS 协议防止 SSLStrip 攻击。

1.5.5.输入验证

规则 5.5.1：使用与操作相符的 HTTP 操作函数，GET (读取)，POST (创建)，PUT (替换/更新) 以及 DELETE (删除记录)，如果请求的方法不适用于请求的资源则返回 405 Method Not Allowed。

规则 5.5.2：在请求头中的 content-type 字段使用内容验证来只允许支持的格式 (如 application/xml，application/json 等等) 并在不满足条件的时候返回 406 Not Acceptable。

规则 5.5.3：验证 content-type 的发布数据和你收到的一样 (如 application/x-www-form-urlencoded，multipart/form-data，application/json 等等)。

规则 5.5.4：验证用户输入来避免一些普通的易受攻击缺陷 (如 XSS，SQL-注入，远程代码执行 等等)。

规则 5.5.5：不要在 URL 中使用任何敏感的数据 (credentials，Passwords，security tokens，or API keys)，而是使用标准的认证请求头。

规则 5.5.6：使用一个 API Gateway 服务来启用缓存、访问速率限制 (如 Quota，Spike Arrest，Concurrent Rate Limit) 以及动态地部署 APIs resources。

1.5.6.资源处理

规则 5.6.1：检查是否所有的终端都在身份认证之后，以避免被破坏了的认证

体系。

规则 5.6.2：避免使用特有的资源 id. 使用 /me/orders 替代 /user/654321/orders。

规则 5.6.3：使用 UUID 代替自增长的 id。

规则 5.6.4：如果需要解析 XML 文件，确保实体解析(entity parsing)是关闭的以避免 XXE 攻击。

规则 5.6.5：如果需要解析 XML 文件，确保实体扩展(entity expansion)是关闭的以避免通过指数实体扩展攻击实现的 Billion Laughs/XML bomb。

规则 5.6.6：在文件上传中使用 CDN。

规则 5.6.7：如果需要处理大量的数据，使用 Workers 和 Queues 来快速响应，从而避免 HTTP 阻塞。

规则 5.6.8：不要忘了把 DEBUG 模式关掉。

1.5.7.输出处理

规则 5.7.1：发送 X-Content-Type-Options: nosniff 头。

规则 5.7.2：发送 X-Frame-Options: deny 头，X-XSS-Protection: 1; mode=block 头。

规则 5.7.3：发送 Content-Security-Policy: default-src 'none' 头。

规则 5.7.4：删除指纹头 - X-Powered-By, Server, X-AspNet-Version 等等。

规则 5.7.5：在响应中强制使用 content-type，如果你的类型是 application/json 那么你的 content-type 就是 application/json。

规则 5.7.6：不要返回敏感的数据，如 credentials, Passwords, security tokens。

规则 5.7.7：在操作结束时返回恰当的状态码. (如 200 OK, 400 Bad Request, 401 Unauthorized, 405 Method Not Allowed 等等)。

1.6. Spring Boot 安全

针对 Spring Boot 的安全设计，安全编码要求参考：

https://docs.spring.io/spring-security/site/docs/current/reference/html/

1.6.1.使用 HTTPS

规则 6.1.1：通过扩展 WebSecurityConfigurerAdapter 要求安全连接，强制使用 HTTPS;

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(HttpSecurity http) throws Exception {
     http.requiresChannel().requiresSecure();
  }
}
```

1.6.2.依赖检查

规则 6.2.1：对依赖库进行漏洞检测，及时更新有漏洞的组件，可以使用 OWASP Dependency Check 或 Snyk（需要登录）工具等进行扫描检查。

https://www.owasp.org/index.php/OWASP_Dependency_Check

https://github.com/snyk/snyk/releases

1.6.3.CSRF 保护

规则 6.3.1：Spring Security 自带 CSRF 支持，默认情况下处于启用状态，使用 Spring MVC 的<form:form>标记或 Thymeleaf 和@EnableWebSecurity ，则 CSRF 标记将自动添加为隐藏输入字段。

如果使用的是像 Angular 或 React 这样的 JavaScript 框架，则需要配置 CookieCsrfTokenRepository 以便 JavaScript 可以读取 cookie：

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf()
                .csrfTokenRepository(CookieCsrfTokenRepository.withHttp
OnlyFalse());
    }
}
```

1.6.4.防止 XSS

规则 6.4.1：可以配置应用程序以返回 Content-Security-Policy 头，来开启内容安全策略 （CSP）来缓解 XSS（跨站点脚本）和数据注入攻击，Spring Security 默认不添加 CSP，可以使用以下配置在 Spring Boot 应用程序中启用 CSP 标头：

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.headers()
            .contentSecurityPolicy("script-src 'self'
https://trustedscripts.example.com; object-src
https://trustedplugins.example.com; report-uri /csp-report-endpoint/");
```

```
    }
}
```

1.6.5.OpenID 验证

规则 6.5.1：使用 OpenID Connect 进行身份验证，OpenID Connect（OIDC）是一个 OAuth 2.0 扩展，提供用户信息， 除了访问令牌之外，它还添加了 ID 令牌，以及可以从中获取其他信息的/userinfo 端点。 它还添加了端点发现功能和动态客户端注册。

1.6.6.加密算法

规则 6.6.1：Spring Security 默认提供了三种密码存储方式：BCryptPasswordEncoder 和 Pbkdf2PasswordEncoder 等，MD5+Salt 的自定义的加密方式可以在 configure(HttpSecurity http) 后面添加 configure(AuthenticationManagerBuilder auth)来实现。

1.6.7.最新版本

规则 6.7.1：Spring Security 及其依赖库也是会有漏洞的，因此必须更新到无漏洞的版本，和支持版本兼容更新机制。

1.6.8.数据存储

规则 6.8.1：对一些敏感数据，需要加密存储，不允许明文存储。

1.6.9.安全测试

规则 6.9.1：需要对应用系统进行安全测试，针对 Web 安全漏洞和业务逻辑漏洞等。

1.6.10. 源码审计

规则 6.10.1：需要对源码进行安全审计，针对 Web 安全漏洞和业务逻辑漏洞、安全配置项、后门情况等进行分析。

1.7. 组件安全

针对主要的组件 Spring、Hibernate、MyBatis、Struts2、jQuery 等，安全设计，安全编码要求请参考：

1.7.1.Spring Security

规则 7.1.1：注册功能的执行过程中，必须对用户的输入进行安全验证。

规则 7.1.2：验证机制执行过程中，必须对用户越权访问进行安全验证。

规则 7.1.3：对会话管理、用户信息处理等核心安全必须进行记录。

规则 7.1.4：通过 OAuth2 实现使用 JSON 令牌时，必须对用户越权访问进行安全验证。

规则 7.1.5：使用 REST API 时，必须进行身份验证。

1.7.2.Hibernate

规则 7.2.1：不要直接使用 SQL 语句拼接，会导致 SQL 注入漏洞发生：

List results = session.createQuery("from Orders as orders where orders.id = " + currentOrder.getId()).list();

List results = session.createSQLQuery("Select * from Books where author = " + book.getAuthor()).list();

规则 7.2.1：使用参数化查询方式，安全的示例如下：

/* Positional parameter in HQL */

Query hqlQuery = session.createQuery("from Orders as orders where orders.id = ?");

List results = hqlQuery.setString(0, "123-ADB-567-QTWYTFDL").list();

/* named parameter in HQL */

Query hqlQuery = session.createQuery("from Employees as emp where emp.incentive > :incentive");

List results = hqlQuery.setLong("incentive", new Long(10000)).list();

/* named parameter list in HQL */

List items = new ArrayList();

items.add("book"); items.add("clock"); items.add("ink");

List results = session.createQuery("from Cart as cart where cart.item in (:itemList)").setParameterList("itemList", items).list();

/* JavaBean in HQL */

Query hqlQuery = session.createQuery("from Books as books where book.name = :name and book.author = :author");

List results = hqlQuery.setProperties(javaBean).list(); //assumes javaBean has getName() & getAuthor() methods.

/* Native-SQL */

Query sqlQuery = session.createSQLQuery("Select * from Books where author = ?");

List results = sqlQuery.setString(0, "Charles Dickens").list();

1.7.3.MyBatis

规则 7.3.1：不要直接使用 ${} 语法拼接，会导致 SQL 注入漏洞发生：

<select id="getPerson" parameterType="string" resultType="org.application.vo.Person">

SELECT * FROM PERSON WHERE NAME = #{name} AND PHONE LIKE

'${phone}';

</select>

规则 7.3.2：使用 #{} 语法生成 PreparedStatement 参数方式，安全的示例如下：

<select id="getPerson" parameterType="int" resultType="org.application.vo.Person">

SELECT * FROM PERSON WHERE ID = #{id}

</select>

类似：

/* Comparable JDBC code */

String selectPerson = "SELECT * FROM PERSON WHERE ID = ?";

PreparedStatement ps = conn.prepareStatement(selectPerson);

ps.setInt(1, id);

1.7.4.JPA

规则 7.4.1：不要直接使用 SQL 语句拼接，会导致 SQL 注入漏洞发生：

List results = entityManager.createQuery("Select order from Orders order where order.id = " + orderId).getResultList();

List results = entityManager.createNativeQuery("Select * from Books where author = " + author).getResultList();

int resultCode = entityManager.createNativeQuery("Delete from Cart where itemId = " + itemId).executeUpdate();

规则 7.4.2：使用参数绑定方式，安全的示例如下：

/* positional parameter in JPQL */

Query jpqlQuery = entityManager.createQuery("Select order from Orders order where order.id = ?1");

List results = jpqlQuery.setParameter(1, "123-ADB-567-QTWYTFDL").getResultList();

/* named parameter in JPQL */

Query jpqlQuery = entityManager.createQuery("Select emp from Employees emp where emp.incentive > :incentive");

List results = jpqlQuery.setParameter("incentive", new Long(10000)).getResultList();

/* named query in JPQL - Query named "myCart" being "Select c from Cart c where c.itemId = :itemId" */

Query jpqlQuery = entityManager.createNamedQuery("myCart");

List results = jpqlQuery.setParameter("itemId", "item-id-0001").getResultList();

/* Native SQL */

Query sqlQuery = entityManager.createNativeQuery("Select * from Books where author = ?", Book.class);

List results = sqlQuery.setParameter(1, "Charles Dickens").getResultList();

1.7.5.SQL

规则 7.5.1：不要直接使用 SQL 语句拼接，会导致 SQL 注入漏洞发生：

// 示例 #1

String query = "SELECT * FROM users WHERE userid ='"+ userid + "'" + " AND password='" + password + "'";

Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(query);

// 示例 #2

String query = "SELECT * FROM users WHERE userid ='"+ userid + "'" + " AND password='" + password + "'";

PreparedStatement stmt = connection.prepareStatement(query);

ResultSet rs = stmt.executeQuery();

规则 7.5.2：使用参数化查询方式，安全的示例如下：

PreparedStatement stmt = connection.prepareStatement("SELECT * FROM users WHERE userid=? AND password=?");

stmt.setString(1, userid);

stmt.setString(2, password);

ResultSet rs = stmt.executeQuery();

1.7.6.Struts2

规则 7.6.1：限制对 Config Browser 插件的访问，设置验证。

规则 7.6.2：不要在同一命名空间中混合使用不同的访问级别。

规则 7.6.3：不要直接暴露 JSP 文件，可以通过将所有 JSP 文件放在 WEB-INF 文件夹下，或向 web.xml 文件添加安全性约束。

规则 7.6.4：禁止 devMode，可以通过 struts.xml 配置：

<constant name ="struts.devMode" value="false" />

规则 7.6.5：降低日志记录级别，不要设置为 DEBUG 级别，设置 WARN 或至少 INFO 级别。

规则 7.6.6：使用 UTF-8 编码，使用 JSP 时请将以下标头添加到每个 JSP 文件中：

<%@ page contentType="text/html; charset=UTF-8" %>

规则 7.6.7：不要定义不需要的 setter，可能会导致表达式注入问题。

规则 7.6.8：不要将传入值用作本地化逻辑的输入，所有 TextProvider 的 getText（...）方法（例如 inActionSupport）都会对消息中包含的参数进行评估，以正确本地化文本。 这意味着使用带有 getText（...）方法的传入请求参数具有潜在的危险性，应该避免使用。以下代码会导致 OGNL 表达式注入：

public String execute() throws Exception {

```
        setMessage(getText(getMessage()));
        return SUCCESS;
    }
```

规则 7.6.9：使用 Struts 标记而不是原始 EL 表达式，切勿对传入值使用原始 $ {} EL 表达式，这会导致代码注入问题。

规则 7.6.10：不要更改内置的安全管理器，它阻止对特定类和 Java 包的访问，它是一个 OGNL 范围的机制，这意味着它影响框架的任何方面。

规则 7.6.11：禁止从表达式访问静态方法，这会导致代码注入问题。

规则 7.6.12：OGNL 调用 action 方法时，不要在层次结构中使用相同方法的名称，只需将操作的方法从 save（）更改为 saveAction（）并保留注释，以允许通过/save.action 请求调用此操作。

规则 7.6.13：接受/排除的模式，从 2.3.20 版本开始，框架提供了两个新接口，用于接受/排除参数名称和值 - 带默认实现的 AcceptedPatternsChecker 和 ExcludedPatternsChecker 。 可以使用 Parameters Interceptor 和 Cookie Interceptor 这两个接口来检查 param 是否可以被接受或必须被排除。

规则 7.6.14：严格的方法调用，在 2.5 版本中引入这种机制，它允许通过动态方法调用使用“！”运算符控制可以访问的方法。

1.7.7.jQuery

规则 7.7.1：使用 jQuery 3.*以上无漏洞版本。

规则 7.7.2：对于 jQuery 2.*的版本，使用 jQuery-Form-Validator 插件进行输入验证：

https://github.com/victorjonsson/jQuery-Form-Validator/

1.7.8.AJAX

规则 7.8.1：使用.innerText 而不是.innerHtml，使用.innerText 可以防止大多数 XSS 问题，因为它会自动编码文本。

规则 7.8.2：禁止使用 eval。

规则 7.8.3：当使用数据构建 HTML，脚本，CSS，XML，JSON 等时，确保对数据进行适当编码，以防止注入样式问题。

规则 7.8.4：不要依赖客户端逻辑来提高安全性，客户端可以使用一些浏览器插件来设置断点，跳过代码，更改值等。

规则 7.8.5：不要依赖客户端业务逻辑，确保在服务器端验证逻辑。

规则 7.8.6：避免编写序列化代码。

规则 7.8.7：避免动态构建 XML 或 JSON，可能导致 XML 注入错误，如果必须用，要使用编码库或安全的 JSON 或 XML 库来保证属性和元素数据的安全。

规则 7.8.8：不要把敏感信息存储在客户端本地。

规则 7.8.9：不要在客户端代码中执行加密，使用 TLS / SSL 并在服务器上加密.

规则 7.8.10：防止旧版浏览器的 JSON 劫持，可以提醒用户更新到新版浏览器。

1.8. Java 语言

针对 Java 语言的安全编码规范主要包括对各种安全漏洞缓解措施进行安全编码的指引。

1.8.1.输入过滤

通常针对用户输入的过滤，简单代码示例如下:

```
private Pattern regexPattern;
private Matcher regMatcher;

public String validateEmailAddress(String emailAddress) {
    regexPattern                                          =
```

```
Pattern.compile("^[(a-zA-Z-0-9-\\_\\+\\.)]+@[(a-z-A-z)]+\\.[(a-zA-z)]{2,3}$");
            regMatcher      = regexPattern.matcher(emailAddress);
            if (regMatcher.matches()) {
                return "Valid Email Address";
            } else {
                return "Invalid Email Address";
            }
        }

        public String validateMobileNumber(String mobileNumber) {
            regexPattern = Pattern.compile("^\\+[0-9]{2,3}+-[0-9]{10}$");
            regMatcher      = regexPattern.matcher(mobileNumber);
            if (regMatcher.matches()) {
                return "Valid Mobile Number";
            } else {
                return "Invalid Mobile Number";
            }
        }
```

1.8.2.安全工具类

安全工具类提供一套安全组件接口，实现用户输入过滤和输出编码等针对恶意输入可能产生的漏洞进行缓解和代码加固，同时还有公开的 Java 安全框架可以参考：

框架	URL
Spring Security	https://projects.spring.io/spring-security/
ESAPI	https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API

Apache Commons Validator	http://commons.apache.org/proper/commons-validator/
Hibernate Validator	http://hibernate.org/validator/
Apache Santuario	http://santuario.apache.org/
Jasypt	http://www.jasypt.org/
Apache Shiro	https://shiro.apache.org/
HDIV	https://hdivsecurity.com/ https://github.com/hdiv/hdiv

1.8.3.SQL 注入

针对 Hibernate、MyBatis、JPA、SQL 等数据库操作表单，不要直接使用 SQL 语句拼接，会导致 SQL 注入漏洞，需使用参数化查询方式避免漏洞产生，对代码加固建议，请对用户提交的 GET、POST、Cookie 等数据进行非法关键字过滤，并封装简约的错误返回码，比如：234561。

针对 GET 过滤的正则参考如下：

<[^>]*?=[^>]*?&#[^>]*?>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\()|<[^>]*?\\b(onerror|onmousemove|onload|onclick|onmouseover)\\b[^>]*?>|^\\+\/v(8|9)|\\b(and|or)\\b\\s*?([\\(\\)'\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\w]+?\s+?\bin\b\s*?\(|\blike\b\s+?[\"'])|\/*.+?*\/|<\s*script\b|\bEXEC\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\s+(TABLE|DATABASE)"

针对 POST 过滤的正则参考如下：

<[^>]*?=[^>]*?&#[^>]*?>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\()|<[^>]*?\\b(onerror|onmousemove|onload|onclick|onmouseover)\\b[^>]*?>|\\b(a

nd|or)\\b\\s*?([\\(\\)'\\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"']|\\/*.+?*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)"

针对 Cookie 过滤的正则参考如下：

\\b(and|or)\\b\\s*?([\\(\\)'\\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"']|\\/*.+?*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)"

1.8.4.跨站脚本

针对 JavaScript、JQuery、IFrame 等客户端代码和组件，不要直接输出“’、”、<、>、(、=、.”等特殊字符到页面，会导致跨站脚本（XSS）漏洞，需要对特殊字符进行转义或者编码避免漏洞产生，对于代码加固建议，对用户提交的 GET、POST、Cookie 等数据进行非法关键字过滤。

针对 GET 过滤的正则参考如下：

<[^>]*?=[^>]*?&#[^>]*?>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\()|<[^>]*?\\b(onerror|onmousemove|onload|onclick|onmouseover)\\b[^>]*?>|^\\+\\/v(8|9)|\\b(and|or)\\b\\s*?([\\(\\)'\\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"']|\\/*.+?*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)"

针对 POST 过滤的正则参考如下：

<[^>]*?=[^>]*?&#[^>]*?>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\()|<[^>]*?\\b(onerror|onmousemove|onload|onclick|onmouseover)\\b[^>]*?>|\\b(and|or)\\b\\s*?([\\(\\)'\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"'])|\\/*.+?*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)"

针对 Cookie 过滤的正则参考如下:

\\b(and|or)\\b\\s*?([\\(\\)'\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"'])|\\/*.+?*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)"

示例参考:

JSTL Tag: <c:out value="${bean.userControlledValue}"/>

Spring Form Tags: <form:hidden path="${bean.userControlledValue}" htmlEscape="true" />

Spring Eval: <spring:eval expression="bean.userControlledValue" htmlEscape="true"/>

Struts 1: <bean:write name="bean" property="userControlledValue" filter="true"/>

Struts 2: <s:property value="#request.bean.userControlledValue" escapeHtml='true'/>

1.8.5.任意文件上传

针对文件上传功能，需采用白名单形式限制文件上传的后缀名称，只允许上传“jpg、png、gif、bmp 、doc、docx、rar、zip”等

非 Web 脚本执行文件避免被恶意上传 Webshell 后门。

Java 示例代码:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.*,java.io.*" %>
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page import="org.apache.commons.fileupload.*" %>
<%@ page import="org.apache.commons.fileupload.disk.*" %>
<%@ page import="org.apache.commons.fileupload.servlet.*" %>
<%
//文件保存目录路径
String savePath = pageContext.getServletContext().getRealPath("/") + "uploads/";
//文件保存目录 URL
String saveUrl  = request.getContextPath() + "/uploads/";
//定义允许上传的文件扩展名
String extStr = "gif,png,jpg,bmp,doc,docx,xls,xlsx,ppt,pptx";
//最大文件大小
long maxSize = 20000000;
response.setContentType("text/html; charset=UTF-8");
if(!ServletFileUpload.isMultipartContent(request)){
 out.println("请选择文件。");
 return;
}
//检查目录
File uploadDir = new File(savePath);
if(!uploadDir.isDirectory()){
 out.println("上传目录不存在。");
```

```
  return;
}
//检查目录写权限
if(!uploadDir.canWrite()){
  out.println("上传目录没有写权限。");
  return;
}
File saveDirFile = new File(savePath);
if (!saveDirFile.exists()) {
  saveDirFile.mkdirs();
}
FileItemFactory factory = new DiskFileItemFactory();
ServletFileUpload upload = new ServletFileUpload(factory);
upload.setHeaderEncoding("UTF-8");
List items = upload.parseRequest(request);
Iterator itr = items.iterator();
while (itr.hasNext()) {
  FileItem item = (FileItem) itr.next();
  String fileName = item.getName();
  long fileSize = item.getSize();
  if (!item.isFormField()) {
    //检查文件大小
    if(item.getSize() > maxSize){
      out.println("上传文件大小超过限制。");
      return;
    }
    //检查扩展名
    String fileExt = fileName.substring(fileName.lastIndexOf(".") +
```

```
1).toLowerCase();
        if(!Arrays.<String>asList(extStr.split(",")).contains(fileExt)){
          out.println("上传文件扩展名是不允许的扩展名。\n 只允许" + extStr + "
格式。");
          return;
        }

        SimpleDateFormat df = new SimpleDateFormat("yyyyMMddHHmmss");
        String newFileName = df.format(new Date()) + "_" + new
Random().nextInt(100000) + "." + fileExt;
        try{
          File uploadedFile = new File(savePath, newFileName);
          item.write(uploadedFile);
        }catch(Exception e){
          out.println("上传文件失败。");
          return;
        }
        out.println("上传成功，十分感谢|upload success,thanks...");
       }
      }
      %>
```

1.8.6.XML 实体注入

针对 XML 操作，不要直接使用字符串拼接来构建 XML 查询，会导致 XML 注入漏洞，需对用户的输入进行过滤避免漏洞产生，或使用文档类型定义（DTD）或模式对其进行验证，对于代码加固建议参考如下：

```
import java.io.BufferedOutputStream;
```

```
import java.io.ByteArrayOutputStream;
import java.io.IOException;

public class OnlineStore {
  private static void createXMLStream(final BufferedOutputStream outStream,
      final String quantity) throws IOException, NumberFormatException {
    // 仅当数量是无符号整数（count）时才写入 XML 字符串。
    int count = Integer.parseUnsignedInt(quantity);
    String xmlString = "<item>\n<description>Widget</description>\n"
        + "<price>500</price>\n" + "<quantity>" + count + "</quantity></item>";
    outStream.write(xmlString.getBytes());
    outStream.flush();
  }
}
```

1.8.7.EL 表达式注入

针对表达式语言解释器，不要直接使用字符串拼接来构建 EL 表达式查询，会导致 EL 表达式注入漏洞，需对用户的输入进行过滤避免漏洞产生，要检测的参数主要包括“${”和“#{”，对代码加固建议，请对用户提交的 GET、POST、Cookie 等数据进行非法关键字过滤，并封装简约的错误返回码，比如：234561。

针对 GET 过滤的正则参考如下：

```
<[^>]*?=[^>]*?&#[^>]*?>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\()|<[^>]*?\\b(onerror|onmousemove|onload|onclick|onmouseover)\\b[^>]*?>|^\\+\/v(8|9)|\\b(and|or)\\b\\s*?([\\(\\)'\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\w]+?\s+?\bin\b\s*?\(|\blike\b\s+?[\"'])|\/\*.+?\
```

*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)"

针对 POST 过滤的正则参考如下：

<[^>]*?=[^>]*?&#[^>]*?>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\()|<[^>]*?\\b(onerror|onmousemove|onload|onclick|onmouseover)\\b[^>]*?>|\\b(and|or)\\b\\s*?([\\(\\)'\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"']|\\/*.+?*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)"

针对 Cookie 过滤的正则参考如下：

\\b(and|or)\\b\\s*?([\\(\\)'\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"']|\\/*.+?*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)"

1.8.8.跨站请求伪造

针对一些关键页面如果没有加验证码或 Token 验证，可能会导致跨站请求伪造（CSRF）问题，请采用隐藏 FORM 元素，内容为随机数，存储在会话中，然后在服务端通过比较表单的隐藏变量是否与原来设置的一样，判断是否来自外面构造的提交。

通过 Token 的方式，也有现成的第三方库实现，示例说明如下：

（1）在用户刚登录的时候，产生一个新的不可预知的 CSRF Token，并且把此 Token 存放在用户的 session 中。

用法如下：

```
String username = ESAPI.randomizer().getRandomString(8,
EncoderConstants.CHAR_ALPHANUMERICS);
    DefaultUser user = (DefaultUser) ESAPI.authenticator().createUser(username,
password, password);
     String token = user.resetCSRFToken();
```

（2） 在任何一个需要保护的表单中，增加一个隐藏的字段来存放这个 Token；对于需要保护的 URL，增加一个参数来存放此 Token。

```
String username = ESAPI.randomizer().getRandomString(8,
EncoderConstants.CHAR_ALPHANUMERICS);
  User user = ESAPI.authenticator().createUser(username, "addCSRFToken",
"addCSRFToken");
  ESAPI.authenticator().setCurrentUser( user );
```

（3）验证请求的时候，在服务器端检查提交的 Token 与用户 session 中的 Token 是否一致，如果一致，继续处理请求，否则返回一个错误信息给用户。

ESAPI sample：ESAPI.httpUtilities().verifyCSRFToken(request);

```
MockHttpServletRequest request = new MockHttpServletRequest();
try {
ESAPI.httpUtilities().verifyCSRFToken(request);
fail();
} catch( Exception e ) {
    // expected
}
```

（4）在用户退出或者 session 过期的时候，用户信息（包括 CSRF Token）从 session 中移除并且销毁 session。

ESAPI sample：user = ESAPI.authenticator().logout();

```
DefaultUser user =
createTestUser(ESAPI.authenticator().generateStrongPassword());
```

```
user.logout();
```

1.8.9.文件路径越权

针对一些上传和下载文件的功能，未严格限制访问文件的相对路径可能导致文件越权访问，请采用白名单形式限制访问文件路径和目录，禁止../或..\访问。

通过现成的第三方库实现，示例说明如下：

```
import java.nio.file.Files;
import java.nio.file.Paths;

while ((ze = zis.getNextEntry()) != null) {
    String fileName = ze.getName();
    String esapiFileName = ESAPI.encoder().canonicalize(fileName);
    boolean esapiValidFileName = ESAPI.validator().isValidFileName("upload", esapiFileName, false);
    String _completefileNamePath = null;
    if (esapiValidFileName) {
        _completefileNamePath = _destination + esapiFileName;
        // optional: Files.createDirectories(Paths.get(_completefileNamePath).getParent());
        Files.copy(zis, Paths.get(_completefileNamePath));
        zis.closeEntry();
    }
}
```

1.8.10. 明确不信任项

针对通用的风险点，明确不能被信任的数据包括：

不安全的随机数

不能信任 servlet 参数

不能信任 Content-Type 头

不能信任 Hostname 头

不能信任 session cookie 值

不能信任 HTTP 头

不能信任 Referer 头

不能信任 User-Agent 头

不能信任 query 字符串

不要在 cookie 中包含敏感数据

不允许路径遍历

不允许命令注入

不要使用 FilenameUtils

TrustManager 要基于 keystore

不要使用 MD2, MD4、MD5、SHA-1

不要使用默认的 HttpClient

不要使用 SSLContext 的 SSL

不要禁用 EnableWebSecurity

不要使用 RequestMapping

不要直接拼接 SQL 语句

不要使用不安全加密算法

不要传入动态变量到表达式

不要禁用 XML 特殊字符转义

不要使用不安全的加密模式

不要禁用 Cookie 的 Secure 标识

不要禁用 Cookie 的 HttpOnly 标识

2. WEB 安全设计规范

2.1. 安全设计

2.1.1.适用范围

本规范的制定考虑了 xxxx 各种 Web 应用的共性，适合于 xxxx 绝大部分 Web 应用系统，要求 Web 应用系统开发必须遵循。

2.1.2.用词约定

规则：强制必须遵守的原则；

建议：需要加以考虑的原则；

说明：对此规则或建议进行相应的解释；

实施指导：对此规则或建议的实施进行相应的指导；

2.1.3.规范评审

根据研发安全流程控制活动要求，项目立项后，开发团队项目经理或指定接口人根据《需求分析说明书》编写《概要设计说明书》，再根据《概要设计说明书》编写《详细设计说明书》，同时，信息安全工作组对安全需求项和安全设计项进行评审。

2.1.4.设计规范

安全设计规范建立：安全设计规范由开发团队安全接口人提供咨询支持、信息安全工作主要维护并提供版本更新，结合 xxxx 具体业务系统共性，建立 Web 应用、移动 APP 两种架构的安全设计规范，主要包括安全技术实现细节和安全编码要求规范等。

2.1.5.规范分类

xxxx 研发安全流程所提安全设计规范主要包括针对：监管合规需求、安全功能需求、安全技术需求、最低需求等的安全建设对策；具体到安全性机制包括访问控制、授权、缓存控制、加密算法、错误处理、日志记录、会话管理、输入验证等，安全特性规范等在业务系统中的落地进行具体要求和指引。

2.2. 监管合规和设计原则

安全设计规范对标的控制措施包括但不限于：

网络安全等级保护相关要求

ISO27001:2013 信息安全管理体系

网上银行系统信息安全通用规范

中国人民银行网上银行系统信息安全通用规范

金融行业信息系统信息安全等级保护实施指引

中国金融移动支付远程支付安全相关技术规范

支付应用程序数据安全标准（PCI DSS）

2.2.1.原则适用性

安全原则是系统安全的最基本宗旨。所有系统安全基线的制定、系统的安全设计、开发和使用都必须符合安全基本原则。

2.2.2.不信任原则

应该严格限制对用户、外部系统的信任，应该假设系统环境是不安全的。

说明：在应用程序开发时，不能假定用户输入是合法可信，同时，没有任何的系统是绝对安全的，因此保护两个系统之间的接口，对外部系统的信任降至最低是非常有必要的。

2.2.3.最小授权原则

只授予每个用户、接口在执行操作时所必需的最小特权。

说明：该原则限制了事故、错误或攻击带来的危害，减少了特权程序之间潜在的相互作用，从而使对特权无意的、没必要的或不适当的使用不太可能发生。

2.2.4.异常处理原则

考虑到发生故障的必然性并设计异常处理机制，当系统发生异常时，对任何请求默认应加以拒绝。

2.2.5.弹性防御原则

应该采用多层的安全机制去保护系统安全。

说明：在攻击者导致一个安全机制失效后，其他的安全机制依然可以保护系统安全。

2.2.6.权限分离原则

不允许基于单一条件的授权操作过程。

说明：此原则的目的是使得攻击者只在得到一项权限而无法得到另一项的情况下不能发起有效的攻击，如双因素身份认证、禁止 root 用户远程登录、职责分离等。

2.2.7.经济机制原则

保证系统的设计和代码尽可能简单、紧凑。

说明：系统流程越复杂，代码中出现 Bug 的几率越高，如果代码尽可能简短，那么出错几率也小。

2.2.8.公开设计原则

安全保护机制不应该依赖于攻击者对实现机制的无知，而只应该依赖于像口令或密钥这样比较少的、且容易改变的项目的保密。

说明：反编译和反汇编等技术能够使攻击者获得程序的实现原理和二进制文件的敏感信息。

2.2.9.安全中介原则

对受保护对象的每一个访问都应当被检查。

2.2.10. 心理可接收原则

不应该通过限制资源访问的方式去阻止攻击。

说明：如果安全机制妨碍了资源的可用性或系统的易用性，那么用户很可能会关闭这些机制。安全机制应该尽可能对系统用户透明，或者引入少量的资源使用阻碍。

2.3. 安全漏洞

2.3.1.Web 安全漏洞

针对众多的 Web 漏洞，OWASP 的专家们结合各自在各领域的应用安全工作经验及智慧，提出了十大 Web 应用程序安全风险，帮助人们关注最严重的漏洞。（OWASP 即开放 Web 应用安全项目，是一个旨在帮助人们理解和提高 Web 应用及服务安全性的项目组织。）xxxx 维护的威胁风险库中，Web 安全漏洞基于 OWASP１０大应用风险的细分项：

序号	风险名称	风险描述

1	SQL 注入	攻击者利用注入漏洞，通过构造特殊的语句，可进行后台数据库操作并插入木马，以获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
2	跨站脚本	攻击者利用此漏洞，可以获取其他合法用户的 Cookie 身份信息、访问地址等，通过获取到的 Cookie 信息，即可以被攻击者的身份访问 Web 应用，如获取到管理员的 Cookie，就可以以管理员的身份访问应用系统。
3	任意文件上传	攻击者利用此漏洞，可直接上传木马文件，以获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
4	XML 实体注入	攻击者利用此漏洞，通过构造特殊的引用或请求可以读取服务器文件或执行操作系统命令，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
5	EL 表达式注入	攻击者利用 EL 表达式的方法，EL 表达式语法允许开发人员开发自定义函数，以调用 Java 类的方法。语法：${prefix：method(params)}。在 EL 表达式中调用的只能是 Java 类的静态方法，这个 Java 类的静态方法需要在 TLD 文件中描述，才可以被 EL 表达式调用。EL 自定义函数用于扩展 EL 表达式的功能，可以让 EL 表达式完成普通 Java 程序代码所能完成的功能。从而攻击者可以构造代码执行，获取系统的执行命令权限。如果这些信息有助于攻击者进一步攻击利用，可能导致更严重的危害。（存在风险点的位置：用于表达式的运算。如：加、减、乘、除,用于从作用域中取出数据）
6	账号弱口令	攻击者利用弱口令，可以获取特定账户或应用的访问控制权限，如果进一步攻击利用可能获取服务器控制权限。
7	跨站请求伪造	攻击者利用此漏洞，可以以受害者的身份发起 HTTP 请求，访问 Web 应用的相应功能，如果利用成功，那么攻击者就可以更新用户的相关数据（添加、删除、修改），如添加管理员账户、向指定账户转账。

8	文件包含	攻击者利用此漏洞，可以执行任意代码，获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
9	目录遍历	攻击者利用此漏洞，可以操作服务器指定文件，特别是一些敏感文件，如程序配置文件、数据库配置文件、程序代码文件、服务器账户文件等；如果进一步利用可获取服务器敏感数据或获取服务器控制权限。
10	服务端请求伪造	攻击者利用此漏洞，可以利用 Web 服务器发起请求（HTTP、FTP、HTTPS 等等），可以突破内外网访问控制、部分安全控制措施，如果进一步利用可获取内网服务器权限及敏感数据。
11	Java 反序列化	攻击者利用此漏洞，通过构造特殊的引用或请求可以读取服务器文件或执行操作系统命令，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。（导入模版文件、网络通信、数据传输、日志格式化存储、对象数据落磁盘或 DB 存储等业务场景等）。
12	有漏洞的组件	攻击者利用此漏洞，可执行恶意操作系统命令，获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
13	敏感信息泄漏	攻击者利用此漏洞，可以访问到程序备份文件，可能导致源代码或者数据泄露，如果进一步被利用可能导致更严重的危害。
14	不安全加密算法	攻击者利用此漏洞，可以获取到对应的铭感信息进行下一步的攻击。
15	任意文件下载	攻击者利用此漏洞，可以下载服务器任意文件，如脚本代码，服务及系统配置文件等；可用得到的代码进一步代码审计，得到更多可利用漏洞。
16	任意密码找回	攻击者利用此漏洞，可以找回任意账户的密码。
17	HTML 注入	攻击者利用此漏洞，可以访问引导用户访问恶意的网页，如果这个网页是钓鱼、挂马的网页，可能导致更严重的危害。

18	隐藏暗链	暗链对政府网站来讲危害更大，当用户通过搜索引擎搜索某地政府网站时，可能从搜索引擎的描述中看到一些非法的关键词，从而影响政府的形象。境外敌对势力甚至可以根据政府网站是否存在暗链来判断一个政府网站是否存在安全漏洞，并决定是否发起攻击。成功入侵后，可以发布虚假政策信息造成群众对政府的不信任，制造政府与群众之间的矛盾，引发社会事件。
19	未授权访问	攻击者利用此漏洞，可以直接访问相应的页面，如果此页面存在敏感数据或信息，那么这些敏感数据或信息可能发生泄露，如果这些信息有助于攻击者进一步攻击利用，可能导致更严重的危害。
20	越权访问	攻击者利用此漏洞，可以访问其他用户的数据，可能导致数据泄露，如果进一步被利用可能导致更严重的危害。
21	固定 SessionID 漏洞	攻击者可以简单的伪造一个 SessionID 诱使用户使用该 SessionID 登录，即可获取登录权限。
22	URL 重定向	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。
23	Flash 访问	攻击者利用此漏洞，可以以受害者的身份发起 HTTP 请求，访问 Web 应用的相应功能，如果利用成功，那么攻击者就可以更新用户的相关数据（添加、删除、修改），如添加管理员账户、向指定账户转账。

2.3.2.业务场景漏洞

业务场景漏洞基于业务模块功能可能产生的漏洞细分项：

序号	风险名称	风险描述

1	用户登录	攻击者通过用户登录模块漏洞，可以获取特定账户或应用的访问控制权限，如果进一步攻击利用可能获取服务器控制权限。
2	用户注册	攻击者通过用户注册模块漏洞，可以通过恶意注册，结合其他攻击手段进一步利用并扩大漏洞危害。
3	密码找回	攻击者利用找回密码用其他攻击进行下一步的攻击操作，甚至攻击成功的话，可能导致获取网站用户登录的部分权限，包括修改删除用户信息、窃取用户敏感信息。
4	后台管理	攻击者利用后台管理漏洞，可以通过登录后台或者 url 跳转等操作，对服务器指定文件，特别是一些敏感文件，如程序配置文件、数据库配置文件、程序代码文件、服务器账户文件等，从而达到下一步的攻击。
5	接口调用	可以通过攻击接口，获取敏感的数据或者能访问一些访问权限。
6	会员系统	通过攻击会员系统，可以越权访问权限，个人信息，图片上传，从而获取下一步的攻击，结合其他安全漏洞可能造成更严重的危害。
7	业务办理	攻击者通过攻击业务办理模块，可能存在替代办理、业务绕过、篡改他人的信息。
8	业务查询	攻击者利用此漏洞，可能会造成恶意查询或者可能办理人的信息泄露，如果进一步被利用可能导致更严重的危害。
9	业务传输	攻击者通过攻击 cookie，劫持 cookie，替换会话等攻击手段，从而进一步攻击利用可能获取服务器控制权限。

10	发表评论	攻击者利用发表评论漏洞，可能访问其他用户的数据，可能导致数据泄露，如果进一步被利用可能导致更严重的危害。
11	购买支付	攻击者利用此漏洞，可以篡改商品的订单，金额，遍历交易信息。如果进一步篡改可导致平台上损失。
12	账号充值	攻击者通过账号充值，对充值功能模块进行虚拟充值，篡改充值，篡改账号等，从而导致充值业务被破坏。
13	抽奖活动	攻击者可以通过抽奖活动，进行薅羊毛，盗刷积分，刷取奖品等攻击行为，从而导致正常的抽奖活动被破坏。
14	代金优惠	攻击者通过攻击代金优惠，可以进行对代金业务进行盗刷，篡改等，甚至会给平台带来一定的损失。
15	订单信息	如果订单泄露，可能会造成个人的敏感信息泄露。如果进一步被利用可能导致更严重的危害。
16	运费账单	攻击者利用运费账单漏洞，通过运费的漏洞利用，篡改或者构造假账单，从而让平台造成一定的损失。
17	第三方商家	攻击者利用平台上的第三方商家，可能会造成第三方商家被盗号，商家账号被遍历，或者越权访问其他商家的用户信息，甚至可能窃取用户敏感信息。

2.3.3.攻击行为风险

业务攻击风险基于业务场景功能可能产生的风险细分项：

序号	风险名称	风险描述
1	短信验证	攻击者利用验证码爆破漏洞，通过构造的密码字典，对用户的验证码进行暴力猜测，可以获取网站用户登录的部分权限，包括修改删除用户信息、窃取用户敏感信息。(存在风险点的位置：登录处、注册处)。

2	用户撞库	攻击者撞库成功后,可以获取网站用户登录的部分权限，包括修改删除用户信息、窃取用户敏感信息。(存在风险点的位置：登录处、注册处)
3	爆破密码	攻击者利用弱口令漏洞，通过进行暴力猜解,获取网站管理权限，包括修改删除网站页面、窃取数据库敏感信息、植入恶意木马；甚至以网站为跳板，获取整个内网服务器控制权限。
4	ID 遍历	攻击者利用该漏洞，通过遍历个人信息，可能获取网站帐户等敏感信息;通过结合其他漏洞可能造成更严重的危害。（存在风险点位置:用户个人信息，订单信息等。）
5	风控缺失	攻击者利用此漏洞,通过脚本和程序进行批量的用户注册和登录操作,从而进行下一步的攻击，如薅羊毛等攻击。（存在分析点位置：登录处，密码找回处，用户注册处。）

2.4. 应用安全

应用安全性机制包括访问控制、授权、缓存控制、加密算法、错误处理、日志记录、会话管理、输入验证等，安全特性规范将针对安全性机制在业务系统中的落地进行具体要求。

2.4.1.Web 部署规范

规则 4.1.1：如果 Web 应用对 Internet 开放，Web 服务器应当置于 DMZ 区，在 Web 服务器与 Internet 之间，Web 服务器与内网之间应当有防火墙隔离，并设置合理的策略。

规则 4.1.2：如果 Web 应用对 Internet 开放，Web 服务器应该部署在其专用的服务器上，应避免将数据库服务器或其他核心应用与 Web 服务器部署在同一台主机上。

说明：Web 服务器比较容易被攻击，如果数据库或核心应用与 Web 服务器

部署在同一台主机，一旦 Web 服务器被攻陷，那么数据库和核心应用也就被攻击者掌控了。

规则 4.1.3：Web 站点的根目录必须安装在非系统卷中。

说明：Web 站点根目录安装在非系统卷，如单独创建一个目录/home/Web 作为 Web 站点根目录，能够防止攻击者使用目录遍历攻击访问系统工具和可执行文件。

建议 4.1.1：Web 服务器与应用服务器需物理分离(即安装在不同的主机上)，以提高应用的安全性。

建议 4.1.2：如果 Web 应用系统存在不同的访问等级（如个人帐号使用、客户服务、管理），那么应该通过不同的 Web 服务器来处理来自不同访问等级的请求，而且 Web 应用应该鉴别请求是否来自正确的 Web 服务器。

说明：这样便于通过防火墙的访问控制策略和 Web 应用来控制不同访问等级的访问，比如通过防火墙策略控制，只允许内网访问管理 Portal。

建议 4.1.3：对于“客户服务”和“管理”类的访问，除了普通的认证，还应该增加额外的访问限制。

说明：额外的访问限制，可以限制请求来自企业内网，可以建立 VPN，或采用双向认证的 SSL；或采用更简单的办法，通过 IP 地址白名单对客户端的 IP 地址进行过滤判断。

2.4.2.身份验证

规则 4.2.1：关于 Web 应用及容器涉及到的口令，请遵循以下口令安全要求：

	要求概述	详细描述
口令安全策略管理	设置口令时，默认检测口令复杂度	系统默认检测口令复杂度，口令至少满足如下要求：
		1、口令长度至少 8 个字符（特权用户至少 12 个字符）；
		2、口令必须包含如下至少两种字符的组合：
		– 至少一个小写字母；
		– 至少一个大写字母；

		– 至少一个数字;
		– 至少一个特殊字符: `~!@#$%^&*()-_=+|[{}];:'",<.>/? 和空格
		3、口令不能和帐号或者帐号的倒写一样;
		若设置的口令不符合上述规则，必须进行警告。
	可设置口令出错锁定阈值	系统必须提供锁定用户的机制。可选择如下两种方式之一:
		方式一: 当重复输入错误口令次数（默认 3 次，次数系统可以设置）超过系统限制时，系统要锁定该用户。
		方式二: 系统还可以设置下次允许输入口令的间隔时间加倍，采用这种方式时，用户可以不设置自动锁定。
	可设置自动解锁时间（只适用于由于口令尝试被锁定的用户）	1、对于口令尝试 N 次失败被锁定的用户，系统要能够设置自动解锁时间，建议默认解锁时间为 5 分钟。
		2、用户被锁时间达到预定义时间，可自动解锁该用户，或者也可通过安全管理员手工解锁该用户。
		3、在锁定时间内，仅能允许应用安全管理员角色所属账号手动解锁该用户。
口令安全使用规则	操作界面中的口令不能明文显示	键入口令时不能明文显示出来(操作界面中的输入口令可不显示或用*代替)，包括在终端上打印或存储在日志中时也不能明文显示口令，即使是内存中的明文口令（如登录期间），也应在使用后立即覆盖。
	口令输入框内容禁止拷贝	口令输入框不支持拷贝功能。
	缺省口令符合复杂度要求	对于系统内置账号的缺省口令，口令应符合复杂度的要求，并在客户资料中提醒用户修改。
	用户可修改自己的口令	1、用户修改自己口令时必须验证旧口令;
		2、不允许修改除自身账号以外的账号的口令（管理员除外）。
	口令不能在网络中明文传输	口令等认证凭证在传输过程中必须加密，使用高安全等级的加密算法。
	口令在本地存储时必须加密	1、口令不能够明文写入日志文件、配置文件以及 cookie 中;
		2、口令文件必须设置访问控制，普通用户不能读取或拷贝加密的内容。

	提供账号口令清单	产品配套资料提供清晰的账号、口令清单。

规则 4.2.2：关于 Web 应用及容器涉及到的认证，请遵循以下认证安全要求：

规则 4.2.2.1：对用户的最终认证处理过程必须放到应用服务器进行。

说明：不允许仅仅通过脚本或其他形式在客户端进行验证，必须在应用服务器进行最终认证处理（如果采用集中认证，那么对用户的最终认证就是放在集中认证服务器进行）。

规则 4.2.2.2：网页上的登录/认证表单必须加入验证码。

说明：使用验证码的目的是为了阻止攻击者使用自动登录工具连续尝试登录，从而降低被暴力破解的可能。如果觉得验证码影响用户体验，那么可以在前 3 次登录尝试中不使用验证码，3 次登录失败后必须使用验证码。验证码在设计上必须要考虑到一些安全因素，以免能被轻易地破解。具体实现细节请查看 4.2.3 验证码。如图：

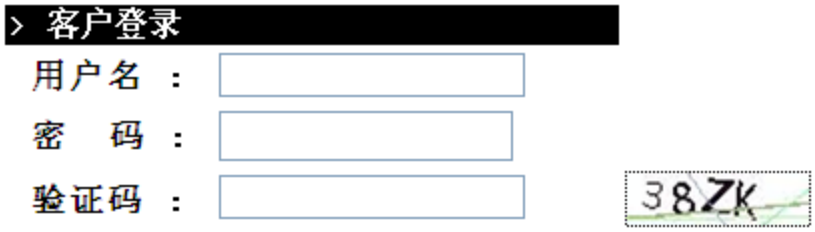

图1 验证码

实施指导：

建议使用安全工程部提供的验证码。

规则 4.2.2.3：用户名、密码和验证码必须在同一个请求中提交给服务器，必须先判断验证码是否正确，只有当验证码检验通过后才进行用户名和密码的检验，否则直接提示验证码错误。

说明：如果验证码和用户名、密码分开提交，攻击者就可以绕过验证码校验（如：先手工提交正确的验证码，再通过程序暴力破解），验证码就形同虚设，攻击者依然可以暴力破解用户名及口令。

规则 4.2.2.4：所有登录页面的认证处理模块必须统一。

说明：可以存在多个登录页面，但是不允许存在多个可用于处理登录认证请求的模块，防止不一致的认证方式。

规则 4.2.2.5：所有针对其他第三方开放接口的认证处理模块必须统一。

规则 4.2.2.6：认证处理模块必须对提交的参数进行合法性检查。

说明：具体输入校验部分请查看 5.1 输入校验。

规则 4.2.2.7：认证失败后，不能提示给用户详细以及明确的错误原因，只能给出一般性的提示。

说明：可以提示："用户名或者口令错误，登录失败"；不能提示："用户名不存在"、"口令必须是 6 位"等等。

规则 4.2.2.8：最终用户 portal 和管理 portal 分离。

说明：最终用户 portal 和管理 portal 分离，防止相互影响，防止来自用户面的攻击影响管理面。

实施指导：

将最终用户 portal 和管理 portal 分别部署在不同的物理服务器；如果为了解决成本合设（部署在同一台物理服务器上），那么，必须做到端口分离（通过不同的端口提供 Web 服务），一般的 Web 容器（如 tomcat）支持为不同的 Web 应用创建不同的端口。

规则 4.2.2.9：禁止在系统中预留任何的后门帐号或特殊的访问机制。

规则 4.2.2.10：对于重要的管理事务或重要的交易事务要进行重新认证，以防范会话劫持和跨站请求伪造给用户带来损失。

说明：重要的管理事务，比如重新启动业务模块；重要的交易事务，比如转账、余额转移、充值等。重新认证，比如让用户重新输入口令。

规则 4.2.2.11：用户名和密码认证通过后，必须更换会话标识，以防止会话固定（session fixation）漏洞。

实施指导：

场景一：对于从 HTTP 转到 HTTPS 再转到 HTTP（也就是仅在认证过程采用 HTTPS，认证成功后又转到 HTTP）的，在用户名和密码认证通过后增加以下行代码：

```
request.getSession().invalidate();
HttpSession newSession=request.getSession(true);
Cookie cookie = new Cookie("JSESSIONID",newSession.getId());
cookie.setMaxAge(-1);
cookie.setSecure(false);
cookie.setPath(request.getContextPath());
response.addCookie(cookie);
```

场景二：对于全程采用 HTTPS 协议，或者全程采用 HTTP 协议的，在用户名和密码认证通过后增加以下行代码：

```
request.getSession().invalidate();
request.getSession(true);
```

建议 4.2.2.1：管理页面建议实施强身份认证。

说明：如双因素认证、SSL 双向证书认证、生物认证等；还可以通过应用程序限制只允许某些特定的 IP 地址访问管理页面，并且这些特定的 IP 地址可配置。

建议 4.2.2.2：同一客户端在多次连续尝试登录失败后，服务端需要进行用户帐号或者是客户端所在机器的 IP 地址的锁定策略，且该锁定策略必须设置解锁时长，超时后自动解锁。

说明：

登录失败应该提示用户：如果重试多少次不成功系统将会锁定。在锁定期间不允许该用户帐号（或者客户端所在机器的 IP 地址）登录。

允许连续失败的次数（指从最后一次成功以来失败次数的累计值）可配置，取值范围为：0-99 次，0 表示不执行锁定策略，建议默认：5 次。

锁定时长的取值范围为：0-999 分钟，建议默认：30 分钟，当取值为 0 时，表示无限期锁定，只能通过管理员手动解锁（需要提供管理员对服务器锁定其它用户帐号/IP 进行解锁的功能界面）。建议优先使用帐号锁定策略。

注意：应用程序的超级用户帐号不能被锁定，只能锁定操作的客户端所在的 IP，这是为了防止系统不可用。特别说明：锁客户端 IP 策略存在缺陷，当用户使

用 proxy 上网时，那么锁定客户端 IP 会导致使用该 proxy 上网的所有用户在 IP 锁定期间都不能使用该 Web 应用；锁定用户帐户的策略也存在缺陷，当攻击者不断尝试某帐户的口令，就给该帐户带来拒绝服务攻击，使该帐户不可用。

规则 4.2.3：关于 Web 应用及容器涉及到的验证码，请遵循以下验证码安全要求：

规则 4.2.3.1：验证码必须是单一图片，且只能采用 JPEG、PNG 或 GIF 格式。

说明：验证码不能使用文本格式，不允许多图片组合（如用四个图片拼成的验证码）。

规则 4.2.3.2：验证码内容不能与客户端提交的任何信息相关联。

说明：在使用验证码生成模块时不允许接收来自客户端的任何参数，例如：禁止通过 getcode.jsp?code=1234 的 URL 请求，将 1234 作为验证码随机数。

规则 4.2.3.3：验证码模块生成的随机数不能在客户端的静态页面中的网页源代码里出现。

说明：在客户端网页上点击鼠标右键、选择“查看源文件”时，必须看不到验证码模块生成的随机数。

规则 4.2.3.4：验证码字符串要求是随机生成，生成的随机数必须是安全的。

说明：对于 java 语言可以使用类 java.security.SecureRandom 来生成安全的随机数。

规则 4.2.3.5：验证码要求有背景干扰，背景干扰元素的颜色、位置、数量要求随机变化。

规则 4.2.3.6：验证码在一次使用后要求立即失效，新的请求需要重新生成验证码。

说明：进行验证码校验后，立即将会话中的验证码信息清空，而不是等到生成新的验证码时再去覆盖旧的验证码，防止验证码多次有效；注意：当客户端提交的验证码为空，验证不通过。

2.4.3.会话管理

规则 4.3.1：使用会话 cookie 维持会话。

说明：目前主流的 Web 容器通过以下几种方式维持会话：隐藏域、URL 重写、持久性 cookie、会话 cookie，但通过隐藏域、URL 重写或持久性 cookie 方式维持的会话容易被窃取，所以要求使用会话 cookie 维持会话。如果条件限制必须通过持久性 cookie 维持会话的话，那么 cookie 信息中的重要数据部分如身份信息、计费信息等都必须进行加密。（cookie 有两种：会话 cookie 和持久性 cookie；会话 cookie，也就是非持久性 cookie，不设置过期时间，其生命期为浏览器会话期间，只要关闭浏览器窗口，cookie 就消失了；会话 cookie 一般不存储在硬盘上而是保存在内存里。持久性 cookie，设置了过期时间，被浏览器保存到硬盘上，关闭后再次打开浏览器，持久性 cookie 仍然有效直到超过设定的过期时间。）

规则 4.3.2：会话过程中不允许修改的信息，必须作为会话状态的一部分在服务器端存储和维护。

说明：会话过程中不允许修改的信息，例如，当用户通过认证后，其用户标识在整个会话过程中不能被篡改。禁止通过隐藏域或 URL 重写等不安全的方式存储和维护。对 JSP 语言，就是应该通过 session 对象进行存储和维护。

规则 4.3.3：当 Web 应用跟踪到非法会话，则必须记录日志、清除会话并返回到认证界面。

说明： 非法会话的概念就是通过一系列的服务端合法性检测（包括访问未授权资源，缺少必要参数等情况），最终发现的不是正常请求产生的会话。

规则 4.3.4：禁止使用客户端提交的未经审核的信息来给会话信息赋值。

说明：防止会话信息被篡改，如恶意用户通过 URL 篡改手机号码等。

规则 4.3.5：当用户退出时，必须清除该用户的会话信息。

说明：防止遗留在内存中的会话信息被窃取，减少内存占用。

实施指导：对于 JSP 或 java 语言使用如下语句：request.getSession().invalidate();

规则 4.3.6：必须设置会话超时机制，在超时过后必须要清除该会话信息。

说明：建议默认会话超时时间为 10 分钟（备注：对于嵌入式系统中的 Web，建议默认超时时间为 5 分钟，以减少系统资源占用）。如果没有特殊需求，禁止使用自动发起请求的机制来阻止 session 超时。

规则 4.3.7：在服务器端对业务流程进行必要的流程安全控制，保证流程衔接正确，防止关键鉴别步骤被绕过、重复、乱序。

说明：客户端流程控制很容易被旁路（绕过），因此流程控制必须在服务器端实现。

实施指导：可以通过在 session 对象中创建一个表示流程当前状态的标识位，用 0、1、2、3、…、N 分别表示不同的处理步骤，标识位的初始值为 0，当接收到步骤 N 的处理请求时，判断该标识位是否为 N-1，如果不为 N-1，则表示步骤被绕过（或重复或乱序），拒绝受理，否则受理，受理完成后更改标识位为 N。

规则 4.3.8：所有登录后才能访问的页面都必须有明显的“注销（或退出）”的按钮或菜单，如果该按钮或菜单被点击，则必须使对应的会话立即失效。

说明：这样做是为了让用户能够方便地、安全地注销或退出，减小会话劫持的风险。

规则 4.3.9：如果产品（如嵌入式系统）无法使用通用的 Web 容器，只能自己实现 Web 服务，那么必须自己实现会话管理，并满足以下要求：

- 采用会话 cookie 维持会话。
- 生成会话标识（session ID）要保证足够的随机、离散，以便不能被猜测、枚举，要求 session ID 要至少要 32 字节，要支持字母和数字字符集。
- 服务端必须对客户端提交的 session ID 的有效性进行校验。

说明：在嵌入式系统中部署 Web 应用，由于软硬件资源所限，往往无法使用通用的 Web 容器及容器的会话管理功能，只能自己实现。另外，为了节省内存，嵌入式 webserver 进程往往是动态启动，为了使 session 更快的超时，建议增加心跳机制，对客户端浏览器是否关闭进行探测，5s 一个心跳，30s 没有心跳则 session 超时，关闭该 session。

2.4.4.权限管理

规则 4.4.1：对于每一个需要授权访问的页面或 servlet 的请求都必须核实用户的会话标识是否合法、用户是否被授权执行这个操作。

说明：防止用户通过直接输入 URL，越权请求并执行一些页面或 servlet；建议通过过滤器实现。

规则 4.4.2：授权和用户角色数据必须存放在服务器端，不能存放在客户端，鉴权处理也必须在服务器端完成。

说明：禁止将授权和角色数据存放在客户端中（比如 cookie 或隐藏域中），以防止被篡改。

规则 4.4.3：一个帐号只能拥有必需的角色和必需的权限。一个组只能拥有必需的角色和必需的权限。一个角色只能拥有必需的权限。

说明：做到权限最小化和职责分离（职责分离就是分清帐号角色，系统管理帐号只用于系统管理，审计帐号只用于审计，操作员帐号只用于业务维护操作，普通用户帐号只能使用业务。）这样即使帐号被攻击者盗取，也能把安全损失控制在最小的限度。

规则 4.4.4：对于运行应用程序的操作系统帐号，不应使用“root”、“administrator”、“supervisor”等特权帐号或高级别权限帐号，应该尽可能地使用低级别权限的操作系统帐号。

规则 4.4.5：对于应用程序连接数据库服务器的数据库帐号，在满足业务需求的前提下，必须使用最低级别权限的数据库帐号。

说明：根据业务系统要求，创建相应的数据库帐号，并授予必需的数据库权限。不能使用“sa”、“sysman”等管理帐号或高级别权限帐号。

2.4.5.敏感数据

规则 4.5.1：敏感数据定义主要包括：

个人数据：真实姓名、身份证号码（护照号码、港澳通行证号码等）、车牌号码、银行卡号码、现住地址、家庭地址、地理位置（GPS、IP 等）、生物信息（指纹、DNA、视网膜等）、医疗信息（体检结果、病历等）、财务信息（工资、交易信息、对账单等）等
用户标识：用户真实姓名、身份证号码（护照号码、港澳通行证号码等）、用户登录名、用户 ID、用户昵称、邮箱地址、手机号码、车牌号码、设备 ID、银行卡号码、社保号码等
用户验证：明文密码（内部默认密码、临时密码等）、令牌、会话 ID、硬件 Key、生物信息（指纹、视网膜、人脸等）
业务数据：财务数据、交易数据、知识产权数据、机密文件、系统源代码、密钥等
商户机构：各类密码等鉴权信息、磁道信息等、机构或商户的各类未公开数据，如终端设备信息（IMEI、无线 MAC 地址、IP 地址、DNS 地址等）
服务数据：特定主体的操作行为数据：操作记录、浏览记录等、特定主体的交易明细数据：转接清算交易信息、购物清单明细等、特定主体的服务记录数据：通话记录、网络对话数据、具体咨询事项、各类账单数据等

规则 4.5.2：关于敏感数据存储，请遵循以下安全要求：

规则 4.5.2.1：禁止在代码中存储敏感数据。

说明：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。

规则 4.5.2.2：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。

说明：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。

规则 4.5.2.3：禁止在 cookie 中以明文形式存储敏感数据。

说明：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。

规则 4.5.2.4：禁止在隐藏域中存放明文形式的敏感数据。

规则 4.5.2.5：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。

实施指导:

场景 1：后台服务端保存数据库的登录口令

后台服务器登录数据库需要使用登录数据库的明文口令，此时后台服务器加密保存该口令后，下次登录时需要还原成明文，因此，在这种情况下，不可用不可逆的加密算法，而需要使用对称加密算法或者非对称加密算法，一般也不建议采用非对称加密算法。

推荐的对称加密算法：AES128、AES192、AES256。

场景 2：后台服务端保存用户的登录口令

在该场景下，一般情况是：客户端提交用户名及用户口令，后台服务端对用户名及用户口令进行验证，然后返回验证的结果。此时，在后台服务端，用户口令可以不需要还原，因此建议使用不可逆的加密算法，对“用户名+口令”字符串进行加密。

推荐的不可逆加密算法： SHA256、SHA384、SHA512，HMAC-SHA256、HMAC-SHA384、HMAC-SHA512。

规则 4.5.2.6：禁止在日志中记录明文的敏感数据。

说明: 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。

规则 4.5.2.7：禁止带有敏感数据的 Web 页面缓存。

说明：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。

实施指导:

在 HTML 页面的<HEAD>标签内加入如下代码:

```
<HEAD>
<META HTTP-EQUIV="Expires" CONTENT="0">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-control" CONTENT="no-cache">
<META HTTP-EQUIV="Cache" CONTENT="no-cache">
</HEAD>
```

在 JSP 页面的最前面加入如下代码:

```
<%
response.setHeader("Cache-Control","no-cache");
response.setHeader("Pragma","no-cache");
response.setDateHeader("Expires",0);
%>
```

注意：以上代码对于采用强制缓存策略的代理服务器不生效（代理服务器默认是不缓存的），要防止代理服务器缓存页面，可以在链接后加入一个随机数 pageid,此时链接变成： http://localhost:8080/query.do?a=2&pageid=2245562, 其中 2245562 数字是随机生成的，每次请求此页面时，随机数都不同，IE 始终认为此为一个新请求，并重新解析，生成新的响应页面。

规则 4.5.3：关于敏感数据传输，请遵循以下安全要求:

规则 4.5.3.1：带有敏感数据的表单必须使用 HTTP-POST 方法提交。

说明：禁止使用 HTTP-GET 方法提交带有敏感数据的表单（form），因为该方法使用查询字符串传递表单数据，易被查看、篡改。如果是使用 servlet 处理提交的表单数据，那么不在 doGet 方法中处理，只在 doPost 方法处理。

实施指导:

1. 对于 JSP 页面，将表单的属性 method 赋值为"post"，如下

```
<form name="form1" method="post" action="switch.jsp">
```

2. 如果是使用 servlet 处理提交的表单数据，那么只在 doPost 方法中处理，参考代码如下

```
public class ValidationServlet extends HttpServlet
{
public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
{
```

```
        //对提交的表单数据进行校验
    }
    }
```

规则 4.5.3.2：在客户端和服务器间传递明文的敏感数据时，必须使用带服务器端证书的 SSL。

说明：如果在客户端和服务器间传递如帐号、口令等明文的敏感数据，必须使用带服务器端证书的 SSL。由于 SSL 对服务端的 CPU 资源消耗很大，实施时必须考虑服务器的承受能力。

实施指导：

1. SSL 的配置请参考《附件 1 Tomcat 配置 SSL 指导》。

2. Web 应用中，从 https 切换到 http 过程中会丢失 session，无法保持会话的连续。解决的办法就是用 http-https-http 过程代替 https-http 过程，保证会话的连续性。原因：当 https 请求转为 http 请求的时候，因为原先的 session 的 secure 属性值是 true，无法再 http 协议中传输，因此，系统生成新的 session，且新的 session 没有继承旧 session 的属性和值，因此，无法保持会话连续。而 http-https-http 这个过程，session 始终不变，因此，可以保持会话连续。

规则 4.5.3.3：禁止在 URL 中携带会话标识（如 jsessionid）。

说明：由于浏览器会保存 URL 历史记录，如果 URL 中携带会话标识，则在多人共用的 PC 上会话标识容易被其他人看到，一旦该会话标识还在其生命有效期，则恶意用户可以冒充受害用户访问 Web 应用系统。

规则 4.5.3.4：禁止将对用户保密的信息传送到客户端。

说明：这些信息一旦传送到客户端，那么用户也就可以获取到了。

2.4.6.安全审计

安全审计主要针对 Web 业务应用，不包括对操作系统、Web 容器的安全审计。对于操作系统和 Web 容器的安全审计，可以参考对应的操作系统安全基线和 Web 安全配置规范。

规则 4.6.1：应用服务器必须对安全事件及操作事件进行日志记录。

说明：安全事件包括登录、注销、添加、删除、修改用户、授权、取消权限、鉴权、修改用户口令等；操作事件包括对业务系统配置参数的修改，对重要业务数据的创建、删除、修改、查询等；对于上述事件的结果，不管是成功还是失败，都需要记录日志。

规则 4.6.2：安全日志必须包括但不限于如下内容：事件发生的时间、事件类型、客户端 IP、客户端机器名、当前用户的标识、受影响的个体（数据、资源）、成功或失败标识、启动该事件的进程标识以及对该事件的详细描述。

规则 4.6.3：严格限制对安全日志的访问。

说明：只有 Web 应用程序的管理员才能查询数据库表形式或文件形式的安全日志；除数据库超级管理员外，只有应用程序连接数据库的帐号可以查询（select）及插入（insert）安全日志表；除操作系统超级管理员外，只有应用程序的运行帐户才能读、写文件形式的安全日志（但不允许删除）。确保日志的安全，限制对日志的访问，这加大了攻击者篡改日志文件以掩饰其攻击行为的难度。

规则 4.6.4：对日志模块占用资源必须有相应的限制机制。

说明：限制日志模块占用的资源，以防止如自动的恶意登陆尝试导致的资源枯竭类 DOS 攻击；比如限制日志记录占用的磁盘空间。

规则 4.6.5：禁止日志文件和操作系统存储在同一个分区中，同时，应使用转储、滚动、轮循机制，来防止存储日志的分区写满。

说明：所需空间和具体业务、局点容量、日志保存周期相关，要根据实际情况估算。

建议 4.6.6：安全日志应该有备份及清理机制。

说明：备份及清理机制包括定期备份及清理安全日志和监控用于存放安全日志的磁盘空间的使用情况。可以配置定期备份及清理的时间，可以配置以用于存放安全日志的磁盘空间使用率达到多少时进行备份及清理。

建议 4.6.7：通过网络形式保存安全日志。

说明：在生成安全日志时，即时将日志保存到网络上其他主机，而且生成安全日志的应用程序不能再访问存放在其他主机的日志。

2.4.7.加密解密

规则 4.7.1：不使用自己开发的加密算法，而使用没有专利的、安全的、公开的标准加密算法。

说明：1、当前的专利算法包括但不限于：IDEA。已过专利保护期的加密算法已不再受专利保护；

2、已经被证明不再安全的公开加密算法包括：DES、MD5、SHA1、HMAC-MD5、HMAC-SHA1；

3、对称加密算法建议使用：AES；

4、密钥交换算法建议使用：DH；

5、数字签名算法建议使用：DSA、ECDSA；

6、非对称算法建议使用：ECC、RSA；

7、HASH（哈希）算法建议使用：SHA；

8、HMAC（基于哈希的消息验证码）算法建议使用：HMAC-SHA；

9、建议的各算法建议采用如下的加密强度：

加密算法	最小强度	建议的默认强度	要求更高强度时建议的默认强度
AES	128	128	192、256
DH	1024	1024	2048
DSA	1024	1024	1024
ECC	192	192	216
ECDSA	192	192	216
RSA	1024	1024	2048
SHA	224	256	384、512
HMAC	HMAC-SHA224	HMAC-SHA256	HMAC-SHA384 、HMAC-SHA512

10、加密口令如何选择加密算法：

场景 1：后台服务端保存数据库的登录口令

后台服务器登录数据库需要使用登录数据库的明文口令，此时后台服务器加密保存该口令后，下次登录时需要还原成明文，因此，在这种情况下，不可用不可逆的加密算法，而需要使用对称加密算法或者非对称加密算法，一般也不建议采用非对称加密算法。推荐的对称加密算法：AES128、AES192、AES256；

场景 2：后台服务端保存用户的登录口令

在该场景下，一般情况是：客户端提交用户名及用户口令，后台服务端对用户名及用户口令进行验证，然后返回验证的结果。此时，在后台服务端，用户口令可以不需要还原，因此建议使用不可逆的加密算法：HMAC 算法或者 HASH 算法，建议优先使用 HMAC 算法。推荐的不可逆加密算法： SHA256、SHA384、SHA512，HMAC-SHA256、HMAC-SHA384、HMAC-SHA512。

规则 4.7.2：口令、通行码、初始密钥不能直接作为加密算法的密钥，须先采用密钥导出算法（如 PKCS5）对它进行处理，之后得到的结果才能作为加密算法的密钥。

说明：1、导出的密钥比口令具有更好的密码学特性，安全性更高；

2、公司开发的加密库，其中包含了密钥导出函数：PKCS5-deriveKey(…)，可以直接调用该函数导出加密的密钥。Java 中请参考类 PBEKeySpec；

3、请参考“说明”中名词解释对“初始密钥”、“主密钥”、“工作密钥”的解释。

规则 4.7.2：初始密钥遵循如下两个规则之一，优先遵循第一条规则：

1、对于程序启动是可以有操作员干预的情况（如该程序一般都是有操作员手工启动），则初始密钥必须是在程序启动过程中或者运行过程中由操作员手工输入的，同时初始密钥遵循“口令策略”，初始密钥参与加密时遵循“口令作为密钥时使用前需采用密钥导出算法”。

2、对于程序是自动启动（或者是被守护进程带起）、操作员无法干预的情况，初始密钥写死到代码中，同时遵循“口令作为密钥时使用前需采用密钥导出算法”。写死到代码中初始密钥的最小长度为 512 字节，同时遵循以下原则：

不可见原则：由不可见字符组成；

不重复原则：任意两个连续的字符不能是相同的；

不连续原则：任意两个连续的字符不能是两个连续的值（如 0x99、0x98 是连续的两个值）；

字节赋值原则：采用赋值语句给该缓存变量的每个字节分别赋值。

例如：

char szInitKey[512] = {(char)0x99, (char)0x01, …, (char)0x02}; // 定义并初始化，不符合规则

符合规则的做法：

char szInitKey[512]; // 定义

szInitKey[0] = (char)0x99; // 给第 0 字节赋值

szInitKey[1] = (char)0x01; // 给第 1 字节赋值

…

szInitKey[511] = (char)0x02; // 给第 511 字节赋值

规则 4.7.3：主密钥不直接用来加密敏感数据，而是随机生成一个工作密钥用来加密敏感数据，而用主密钥加密工作密钥；解密时，先用主密钥解密得到工作密钥，再用工作密钥解密得到敏感数据。

说明：请参考“说明”中名词解释对“初始密钥”、“主密钥”、“工作密钥”的解释。

2.4.8.没有后门

规则 4.8.1：禁止在系统中留有预留任何的后门帐号或特殊的访问机制用以维护、支持或操作的需要。

说明：例如系统中留了一个特别的帐号和密码、没有公开的通信字（功能码）等、没有公开的参数等。

2.4.9.Web Service

规则 4.9.1：对 Web Service 接口的调用必须进行认证。

说明：认证就是确定谁在调用 Web Service，并且证实调用者身份。

实施指导：

可以通过在消息头中增加用户名和口令，作为认证凭据；

对于安全性要求不高、只向同一信任域内其他主机开放的 Web Service 接口，可以通过简单的 IP 认证来实现接口的认证（只有服务器端指定 IP 地址的客户端才允许调用，IP 地址可配置）。

规则 4.9.2：如果调用者的权限各不相同，那么必须对 Web Service 接口的调用进行鉴权。

说明：鉴权就是判断调用者是否有权限调用该 Web Service 接口。

实施指导：

可以通过 Axis 的 handler 对调用进行鉴权。

规则 4.9.3：通过 Web Service 接口传递敏感数据时，必须保障其机密性。

实施指导：

方案 1：请参考《附件 2 Web Service 安全接入开发指导》。

方案 2：采用 https 安全协议。

规则 4.9.4：通过 Web Service 接口传递重要的交易数据时，必须保障其完整性和不可抵赖性。

说明：重要的交易数据，如转账时涉及的“转入账号”、“转出账号”、“金额”等。

实施指导：

请参考 Web Service 安全接入开发指导。

规则 4.9.5：如果 Web Service 只对特定的 IP 开放，那么必须对调用 Web Service 接口的客户端 IP 进行鉴权，只有在 IP 地址白名单中的客户端才允许调用，IP 地址白名单可配置。

实施指导：请参考客户端 IP 鉴权实施指导。

规则 4.9.6：对 Web Service 接口调用进行日志记录。

说明：日志内容包括但不限于如下内容：调用时间、操作类型、调用接口名称、详细的接口参数、客户端 IP、客户端机器名、调用者的用户标识、受影响的个体（数据、资源）、成功或失败标识。

规则 4.9.7：必须对 Web Service 提交的参数进行输入校验。

说明：具体输入校验部分请查看 5.1 输入校验。

2.4.10. 输入验证

规则 4.10.1：必须对所有用户产生的输入进行校验，一旦数据不合法，应该告知用户输入非法并且建议用户纠正输入。

说明：用户产生的输入是指来自 text、password、textareas 或 file 表单域的数据；必须假定所有用户产生的输入都是不可信的，并对它们进行合法性校验。

规则 4.10.2：必须对所有服务器产生的输入进行校验，一旦数据不合法，必须使会话失效，并记录告警日志。

说明：服务器产生的输入是指除用户产生的输入以外的输入，例如来自 hidden fields、selection boxes、check boxes、radio buttons、cookies、HTTP headers、热点链接包含的 URL 参数的数据或客户端脚本等；必须假定所有服务器产生的输入都是被篡改过的、恶意的，并对它们进行合法性校验，如果不合法，说明有人恶意篡改数据。举例：假如用户资料填写表单中的“性别”为必填项，用 radio button（‘男’和‘女’对应实际值分别为‘1’和‘0’）来限制用户的输入，如果应用程序收到的“性别”值为‘2’，那么可以断定有人恶意篡改数据。

规则 4.10.3：禁止将 HTTP 标题头中的任何未加密信息作为安全决策依据。

说明：HTTP 标题头是在 HTTP 请求和 HTTP 响应的开始阶段发送的。Web 应用程序必须确保不以 HTTP 标题头中的任何未加密信息作为安全决策依据，因为攻击者要操作这一标题头是很容易的。例如，标题头中的 referer 字段包含来自请求源端的 Web 页面的 URL。不要根据 referer 字段的值做出任何安全决策（如检查请求是否来源于 Web 应用程序生成的页面），因为该字段是很容易被伪造的。

规则 4.10.4：不能依赖于客户端校验，必须使用服务端代码对输入数据进行最终校验。

说明：客户端的校验只能作为辅助手段，减少客户端和服务端的信息交互次数。

规则 4.10.5：对于在客户端已经做了输入校验，在服务器端再次以相同的规则进行校验时，一旦数据不合法，必须使会话失效，并记录告警日志。

说明：肯定存在攻击行为，攻击者绕过了客户端的输入校验，因此必须使会话失效，并记入告警日志。

规则 4.10.6：如果输入为数字参数则必须进行数字型判断。

说明：这里的数字参数指的是完全由数字组成的数据。

实施指导：

```
String mobileno = request.getParameter("mobileno");
String characterPattern = "^\\d+$";    //正则表达式表示是否全为数字
if (!mobileno.matches (characterPattern))
{
out.println (“Invalid Input”);
}
```

规则 4.10.7：如果输入只允许包含某些特定的字符或字符的组合，则使用白名单进行输入校验。

说明：对于一些有规则可循的输入，如 email 地址、日期、小数等，使用正则表达式进行白名单校验，这样比使用黑名单进行校验更有效。

实施指导：

```
email 地址校验的方法：
String emailAddress = request.getParameter("emailAddress");
String characterPattern =
"^([a-z0-9A-Z]+[_-]?)+[a-z0-9A-Z]@(([a-z0-9A-Z]+[_-]?)+(-[a-z0-9A-Z]+)?\\.)
+[a-zA-Z]{2,4}$"; //email 正则表达式
if (!emailAddress.matches(characterPattern))
{
out.println (“Invalid Email Address”);
}
```

规则 4.10.8：如果输入为字符串参数则必须进行字符型合法性判断。

说明：可定义一个合法字符集。

实施指导：

```
String text = request.getParameter("text");
String characterPattern = "^[A-Za-z]*$"; //开发者自行定义字符规则(方括号内的字符集)
if (!text.matches (characterPattern))
{
out.println ("Invalid Input");
}
```

规则 4.10.9：校验输入数据的长度。

说明：如果输入数据是字符串，必须校验字符串的长度是否符合要求，长度校验会加大攻击者实施攻击的难度。

规则 4.10.10：校验输入数据的范围。

说明：如果输入数据是数值，必须校验数值的范围是否正确，如年龄应该为 0 ~ 150 之间的正整数。

规则 4.10.11：禁止通过字符串串联直接使用用户输入构造可执行 SQL 语句。

说明：禁止通过字符串串联直接使用用户输入构造可执行 SQL 语句，如：string sql = "select status from Users where UserName='" + txtUserName.Text + "'";这样很容易被 SQL 注入攻击。

规则 4.10.12：对于 java/JSP 语言，使用预编译语句 PreparedStatement 代替直接的语句执行 Statement。

说明：使用预编译语句 PreparedStatement，类型化 SQL 参数将检查输入的类型，确保输入值在数据库中当作字符串、数字、日期或 boolean 等值而不是可执行代码进行处理，从而防止 SQL 注入攻击。而且，由于 PreparedStatement 对象已预编译过，所以其执行速度要快于 Statement 对象。因此，多次执行的 SQL 语句经常创建为 PreparedStatement 对象，还可以提高效率。

实施指导：

参考如下代码：

```
String inssql = "insert into buy(empid, name, age, birthday) values (?,?,?,?)";
PreparedStatement stmt = null;
stmt = conn.prepareStatement(inssql);
stmt.setString(1, empid);
stmt.setString(2, name);
stmt.setInt(3, age);
stmt.setDate(4, birthday);
stmt.execute();
```

备注：使用 like 进行模糊查询时，如果直接用"select * from table where comment like %?%"，程序会报错，必须采用如下方法

```
String express = "select * from table where comment like ?";
pstmt = con.prepareStatement(express);
String c="hello";
pstmt.setString(1, "%"+c+"%");
//参数自动添加单引号，最后的 SQL 语句为：select * from  table where comment like '%hello%'
pstmt.execute();
```

规则 4.10.13：禁止动态构建 XPath 语句。

说明：和动态构建 SQL 一样，动态构建 XPath 语句也会导致注入漏洞（XPath 注入）。动态构建 XPath 语句的例子：public boolean doLogin(String loginID, String password){.....

```
XPathExpression expr = xpath.compile("//users/user[loginID/text()='"+loginID+"' and password/text()='"+password+"' ]/firstname/text()");
.....}
```

规则 4.10.14：在 JavaBean 中禁止使用 property="*"进行参数赋值。

说明：property="*"这表明用户在可见的 JSP 页面中输入的，或是直接通过 Query String 提交的参数值，将存储到与参数名相匹配的 bean 属性中。例如，网上购物程序，一般，用户是这样提交请求的：http://www.somesite.com /addToBasket.jsp?newItem=ITEM0105342 ，如果用户提交：http://www.somesite.com /addToBasket.jsp?newItem=ITEM0105342&balance=0，这样，balance=0 的信息就被在存储到了 JavaBean 中了，而 balance 是整个会话中用来存储总费用的，当他们这时点击“chekout”结账的时候，费用就全免了。

规则 4.10.15：用于重定向的输入参数不能包含回车和换行字符，以防止 HTTP 响应拆分攻击。

说明：注意，“回车”字符有多种表示方式（CR = %0d = \r ），“换行”字符有多种表示方式（LF = %0a = \n）。

规则 4.10.16：如果服务端代码执行操作系统命令，禁止从客户端获取命令。

说明：如果服务端代码中使用 Runtime.getRuntime().exec(cmd) 或 ProcessBuilder 等执行操作系统命令，那么禁止从客户端获取命令；而且最好不要从客户端获取命令的参数，如果必须从客户获取命令的参数，那么必须采用正则表达式对命令参数进行严格的校验，以防止命令注入（因为，一旦从客户端获取命令或参数，通过;&|<>符号，非常容易构造命令注入，危害系统）。

2.4.11. 输出编码

规则 4.11.1：对于不可信的数据，输出到客户端前必须先进行 HTML 编码。

说明：不可信的数据（也就是其他业务系统生成的未经本应用程序验证的表数据或文件数据），通过对输出到客户端的数据进行编码，可以防止浏览器将 HTML 视为可执行脚本，从而防止跨站脚本攻击。

实施指导：

JSP 语言可以通过替换输出数据的特殊字符【& < > ” ’ ( )%+-】为其他表示

形式后再输出给客户端，例如：

```
<%
String OutStr = "<script>alert('XSS')</script>";
OutStr = OutStr.replaceAll("&","&amp;");
OutStr = OutStr.replaceAll("<","&lt;");
OutStr = OutStr.replaceAll(">","&gt;");
OutStr = OutStr.replaceAll("\"","&quot;");
OutStr = OutStr.replaceAll("\'","&#39;");
OutStr = OutStr.replaceAll("\\(","&#40;");
OutStr = OutStr.replaceAll("\\)","&#41;");
out.println(OutStr);
%>
```

ASP.NET 语言可以通过 HtmlEncode 方法对 HTML 的输出进行编码。

PHP 语言可以通过 htmlentities 或 htmlspecialchars 方法对 HTML 输出进行编码。

2.4.12. 上传下载

规则 4.12.1：必须在服务器端采用白名单方式对上传或下载的文件类型、大小进行严格的限制。

规则 4.12.2：禁止以用户提交的数据作为读/写/上传/下载文件的路径或文件名，以防止目录跨越和不安全直接对象引用攻击。

说明：建议对写/上传文件的路径或文件名采用随机方式生成，或将写/上传文件放置在有适当访问许可的专门目录。对读/下载文件采用映射表（例如，用户提交的读文件参数为 1，则读取 file1，参数为 2，则读取 file2）。防止恶意用户构造路径和文件名，实施目录跨越和不安全直接对象引用攻击。

规则 4.12.3：禁止将敏感文件（如日志文件、配置文件、数据库文件等）存

放在 Web 内容目录下。

说明：Web 内容目录指的是：通过 Web 可以直接浏览、访问的目录，存放在 Web 内容目录下的文件容易被攻击者直接下载。

2.4.13. 异常处理

规则 4.13.1：应用程序出现异常时，禁止向客户端暴露不必要的信息，只能向客户端返回一般性的错误提示消息。

说明：应用程序出现异常时，禁止将数据库版本、数据库结构、操作系统版本、堆栈跟踪、文件名和路径信息、SQL 查询字符串等对攻击者有用的信息返回给客户端。建议重定向到一个统一、默认的错误提示页面，进行信息过滤。

规则 4.13.2：应用程序捕获异常，并在日志中记录详细的错误信息。

说明：记录详细的错误消息，可供入侵检测及问题定位。

2.4.14. 代码注释

规则 4.14.1：在注释信息中禁止包含物理路径信息。

规则 4.14.2：在注释信息中禁止包含数据库连接信息。

规则 4.14.3：在注释信息中禁止包含 SQL 语句信息。

规则 4.14.4：对于静态页面，在注释信息中禁止包含源代码信息。

规则 4.14.5：对于动态页面不使用普通注释，只使用隐藏注释。

说明：动态页面包括 ASP、PHP、JSP、CGI 等由动态语言生成的页面。通过浏览器查看源码的功能，能够查看动态页面中的普通注释信息，但看不到隐藏注释（隐藏注释不会发送给客户端）。因此，为了减少信息泄漏，建议只使用隐藏注释。

实施指导：

```
<form action=h.jsp>
<%--隐藏注释 1--%>
```

```
<textarea name=a length=200></textarea>
<input type=submit value=test>
</form>
<%
//隐藏注释 2
java.lang.String str=(String)request.getParameter("a");
/*隐藏注释 3*/
str = str.replaceAll("<","&lt;");
out.println(str);
%>
```

2.4.15. 归档要求

规则 4.15.1：版本归档时，必须删除开发过程（包括现场定制）中的临时文件、备份文件、无用目录等。

说明：恶意用户可以通过 URL 请求诸如.bak 之类的文件，Web 服务器会将这些文件以文本方式呈现给恶意用户，造成代码的泄漏，严重威胁 Web 应用的安全。

实施指导：

在 web 应用的根目录下执行以下命令：

find ./ -name "*.old" -o -name "*.OLD" -o -name "*.bak" -o -name "*.BAK" -o -name "*.temp" -o -name "*.tmp" -o -name "*.save" -o -name "*.backup" -o -name "*.orig" -o -name "*.000" -o -name "*~" -o -name "*~1" -o -name "*.dwt" -o -name "*.tpl" -o -name "*.zip" -o -name "*.7z" -o -name "*.rar" -o -name "*.gz" -o -name "*.tgz" -o -name "*.tar" -o -name "*.bz2"

分析查找到的文件是否临时文件、备份文件、无用文件，如果是则删除。

规则 4.15.2：归档的页面程序文件的扩展名必须使用小写字母。

说明：很多 Web server 对大小写是敏感的，但对后缀的大小写映像并没有做正确的处理。攻击者只要在 URL 中将 JSP 文件后缀从小写变成大写，Web 服务器就不能正确处理这个文件后缀，而将其当作纯文本显示。攻击者可以通过查看源码获得这些程序的源代码。因此，归档的页面程序文件的扩展名必须使用小写字母，如 jsp、html、htm、asp 等页面程序文件的扩展名分别为 jsp、html、htm、asp。

规则 4.15.3：归档的程序文件中禁止保留调试用的代码。

说明：这里的“调试用的代码”是指开发过程中进行临时调试所用的、在 Web 应用运行过程中不需要使用到的 Web 页面代码或 servlet 代码。例如：在代码开发过程中为了测试一个添加帐号的功能，开发人员临时编写了一个 JSP 页面进行测试，那么在归档时，该 JSP 页面必须删除，以免被攻击者利用。

2.4.16. 其他要求

规则 4.16.1：对于 JSP 语言，所有 servlet 必须进行静态映射，不允许通过绝对路径访问。

说明：在 web.xml 文件中为 servlet 配置 URI 映射，使用 servlet 时，引用它的 URI 映射，而不允许通过绝对路径访问。

规则 4.16.2：对客户端提交的表单请求进行合法性校验，防止跨站请求伪造攻击。

说明：跨站请求伪造（CSRF）是一种挟制终端用户在当前已登录的 Web 应用程序上执行非本意的操作的攻击方法。攻击者可以迫使用户去执行攻击者预先设置的操作，例如，如果用户登录网络银行去查看其存款余额，他没有退出网络银行系统就去了自己喜欢的论坛去灌水，如果攻击者在论坛中精心构造了一个恶意的链接并诱使该用户点击了该链接，那么该用户在网络银行帐户中的资金就有可能被转移到攻击者指定的帐户中。当 CSRF 针对普通用户发动攻击时，将对终端用户的数据和操作指令构成严重的威胁；当受攻击的终端用户具有管理员帐户的时候，CSRF 攻击将危及整个 Web 应用程序。

实施指导:

方法一：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
}
```

方法二：受理重要操作请求时，在相应的表单页面增加图片验证码，用户提交操作请求的同时提交验证码，在服务器端先判断用户提交的验证码是否正确，验证码正确再受理操作请求。

规则 4.16.3：使用.innerHtml 时，如果只是要显示文本内容，必须在 innerHTML 取得内容后，再用正则表达式去除 HTML 标签，以预防跨站脚本。

说明：使用.innerHtml 会将内容以 HTML 显示，容易被利用，导致跨站脚本。

实施指导:

```
<a
href="javascript:alert(document.getElementById('test').innerHTML.replace(/<.+?
>/gim,''))">无 HTML,符合 W3C 标准</a>
```

备注：还可以使用.innerText 代替.innerHtml，.innerText 只显示文本内容不

显示 HTML 标签, 但.innerText 不是 W3C 标准的属性, 不能适用于所有浏览器(但适用于 IE 浏览器）。

规则 4.16.4：禁止使用 eval()函数来处理用户提交的字符串。

说明：eval()函数存在安全隐患，该函数可以把输入的字符串当作 JavaScript 表达式执行，容易被恶意用户利用。

建议 4.16.5：关闭登录窗体表单中的自动填充功能，以防止浏览器记录用户名和口令。

说明：浏览器都具有自动保存用户输入数据和自动填充数据的能力。为了保障用户名和口令的安全，必须关闭自动填充选项，指示浏览器不要存储登录窗口中用户名、口令等敏感信息。

实施指导：

在 form 表单头中增加选项（autocomplete="off"），例如：

```
<form action="Login.jsp" name=login method=post autocomplete="off">
```

建议 4.16.6：防止网页被框架盗链或者点击劫持。

说明：框架盗链和点击劫持（ClickJacking）都利用到框架技术，防范措施就是防止网页被框架。

实施指导：

方法一：在每个网页上增加如下脚本来禁止 iframe 嵌套：

```
< <script>
    if(top != self) top.location.href = location.href;
</script>
```

2.5. API 安全

针对 API 的安全设计参考：

2.5.1.身份认证

规则 5.1.1：不要使用 Basic Auth ，使用标准的认证协议 (如 JWT, OAuth)。

规则5.1.2：不要再造 Authentication, token generating, password storing 这些轮子，使用标准的。

规则 5.1.3：在登录中使用 Max Retry 和自动封禁功能。

规则 5.1.4：加密所有的敏感数据。

2.5.2.JWT (JSON Web Token)

规则 5.2.1：使用随机复杂的密钥 (JWT Secret) 以增加暴力破解的难度。

规则 5.2.2：不要在请求体中直接提取数据，要对数据进行加密 (HS256 或 RS256)。

规则 5.2.3：使 token 的过期时间尽量的短 (TTL, RTTL)。

规则 5.2.4：不要在 JWT 的请求体中存放敏感数据，它是可破解的。

2.5.3.OAuth 授权或认证协议

规则 5.3.1：始终在后台验证 redirect_uri，只允许白名单的 URL。

规则 5.3.2：每次交换令牌的时候不要加 token (不允许 response_type=token)。

规则 5.3.3：使用 state 参数并填充随机的哈希数来防止跨站请求伪造 (CSRF)。

规则 5.3.4：对不同的应用分别定义默认的作用域和各自有效的作用域参数。

2.5.4.访问保护

规则 5.4.1：限制流量来防止 DDoS 攻击和暴力攻击。

规则 5.4.2：在服务端使用 HTTPS 协议来防止 MITM 攻击。

规则 5.4.3：使用 HSTS 协议防止 SSLStrip 攻击。

2.5.5.输入验证

规则 5.5.1：使用与操作相符的 HTTP 操作函数，GET (读取)，POST (创建)，PUT (替换/更新) 以及 DELETE (删除记录)，如果请求的方法不适用于请求的资源则返回 405 Method Not Allowed。

规则 5.5.2：在请求头中的 content-type 字段使用内容验证来只允许支持的格式 (如 application/xml，application/json 等等) 并在不满足条件的时候返回 406 Not Acceptable。

规则 5.5.3：验证 content-type 的发布数据和你收到的一样 (如 application/x-www-form-urlencoded，multipart/form-data，application/json 等等)。

规则 5.5.4：验证用户输入来避免一些普通的易受攻击缺陷 (如 XSS，SQL-注入，远程代码执行 等等)。

规则 5.5.5：不要在 URL 中使用任何敏感的数据 (credentials，Passwords，security tokens，or API keys)，而是使用标准的认证请求头。

规则 5.5.6：使用一个 API Gateway 服务来启用缓存、访问速率限制 (如 Quota，Spike Arrest，Concurrent Rate Limit) 以及动态地部署 APIs resources。

2.5.6.资源处理

规则 5.6.1：检查是否所有的终端都在身份认证之后，以避免被破坏了的认证体系。

规则 5.6.2：避免使用特有的资源 id. 使用 /me/orders 替代 /user/654321/orders。

规则 5.6.3：使用 UUID 代替自增长的 id。

规则 5.6.4：如果需要解析 XML 文件，确保实体解析(entity parsing)是关闭的以避免 XXE 攻击。

规则 5.6.5：如果需要解析 XML 文件，确保实体扩展(entity expansion)是关

闭的以避免通过指数实体扩展攻击实现的 Billion Laughs/XML bomb。

规则 5.6.6：在文件上传中使用 CDN。

规则 5.6.7：如果需要处理大量的数据，使用 Workers 和 Queues 来快速响应，从而避免 HTTP 阻塞。

规则 5.6.8：不要忘了把 DEBUG 模式关掉。

2.5.7.输出处理

规则 5.7.1：发送 X-Content-Type-Options: nosniff 头。

规则 5.7.2：发送 X-Frame-Options: deny 头，X-XSS-Protection: 1; mode=block 头。

规则 5.7.3：发送 Content-Security-Policy: default-src 'none' 头。

规则 5.7.4：删除指纹头 - X-Powered-By, Server, X-AspNet-Version 等等。

规则 5.7.5：在响应中强制使用 content-type，如果你的类型是 application/json 那么你的 content-type 就是 application/json。

规则 5.7.6：不要返回敏感的数据，如 credentials, Passwords, security tokens。

规则 5.7.7：在操作结束时返回恰当的状态码. (如 200 OK, 400 Bad Request, 401 Unauthorized, 405 Method Not Allowed 等等)。

2.6. Spring Boot 安全

针对 Spring Boot 的安全设计参考：

https://docs.spring.io/spring-security/site/docs/current/reference/html/

2.6.1.使用 HTTPS

规则 6.1.1：通过扩展 WebSecurityConfigurerAdapter 要求安全连接，强制使用 HTTPS；

@Configuration

```
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(HttpSecurity http) throws Exception {
     http.requiresChannel().requiresSecure();
  }
}
```

2.6.2.依赖检查

规则 6.2.1：对依赖库进行漏洞检测，及时更新有漏洞的组件，可以使用 OWASP Dependency Check 或 Snyk（需要登录）工具等进行扫描检查。

https://www.owasp.org/index.php/OWASP_Dependency_Check

https://github.com/snyk/snyk/releases

2.6.3.CSRF 保护

规则 6.3.1：Spring Security 自带 CSRF 支持，默认情况下处于启用状态，使用 Spring MVC 的<form:form>标记或 Thymeleaf 和@EnableWebSecurity ，则 CSRF 标记将自动添加为隐藏输入字段。

如果使用的是像 Angular 或 React 这样的 JavaScript 框架，则需要配置 CookieCsrfTokenRepository 以便 JavaScript 可以读取 cookie：

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf()
```

```
                .csrfTokenRepository(CookieCsrfTokenRepository.withHttp
OnlyFalse());
        }
    }
```

2.6.4.防止 XSS

规则 6.4.1：可以配置应用程序以返回 Content-Security-Policy 头，来开启内容安全策略 （CSP）来缓解 XSS（跨站点脚本）和数据注入攻击，Spring Security 默认不添加 CSP，可以使用以下配置在 Spring Boot 应用程序中启用 CSP 标头：

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.headers()
            .contentSecurityPolicy("script-src 'self'
https://trustedscripts.example.com; object-src
https://trustedplugins.example.com; report-uri /csp-report-endpoint/");
    }
}
```

2.6.5.OpenID 验证

规则 6.5.1：使用 OpenID Connect 进行身份验证，OpenID Connect（OIDC）是一个 OAuth 2.0 扩展，提供用户信息， 除了访问令牌之外，它还添加了 ID 令牌，以及可以从中获取其他信息的/userinfo 端点。 它还添加了端点发现功能和动态客户端注册。

2.6.6.加密算法

规则 6.6.1： Spring Security 默认提供了三种密码存储方式：BCryptPasswordEncoder 和 Pbkdf2PasswordEncoder 等，MD5+Salt 的自定义的加密方式可以在 configure(HttpSecurity http) 后面添加 configure(AuthenticationManagerBuilder auth)来实现。

2.6.7.最新版本

规则 6.7.1：Spring Security 及其依赖库也是会有漏洞的，因此必须更新到无漏洞的版本，和支持版本兼容更新机制。

2.6.8.数据存储

规则 6.8.1：对一些敏感数据，需要加密存储，不允许明文存储。

2.6.9.安全测试

规则 6.9.1：需要对应用系统进行安全测试，针对 Web 安全漏洞和业务逻辑漏洞等。

2.6.10. 源码审计

规则 6.10.1：需要对源码进行安全审计，针对 Web 安全漏洞和业务逻辑漏洞、安全配置项、后门情况等进行分析。

2.7.组件安全

针对主要的组件 Spring、Hibernate、MyBatis、Struts2、jQuery 等，安全设计请参考:

2.7.1.Spring Security

规则 7.1.1：注册功能的执行过程中，必须对用户的输入进行安全验证。

规则 7.1.2：验证机制执行过程中，必须对用户越权访问进行安全验证。

规则 7.1.3：对会话管理、用户信息处理等核心安全必须进行记录。

规则 7.1.4：通过 OAuth2 实现使用 JSON 令牌时，必须对用户越权访问进行安全验证。

规则 7.1.5：使用 REST API 时，必须进行身份验证。

2.7.2.Hibernate

规则 7.2.1：不要直接使用 SQL 语句拼接，会导致 SQL 注入漏洞发生：

List results = session.createQuery("from Orders as orders where orders.id = " + currentOrder.getId()).list();

List results = session.createSQLQuery("Select * from Books where author = " + book.getAuthor()).list();

规则 7.2.1：使用参数化查询方式，安全的示例如下：

/* Positional parameter in HQL */

Query hqlQuery = session.createQuery("from Orders as orders where orders.id = ?");

List results = hqlQuery.setString(0, "123-ADB-567-QTWYTFDL").list();

/* named parameter in HQL */

Query hqlQuery = session.createQuery("from Employees as emp where emp.incentive > :incentive");

List results = hqlQuery.setLong("incentive", new Long(10000)).list();

/* named parameter list in HQL */

List items = new ArrayList();

items.add("book"); items.add("clock"); items.add("ink");

List results = session.createQuery("from Cart as cart where cart.item in (:itemList)").setParameterList("itemList", items).list();

/* JavaBean in HQL */

Query hqlQuery = session.createQuery("from Books as books where book.name = :name and book.author = :author");

List results = hqlQuery.setProperties(javaBean).list(); //assumes javaBean has getName() & getAuthor() methods.

/* Native-SQL */

Query sqlQuery = session.createSQLQuery("Select * from Books where author = ?");

List results = sqlQuery.setString(0, "Charles Dickens").list();

2.7.3.MyBatis

规则 7.3.1：不要直接使用 ${} 语法拼接，会导致 SQL 注入漏洞发生：

<select id="getPerson" parameterType="string" resultType="org.application.vo.Person">

SELECT * FROM PERSON WHERE NAME = #{name} AND PHONE LIKE '${phone}';

</select>

规则 7.3.2：使用 #{} 语法生成 PreparedStatement 参数方式，安全的示例如下：

<select id="getPerson" parameterType="int" resultType="org.application.vo.Person">

SELECT * FROM PERSON WHERE ID = #{id}

</select>

类似：

/* Comparable JDBC code */

String selectPerson = "SELECT * FROM PERSON WHERE ID = ?";

PreparedStatement ps = conn.prepareStatement(selectPerson);

ps.setInt(1, id);

2.7.4.JPA

规则 7.4.1：不要直接使用 SQL 语句拼接，会导致 SQL 注入漏洞发生：

List results = entityManager.createQuery("Select order from Orders order where order.id = " + orderId).getResultList();

List results = entityManager.createNativeQuery("Select * from Books where author = " + author).getResultList();

int resultCode = entityManager.createNativeQuery("Delete from Cart where itemId = " + itemId).executeUpdate();

规则 7.4.2：使用参数绑定方式，安全的示例如下：

/* positional parameter in JPQL */

Query jpqlQuery = entityManager.createQuery("Select order from Orders order where order.id = ?1");

List results = jpqlQuery.setParameter(1, "123-ADB-567-QTWYTFDL").getResultList();

/* named parameter in JPQL */

Query jpqlQuery = entityManager.createQuery("Select emp from Employees emp where emp.incentive > :incentive");

List results = jpqlQuery.setParameter("incentive", new Long(10000)).getResultList();

/* named query in JPQL - Query named "myCart" being "Select c from Cart c where c.itemId = :itemId" */

Query jpqlQuery = entityManager.createNamedQuery("myCart");

List results = jpqlQuery.setParameter("itemId", "item-id-0001").getResultList();

/* Native SQL */

Query sqlQuery = entityManager.createNativeQuery("Select * from Books where author = ?", Book.class);

List results = sqlQuery.setParameter(1, "Charles Dickens").getResultList();

2.7.5.SQL

规则 7.5.1：不要直接使用 SQL 语句拼接，会导致 SQL 注入漏洞发生：

// 示例 #1

String query = "SELECT * FROM users WHERE userid ='"+ userid + "'" + " AND password='" + password + "'";

Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(query);

// 示例 #2

String query = "SELECT * FROM users WHERE userid ='"+ userid + "'" + " AND password='" + password + "'";

PreparedStatement stmt = connection.prepareStatement(query);

ResultSet rs = stmt.executeQuery();

规则 7.5.2：使用参数化查询方式，安全的示例如下：

PreparedStatement stmt = connection.prepareStatement("SELECT * FROM users WHERE userid=? AND password=?");

stmt.setString(1, userid);

stmt.setString(2, password);

ResultSet rs = stmt.executeQuery();

2.7.6.Struts2

规则 7.6.1：限制对 Config Browser 插件的访问，设置验证。

规则 7.6.2：不要在同一命名空间中混合使用不同的访问级别。

规则 7.6.3：不要直接暴露 JSP 文件，可以通过将所有 JSP 文件放在 WEB-INF 文件夹下，或向 web.xml 文件添加安全性约束。

规则 7.6.4：禁止 devMode，可以通过 struts.xml 配置：

<constant name ="struts.devMode" value="false" />

规则 7.6.5：降低日志记录级别，不要设置为 DEBUG 级别，设置 WARN 或至少 INFO 级别。

规则 7.6.6：使用 UTF-8 编码，使用 JSP 时请将以下标头添加到每个 JSP 文件中：

<%@ page contentType="text/html; charset=UTF-8" %>

规则 7.6.7：不要定义不需要的 setter，可能会导致表达式注入问题。

规则 7.6.8：不要将传入值用作本地化逻辑的输入，所有 TextProvider 的 getText（...）方法（例如 inActionSupport）都会对消息中包含的参数进行评估，以正确本地化文本。 这意味着使用带有 getText（...）方法的传入请求参数具有潜在的危险性，应该避免使用。以下代码会导致 OGNL 表达式注入：

```
public String execute() throws Exception {
    setMessage(getText(getMessage()));
    return SUCCESS;
}
```

规则 7.6.9：使用 Struts 标记而不是原始 EL 表达式，切勿对传入值使用原始 $ {} EL 表达式，这会导致代码注入问题。

规则 7.6.10：不要更改内置的安全管理器，它阻止对特定类和 Java 包的访问，它是一个 OGNL 范围的机制，这意味着它影响框架的任何方面。

规则 7.6.11：禁止从表达式访问静态方法，这会导致代码注入问题。

规则 7.6.12：OGNL 调用 action 方法时，不要在层次结构中使用相同方法的

名称，只需将操作的方法从 save（）更改为 saveAction（）并保留注释，以允许通过/save.action 请求调用此操作。

规则 7.6.13：接受/排除的模式，从 2.3.20 版本开始，框架提供了两个新接口，用于接受/排除参数名称和值 - 带默认实现的 AcceptedPatternsChecker 和 ExcludedPatternsChecker 。 可以使用 Parameters Interceptor 和 Cookie Interceptor 这两个接口来检查 param 是否可以被接受或必须被排除。

规则 7.6.14：严格的方法调用，在 2.5 版本中引入这种机制，它允许通过动态方法调用使用“！”运算符控制可以访问的方法。

2.7.7.jQuery

规则 7.7.1：使用 jQuery 3.*以上无漏洞版本。

规则 7.7.2：对于 jQuery 2.*的版本，使用 jQuery-Form-Validator 插件进行输入验证：

https://github.com/victorjonsson/jQuery-Form-Validator/

2.7.8.AJAX

规则 7.8.1：使用.innerText 而不是.innerHtml，使用.innerText 可以防止大多数 XSS 问题，因为它会自动编码文本。

规则 7.8.2：禁止使用 eval。

规则 7.8.3：当使用数据构建 HTML，脚本，CSS，XML，JSON 等时，确保对数据进行适当编码，以防止注入样式问题。

规则 7.8.4：不要依赖客户端逻辑来提高安全性，客户端可以使用一些浏览器插件来设置断点，跳过代码，更改值等。

规则 7.8.5：不要依赖客户端业务逻辑，确保在服务器端验证逻辑。

规则 7.8.6：避免编写序列化代码。

规则 7.8.7：避免动态构建 XML 或 JSON，可能导致 XML 注入错误，如果必须用，要使用编码库或安全的 JSON 或 XML 库来保证属性和元素数据的安全。

规则 7.8.8：不要把敏感信息存储在客户端本地。

规则 7.8.9：不要在客户端代码中执行加密，使用 TLS / SSL 并在服务器上加密.

规则 7.8.10：防止旧版浏览器的 JSON 劫持，可以提醒用户更新到新版浏览器。

2.8. 安全需求

安全功能需求项主要包括:

2.8.1.登录/注册

涉及范围	应用中有登陆、注册环节时，确认以下安全需求	
安全需求	**需求子项**	**需要**
登录/注册	注册页面有验证码，登录失败 3 次后，需要出现图形验证码	
	登录过程使用 https 安全通道	
	检查登录成功后跳转的 URL，是否在白名单内	
	登录的账号认证只能由 xxxx 登录服务器认证，不能经过第 3 方转发验证	
	对外项目：统一使用 UPI 提供的登录验证	
	登录需要记录日志，日志记录以下信息: 登录时间，登录 IP（内部应用记录内部 IP）	

2.8.2.信息使用

涉及范围	用户隐私数据的使用	
安全需求	**需求子项**	**需要**
用户信息使用	身份证，银行帐号，信用卡号，密码不能明文在页面显示	
	密码不能明文保存，其他建议加密保存	
	收集信息时，需要明确告之用户获取用户数据的方式/内容	
	收集信息时，需要明确告之用户获取数据的用途	

	收集信息时，用户对自己的隐私数据用可操作权限（如是用户决定否输入/是否显示等）	

2.8.3.内部信任

涉及范围	内部应用跟 UPI 进行免登的需求	
安全需求	**需求子项**	**需要**
内部信任免登陆	信任免登只能是银联内部应用，其他的信任免登需要评估	
	内部调用需要有身份认证，能明确标识出调用者	
	免登 token 需要有时效性限制,并不能被伪造,失效时间<=30 分钟	
	免登参数传递需要做签名验证	

2.8.4.嵌套(iframe)

涉及范围	针对需要在应用页面引入第三方面的页面	
安全需求	**需求子项**	**需要**
嵌套第三方	合作前需要跟第三方签订相关安全合作协议	
	需要将合作方 URL 提供给信息安全工作组进行评估	

2.8.5.开放接口

涉及范围	针对外部应用/合作伙伴需要提供接口的地方	
安全需求	**需求子项**	**需要**
开放接口	接口以 HTTP（S）方式开放	
	设计接口需要有身份认证，对来源授权，只允许授权的 IP 访问	
	关键的（功能）接口调用需要有日志记录	
	参数传递需要做签名，签名需要有失效性，建议失效时间：<=30 分钟	

2.8.6.权限设计

涉及范围	针对各种需要对数据进行加/解密的地方的需求	
安全需求	**需求子项**	**需要**
权限设计	明确用户身份的操作权限，客户端中有标志用户权限的 cookie 选项需做加密处理或者把权限控制放在服务端	
	垂直权限控制：数据权限做严格分级，用户只能访问自己具有相应权限的数据，防止用户越权	
	水平权限控制：对数据的访问加拥有者判断，用户只能访问自己拥有授权的数据，防止用户访问到和自己同等权限的其他用户数据	

2.8.7.传输存储

涉及范围	针对应用在网络中传递重要用户信息时的需求	
安全需求	**需求子项**	**需要**
信息传输存储	对重要的数据需要进行加密传输 注：重要信息涉及范围：身份证，银行帐号，信用卡号；涉及交易的信息等	

2.8.8.日志记录

涉及范围	所有涉及用户关键性的业务操作应用	
安全需求	**需求子项**	**需要**
日志记录	需要能记录用户操作的时间，IP ，事件，成功，失败 （如考虑性能，此功能可作为开关，必要时提供）	

2.8.9.违禁信息

涉及范围	所有涉及用户文字输入（如评论、发帖等）或图片上传的应用	
安全需求	**需求子项**	**需要**

违禁信息控制	需要对文本类信息进行监控和过滤	

2.8.10. 注册欺诈

涉及范围	所有涉及用户注册或可以修改显示名称的应用	
安全需求	**需求子项**	**需要**
违禁信息控制	需要对允许注册和显示的名称进行必要限制，如进行关键词过滤，“系统管理员”、“xxxx 管理员”等不允许注册	

2.8.11. 违规处罚

涉及范围	各应用针对用户操作进行管理控制的需求	
安全需求	**需求子项**	**需要**
违规用户处罚	针对应用的业务特点，建立一套“惩罚规则”机制，对用户违规行为进处理	

2.8.12. 交易操作

涉及范围	所有涉及用户重大利益（密码、转账限额等）的敏感操作	
安全需求	**需求子项**	**需要**
重大利益（金钱）操作	涉及到重大利益（密码、转账限额等）相关的操作必须接入二次验证平台（需要密保、手机、证书等 2 次校验方能进行操作）	

2.9. 安全技术

安全技术需求项主要包括:

2.9.1.域名需求

涉及范围	项目需要申请新域名或是原有域名的记录有变更	
安全需求	**需求子项**	**需要**
域名申请和变更	xxxx 域名只可以托管在 xxxx 授权的 NameServer	
	xxxx 域名只可以解析到 xxxx 指定的 IP	
	xxxx 提供服务器和 IP 资源，第三方负责开发和运维的模式，禁止使用 xxxx 的域名（主要适用于第三方合作）	
	外包模式，只有同时满足以下三点，方可使用 xxxx 域名 xxxx 拥有代码所有权 后期的运维由 xxxx 来做 通过了 xxxx 的安全验收	

2.9.2.架构需求

涉及范围	系统部署和 ACL(Access Control List)的开通	
安全需求	**需求子项**	**需要**
系统部署 ACL 开通	系统新申请的服务器放在指定 VLAN, 如有违背生产网安全域原则的 ACL 需求，需经技术安全部门批准	
	生产服务器默认只对互联网开放 80，443 端口，其他端口的开放需求需要信息安全工作组审批	
	xxxx 提供服务器和 IP 资源，第三方负责开发和运维的模式，原则是放在 DMZ 区域	
	外包开发的系统，原则上放在 DMZ 区域	
	放在 DMZ 区域的服务器不可能访问非 DMZ 区域服务器的任何端口	
	服务器之间调用的固定 TCP / UDP 协议端口，需开启主机安全策略仅允许访问该端口	

2.9.3.Java 专项

涉及范围	xxxx 内部及第三方合作项目的 Java 应用	
安全需求	**需求子项**	**需要**
Java 应用专项	Java 版本控制（1.8 以上版本 ）、Java 配置安全加固、Java 自动化代码检查	

	使用 Java 安全框架和组件包、限制部署的应用程序是否使用	

2.9.4.接口认证

涉及范围	涉及需要为其他应用提供访问接口时	
安全需求	**需求子项**	**需要**
接口认证	接口需要检查调用者的身份，提供身份/来源认证接口	
	身份的认证方式：签名/IP 来源控制	

2.9.5.密钥存储

涉及范围	针对各种需要对数据进行加/解密的地方，密钥保存	
安全需求	**需求子项**	**需要**
密钥存储	使用公私钥的加密方式，使用 keystore 存放 key	
	使用字符串作密钥的，应用自行保存，设置好文件权限	

2.9.6.Cookie

涉及范围	所有需要通过 cookie 跟踪用户状态的应用	
安全需求	**需求子项**	**需要**
Cookie	禁止在 Cookie 中记录机密信息，如用户密码，信用卡号，银行卡号等,需要保密或者业务逻辑相关的数据需要加密或签名保存	
	和登录认证相关的 Cookie 必须设置 httponly 属性为 true	
	只在 https 环境下传递的 Cookie 需要设置 secure 属性为 true(ssl)	

2.9.7.数据字段

涉及范围	所有适用数据存储业务数据的应用	
安全需求	**需求子项**	**需要**
数据库字段	明确字段信息的安全级别，同时在数据库中标识	
	安全级别的字段，需要加密保存	

2.9.8.加密解密

涉及范围	所有需要在应用间进行加密的通信	
安全需求	**需求子项**	**需要**
加密解密算法	非对称加密算法推荐 RSA（2048bits），对称加密算法推荐 AES	
	对于需要传递的敏感信息，使用 AES 算法加密和解密	
	不得使用自定义的加密算法。如有自定义加密算法的应用，需通知技术安全部门审核	

2.9.9.参数传递

涉及范围	在页面之间通过 URL 传递关键性的参数	
安全需求	**需求子项**	**需要**
参数传递	生成关键性参数时进行签名，接受参数页面进行签名验证	
	签名方法，推荐两种签名算法方式（任选一） A）md5 (SECRET+ Parameters+ SECRET),SECRET 为协定好的密钥 B）hmac(Parameters+SECRET) , SECRET 为协定好的密钥	
	签名需要有时效性，失效时间：小于等于 30 分钟	

2.9.10. 框架组件

涉及范围	xxxx 内部及第三方合作项目的应用	
安全需求	**需求子项**	**需要**
框架组件版本	使用的开源框架组件，只允许无漏洞或补丁更新的版本	
	框架使用的依赖库，只允许无漏洞或补丁更新的版本	

2.9.11. 漏洞防御

涉及范围	xxxx 内部及第三方合作项目的应用	
安全需求	**需求子项**	**需要**
安全漏洞防御	XSS 防御，对用户的输入内容，在输出时进行 HTML / JS 编码	
	CSRF 防御 ，在功能页面设置 csrf token，提交请求时检查 csrf token 的合法性再执行操作	
	SQL 注入防御，拼接 SQL 语句参数时，使用参数化的方式构造 SQL 语句	
	URL 跳转漏洞防御，设置 URL 跳转的域名白名单，检查只有在白名单中的域名 URL 才允许跳转	
	文件上传漏洞防御，严格控制允许上传的文件类型，检查用户上传的文件内容	
	信息泄露问题防御，限制敏感信息读取的范围、路径/目录，删除服务器上的测试文件	
	Flash 安全，如无 flash 应用，禁止配置 crossdomain.xml 文件；如有，需定义好允许的跨域请求的域；如无特殊情况下，allowscriptaccess 设置为 never，allowNetworking 设置为 none 用户上传的 flash 文件需要单独域名保存	
	通过正则表达式或自实现工具类对所有非法字符进行过滤	

2.10. 攻击行为

安全行为防御需求项主要包括:

2.10.1. 短信验证

涉及范围	应用中有短信验证环节时，确认以下安全需求	
安全需求	**需求子项**	**需要**
防短信验证绕过	短信验证码至少 6 位以上数字	
	短信验证码的失效时间不超过 3 分钟	
	连续输错 3 次验证码暂时锁定继续操作，锁定时间 10 分钟	
	验证码必须确保唯一有效，不能复用	
	验证码在客户端和服务端都要验证	
	对连续试错的用户进行违规处罚，长时间锁定	

2.10.2. 用户撞库

涉及范围	应用中有登录/注册环节时，确认以下安全需求	
安全需求	**需求子项**	**需要**
防用户撞库	识别批量注册和登录行为，进行风控管理	
	对同一 IP 批量注册和登录设置阈值，启用锁定策略	
	识别短信批量发送行为，进行风控管理	
	对同一 IP 批量请求发送短信设置阈值，启用锁定策略	
	对登录注册机制进行分步逻辑，完成第一步后才能进入第二步	
	对连续试错的用户进行违规处罚，长时间锁定	

2.10.3. 爆破密码

涉及范围	应用中有登录环节时，确认以下安全需求	
安全需求	**需求子项**	**需要**
防爆破密码	输错一次密码后弹出验证码验证	
	连续输错 3 次密码暂时锁定继续操作，锁定时间 10 分钟	
	启用密码复杂性策略避免用户使用容易被破解的密码	
	对同一 IP 批量请求登录设置阈值，启用锁定策略	
	向用户提供登录来源 IP 记录，方便用户自己做安全审计	
	对连续试错的用户进行违规处罚，长时间锁定	

2.10.4. ID 遍历

涉及范围	应用中有用户个人信息，订单信息功能时，确认以下安全需求	
安全需求	**需求子项**	**需要**
防 ID 遍历	ID 必须进行编码或使用加密算法处理	
	ID 不能存在连续递增情况，避免可预测	
	ID 最好包含数字和字母两种字符，长度最好超过 32 位	
	对 ID 访问设置失效性，过期时间 30 分钟	
	对同一 IP 批量请求设置阈值，启用锁定策略	

	对连续试错的用户进行违规处罚，长时间锁定	

2.10.5. 风控缺失

涉及范围	应用中有登录 / 密码找回等功能时，确认以下安全需求	
安全需求	**需求子项**	**需要**
风控措施	对用户注册、登录到注销整个环节进行审计	
	建立用户违规处罚机制，比如故意试错、批量尝试等	
	识别批量注册行为，启用锁定策略	
	识别批量登录行为，启用锁定策略	
	识别批量修改用户信息行为，启用违规机制	
	识别批量交易行为，启用违规处罚机制	

2.11. 安全组件

安全组件需求项主要包括:

2.11.1. API 安全

涉及范围	应用中有 API 接口时，确认以下安全需求	
安全需求	**需求子项**	**需要**
API 安全	API 接口调用必须启用身份认证，仅对授权来源调用	
	要考虑 JWT (JSON Web Token)被破解的情况	
	要考虑 OAuth 授权或认证协议被破解的情况	
	使用 HTTPS 防止中间人攻击	
	对用户的输入进行验证	
	强制使用安全的 HTTP 头响应给客户端	

2.11.2. Spring Boot

涉及范围	应用中使用 Spring Boot 时，确认以下安全需求	
安全需求	**需求子项**	**需要**
Spring	通过扩展 WebSecurityConfigurerAdapter 要求安全连接，强制使用	

Boot	HTTPS	
	对依赖库进行漏洞检测，及时更新有漏洞的组件	
	使用 Spring MVC 的<form:form>标记或 Thymeleaf 和 @EnableWebSecurity 防止 CSRF	
	配置应用程序以返回 Content-Security-Policy 头，来开启内容安全策略 （CSP）来缓解 XSS（跨站点脚本）和数据注入攻击	
	使用 OpenID Connect 进行身份验证	
	使用 MD5+Salt 等自定义的加密方式	
	更新到无漏洞的版本，和支持版本兼容更新机制	
	对一些敏感数据，需要加密存储，不允许明文存储	
	打包成 war 包，配置 Tomcat 以服务方式运行	

2.11.3. 其他组件

涉及范围	应用中使用以下开源组件时，确认以下安全需求	
安全需求	**需求子项**	**需要**
组件安全	使用 Spring Security 做为安全框架时确保实现输入过滤工具类	
	使用 Hibernate 时，使用参数化查询方式避免 SQL 注入漏洞	
	使用 MyBatis 时，生成 PreparedStatement 参数方式避免 SQL 注入漏洞	
	使用 JPA 时，启用参数绑定方式避免 SQL 注入漏洞	
	调用 SQL 操作时，使用参数化查询方式避免 SQL 注入漏洞	
	使用 Struts2 时，必须禁止 devMode、使用 Struts 标记而不是原始 EL 表达式等方式避免代码注入漏洞	
	使用 jQuery 3.*以上无漏洞版本	
	使用 AJAX 时，禁止使用 eval、避免动态构建 XML 或 JSON 等方式防止代码注入漏洞	

2.12. 最低需求

针对研发安全流程等级评审结果为部分流程的系统，如下两种情况：

部分流程（一）：二级系统（非互联网）可只选最低安全需求项、快速安全测试等部分流程、中危以上漏洞修复完即可上线；

部分流程（二）：建设周期短要求尽快上线，先选最低安全需求项、快速安全测试等部分流程、中危以上漏洞修复完即可上线，上线后还需重新做深度安全测试（源码审计和黑盒安全测试），并修复发现的漏洞；

可以选择最低安全需求项即可。

3. WEB 安全测试规范

3.1. 安全测试

3.1.1.适用范围

本规范的制定考虑了 xxxx 各种 Web 应用安全测试的共性，适合于 xxxx 绝大部分 Web 应用系统的安全测试，要求 Web 应用系统测试必须遵循。

3.1.2.用词约定

规则：强制必须遵守的原则；

建议：需要加以考虑的原则；

说明：对此规则或建议进行相应的解释；

实施指导：对此规则或建议的实施进行相应的指导；

3.1.3.规范评审

根据研发安全流程控制活动要求，项目进行测试阶段后，根据测试规范，开发团队项目经理或指定接口人通知安全测试人员、信息安全工作组（安全测试人员）进行安全测试，开发团队对于安全测试人员、信息安全工作组发现的测试结果进行确认和漏洞修复，同时等开发团队修复漏洞后进行复测支持。

3.1.4.测试规范

安全测试规范建立：安全测试规范由信息安全工作组提供技术支持、主要包括安全需求满足情况、安全漏洞测试，其中安全需求满足情况通过人工审核代码确认，安全漏洞测试是通过源代码安全漏洞扫描、黑盒测试、人工代码审计等手段进行，规范由信息安全工作主要维护并提供版本更新。

3.1.5.规范分类

xxxx 研发安全流程所提安全测试规范主要包括针对：开发组件和依赖库版本漏洞扫描、安全功能需求实现情况、安全技术需求实现情况等安全需求完成情况；以及针对 Web 安全漏洞的源码扫描、黑盒测试、业务逻辑测试等具体测试条目和测试方法的落地进行具体要求和指引。

3.2. 安全漏洞

3.2.1.Web 安全漏洞

针对众多的 Web 漏洞，OWASP 的专家们结合各自在各领域的应用安全工作经验及智慧，提出了十大 Web 应用程序安全风险，帮助人们关注最严重的漏洞。（OWASP 即开放 Web 应用安全项目，是一个旨在帮助人们理解和提高 Web 应用及服务安全性的项目组织。）xxxx 维护的威胁风险库中，Web 安全漏洞基于 OWASP１０大应用风险的细分项：

序号	风险名称	风险描述
1	SQL 注入	攻击者利用注入漏洞，通过构造特殊的语句，可进行后台数据库操作并插入木马，以获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。

2	跨站脚本	攻击者利用此漏洞，可以获取其他合法用户的 Cookie 身份信息、访问地址等，通过获取到的 Cookie 信息，即可以被攻击者的身份访问 Web 应用，如获取到管理员的 Cookie，就可以以管理员的身份访问应用系统。
3	任意文件上传	攻击者利用此漏洞，可直接上传木马文件，以获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
4	XML 实体注入	攻击者利用此漏洞，通过构造特殊的引用或请求可以读取服务器文件或执行操作系统命令，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
5	EL 表达式注入	攻击者利用 EL 表达式的方法，EL 表达式语法允许开发人员开发自定义函数，以调用 Java 类的方法。语法：${prefix：method(params)}。在 EL 表达式中调用的只能是 Java 类的静态方法，这个 Java 类的静态方法需要在 TLD 文件中描述，才可以被 EL 表达式调用。EL 自定义函数用于扩展 EL 表达式的功能，可以让 EL 表达式完成普通 Java 程序代码所能完成的功能。从而攻击者可以构造代码执行，获取系统的执行命令权限。如果这些信息有助于攻击者进一步攻击利用，可能导致更严重的危害。（存在风险点的位置：用于表达式的运算。如：加、减、乘、除,用于从作用域中取出数据）
6	账号弱口令	攻击者利用弱口令，可以获取特定账户或应用的访问控制权限，如果进一步攻击利用可能获取服务器控制权限。
7	跨站请求伪造	攻击者利用此漏洞，可以以受害者的身份发起 HTTP 请求，访问 Web 应用的相应功能，如果利用成功，那么攻击者就可以更新用户的相关数据（添加、删除、修改），如添加管理员账户、向指定账户转账。
8	文件包含	攻击者利用此漏洞，可以执行任意代码，获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。

9	目录遍历	攻击者利用此漏洞，可以操作服务器指定文件，特别是一些敏感文件，如程序配置文件、数据库配置文件、程序代码文件、服务器账户文件等；如果进一步利用可获取服务器敏感数据或获取服务器控制权限。
10	服务端请求伪造	攻击者利用此漏洞，可以利用 Web 服务器发起请求（HTTP、FTP、HTTPS 等等），可以突破内外网访问控制、部分安全控制措施，如果进一步利用可获取内网服务器权限及敏感数据。
11	Java 反序列化	攻击者利用此漏洞，通过构造特殊的引用或请求可以读取服务器文件或执行操作系统命令，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。（导入模版文件、网络通信、数据传输、日志格式化存储、对象数据落磁盘或 DB 存储等业务场景等）。
12	有漏洞的组件	攻击者利用此漏洞，可执行恶意操作系统命令，获取整个网站和数据库的控制权限，包括修改删除网站页面、窃取数据库敏感信息；甚至以网站为跳板，获取整个内网服务器控制权限。
13	敏感信息泄漏	攻击者利用此漏洞，可以访问到程序备份文件，可能导致源代码或者数据泄露，如果进一步被利用可能导致更严重的危害。
14	不安全加密算法	攻击者利用此漏洞，可以获取到对应的铭感信息进行下一步的攻击。
15	任意文件下载	攻击者利用此漏洞，可以下载服务器任意文件，如脚本代码，服务及系统配置文件等；可用得到的代码进一步代码审计，得到更多可利用漏洞。
16	任意密码找回	攻击者利用此漏洞，可以找回任意账户的密码。
17	HTML 注入	攻击者利用此漏洞，可以访问引导用户访问恶意的网页，如果这个网页是钓鱼、挂马的网页，可能导致更严重的危害。

18	隐藏暗链	暗链对政府网站来讲危害更大，当用户通过搜索引擎搜索某地政府网站时，可能从搜索引擎的描述中看到一些非法的关键词，从而影响政府的形象。境外敌对势力甚至可以根据政府网站是否存在暗链来判断一个政府网站是否存在安全漏洞，并决定是否发起攻击。成功入侵后，可以发布虚假政策信息造成群众对政府的不信任，制造政府与群众之间的矛盾，引发社会事件。
19	未授权访问	攻击者利用此漏洞，可以直接访问相应的页面，如果此页面存在敏感数据或信息，那么这些敏感数据或信息可能发生泄露，如果这些信息有助于攻击者进一步攻击利用，可能导致更严重的危害。
20	越权访问	攻击者利用此漏洞，可以访问其他用户的数据，可能导致数据泄露，如果进一步被利用可能导致更严重的危害。
21	固定 SessionID 漏洞	攻击者可以简单的伪造一个 SessionID 诱使用户使用该 SessionID 登录，即可获取登录权限。
22	URL 重定向	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。
23	Flash 访问	攻击者利用此漏洞，可以以受害者的身份发起 HTTP 请求，访问 Web 应用的相应功能，如果利用成功，那么攻击者就可以更新用户的相关数据（添加、删除、修改），如添加管理员账户、向指定账户转账。

3.2.2.业务场景漏洞

业务场景漏洞基于业务模块功能可能产生的漏洞细分项：

序号	风险名称	风险描述

1	用户登录	攻击者通过用户登录模块漏洞，可以获取特定账户或应用的访问控制权限，如果进一步攻击利用可能获取服务器控制权限。
2	用户注册	攻击者通过用户注册模块漏洞，可以通过恶意注册，结合其他攻击手段进一步利用并扩大漏洞危害。
3	密码找回	攻击者利用找回密码用其他攻击进行下一步的攻击操作，甚至攻击成功的话，可能导致获取网站用户登录的部分权限，包括修改删除用户信息、窃取用户敏感信息。
4	后台管理	攻击者利用后台管理漏洞，可以通过登录后台或者 url 跳转等操作，对服务器指定文件，特别是一些敏感文件，如程序配置文件、数据库配置文件、程序代码文件、服务器账户文件等，从而达到下一步的攻击。
5	接口调用	可以通过攻击接口，获取敏感的数据或者能访问一些访问权限。
6	会员系统	通过攻击会员系统，可以越权访问权限，个人信息，图片上传，从而获取下一步的攻击，结合其他安全漏洞可能造成更严重的危害。
7	业务办理	攻击者通过攻击业务办理模块，可能存在替代办理、业务绕过、篡改他人的信息。
8	业务查询	攻击者利用此漏洞，可能会造成恶意查询或者可能办理人的信息泄露，如果进一步被利用可能导致更严重的危害。
9	业务传输	攻击者通过攻击 cookie，劫持 cookie，替换会话等攻击手段，从而进一步攻击利用可能获取服务器控制权限。

10	发表评论	攻击者利用发表评论漏洞，可能访问其他用户的数据，可能导致数据泄露，如果进一步被利用可能导致更严重的危害。
11	购买支付	攻击者利用此漏洞，可以篡改商品的订单，金额，遍历交易信息。如果进一步篡改可导致平台上损失。
12	账号充值	攻击者通过账号充值，对充值功能模块进行虚拟充值，篡改充值，篡改账号等，从而导致充值业务被破坏。
13	抽奖活动	攻击者可以通过抽奖活动，进行薅羊毛，盗刷积分，刷取奖品等攻击行为，从而导致正常的抽奖活动被破坏。
14	代金优惠	攻击者通过攻击代金优惠，可以进行对代金业务进行盗刷，篡改等，甚至会给平台带来一定的损失。
15	订单信息	如果订单泄露，可能会造成个人的敏感信息泄露。如果进一步被利用可能导致更严重的危害。
16	运费账单	攻击者利用运费账单漏洞，通过运费的漏洞利用，篡改或者构造假账单，从而让平台造成一定的损失。
17	第三方商家	攻击者利用平台上的第三方商家，可能会造成第三方商家被盗号，商家账号被遍历，或者越权访问其他商家的用户信息，甚至可能窃取用户敏感信息。

3.2.3.攻击行为风险

业务攻击风险基于业务场景功能可能产生的风险细分项：

序号	风险名称	风险描述
1	短信验证	攻击者利用验证码爆破漏洞，通过构造的密码字典，对用户的验证码进行暴力猜测，可以获取网站用户登录的部分权限，包括修改删除用户信息、窃取用户敏感信息。(存在风险点的位置：登录处、注册处)。

2	用户撞库	攻击者撞库成功后,可以获取网站用户登录的部分权限，包括修改删除用户信息、窃取用户敏感信息。(存在风险点的位置：登录处、注册处)
3	爆破密码	攻击者利用弱口令漏洞，通过进行暴力猜解,获取网站管理权限，包括修改删除网站页面、窃取数据库敏感信息、植入恶意木马；甚至以网站为跳板，获取整个内网服务器控制权限。
4	ID 遍历	攻击者利用该漏洞，通过遍历个人信息，可能获取网站帐户等敏感信息;通过结合其他漏洞可能造成更严重的危害。（存在风险点位置:用户个人信息，订单信息等。）
5	风控缺失	攻击者利用此漏洞,通过脚本和程序进行批量的用户注册和登录操作,从而进行下一步的攻击，如薅羊毛等攻击。（存在分析点位置：登录处，密码找回处，用户注册处。）

3.3. 漏洞等级

根据漏洞的危害程度将漏洞等级分为【严重】、【高】、【中】、【低】四个等级，由 xxxx 结合实际利用场景中漏洞的严重程度、可利用指数等综合因素给予漏洞定级，每种等级包含的评分标准及漏洞类型如下，例如严重等级漏洞影响:

涉及可大批量获取账号信息、控制用户权限等漏洞;

涉及可大批量获取用户敏感信息，如订单信息等漏洞;

涉及可获取重要服务器控制权限等漏洞。

3.3.1.严重漏洞

1、直接获取核心系统权限的漏洞（服务器权限、数据库权限）。包括但不仅限于远程命令执行、任意代码执行、上传获取 Webshell、SQL 注入获取系统权限、缓冲区溢出。

2、直接导致业务拒绝服务的漏洞。包括但不仅限于直接导致移动网关业务 API 业务拒绝服务、网站应用拒绝服务等造成严重影响的远程拒绝服务漏洞。

3、严重的敏感信息泄漏。包括但不仅限于核心 DB（资金、身份、交易相关） 的 SQL 注入，可获取大量核心用户的身份信息、订单信息、银行卡信息等接口问题引起的敏感信息泄露。

4、严重的逻辑设计缺陷和流程缺陷。包括但不仅限于通过业务接口批量发送任意伪造消息、任意账号资金消费、批量修改任意帐号密码漏洞。

3.3.2.高危漏洞

1、敏感信息泄漏。包括但不仅限于非核心 DB SQL 注入、源代码压缩包泄漏、服务器应用加密可逆或明文、移动 API 访问摘要、硬编码等问题引起的敏感信息泄露。

2、敏感信息越权访问。包括但不仅限于绕过认证直接访问管理后台、后台弱密码、获取大量内网敏感信息的 SSRF。

3、直接获取系统权限的漏洞（移动客户端权限）。包括但不仅限于远程命令执行、任意代码执行。

4、越权敏感操作。包括但不仅限于账号越权修改重要信息、进行订单普通操作、重要业务配置修改等较为重要的越权行为。

5、大范围影响用户的其他漏洞。包括但不仅限于可造成自动传播的重要页面的存储型 XSS（包括存储型 DOM-XSS）和涉及交易、资金、密码的 CSRF。

3.3.3.中危漏洞

1、需交互方可影响用户的漏洞。包括但不仅限于一般页面的存储型 XSS、反射型 XSS（包括反射型 DOM-XSS）、重要操作 CSRF、URL 跳转漏洞。

2、普通越权操作。包括但不仅限于不正确的直接对象引用。影响业务运行的 Broadcast 消息伪造等 Android 组件权限漏洞等。

3、普通信息泄漏。包括但不仅限于客户端明文存储密码、客户端密码明文

传输以及 web 路径遍历、系统路径遍历。

4、普通的逻辑设计缺陷和流程缺陷。

3.3.4.低危漏洞

1、本地拒绝服务漏洞。包括但不仅限于客户端本地拒绝服务（解析文件格式、网络协议产生的崩溃），由 Android 组件权限暴露、普通应用权限引起的问题等。

2、轻微信息泄漏。包括但不仅限于路径信息泄漏、SVN 信息泄漏、phpinfo、异常信息泄露，以及客户端应用本地 SQL 注入（仅泄漏数据库名称、字段名、cache 内容）、日志打印、配置信息、异常信息等。

3、难以利用但存在安全隐患的漏洞。包括但不仅限于难以利用的 SQL 注入点、可引起传播和利用的 Self-XSS、需构造部分参数且有一定影响的 CSRF。

3.4. 基本信息

3.4.1.语言和组件

xxxx 业务系统采用的主要开发语言和调用的框架或组件如下，每次安全测试需要识别出主要调用框架和组件，并做为测试报告的一部分：

序号	类别	详细描述
1	主要语言	Java,JavaScript
2	附加语言	peoplecode，Objective-C
3	调用框架组件	Spring，Hibernate，JeeSite，MyBatis，FastJson，Shiro，jQuery，SpringMVC，Struts2，supcan，dojo，Struts，freemarker，sonarqube，dubbo，spring-data-jpa，emoss，用友
4	应用服务器	Tomcat，WebLogic，Domino，WebSphere

3.4.2.版本更新要求

调用的开源框架不能使用有漏洞的版本，需到官方下载当前最新的版本，当前最新的版本在业务上线前有漏洞爆出时需更新到最新的无漏洞的版本，可以使用开发工具插件检查各模块最新版本，各组件官方下载地址如下，该项通过依赖库扫描，扫描结果做为测试报告的一部分：

序号	组件名称	官网下载地址
1	Spring	http://spring.io/projects/spring-framework https://projects.spring.io/spring-security/
2	Hibernate	http://hibernate.org/orm/downloads/ http://hibernate.org/validator/
3	JeeSite	http://www.jeesite.com/
4	MyBatis	http://www.mybatis.org/mybatis-3/
5	FastJson	https://github.com/alibaba/fastjson
6	Shiro	http://shiro.apache.org/
7	jQuery	http://jquery.com/
8	Struts2	https://struts.apache.org/ https://struts.apache.org/security/
9	Spring boot	http://spring.io/projects/spring-boot
10	dojo	https://dojotoolkit.org/download/
11	freemarker	http://freemarker.org/freemarkerdownload.html
12	sonarqube	https://www.sonarqube.org/
13	dubbo	http://dubbo.apache.org/zh-cn/
14	spring-data-jpa	http://spring.io/projects/spring-data-jpa

3.4.3.应用服务器要求

应用服务器软件不能使用有漏洞的版本，建议更新到最新的无漏洞的版本，各应用服务器软件安全公告官网如下，扫描应用服务器安全漏洞，扫描结果做为测试报告的一部分：

序号	开发工具	官网下载地址
1	Tomcat	http://tomcat.apache.org/

		https://tomcat.apache.org/security.html
2	WebLogic	https://www.oracle.com/middleware/technologies/weblogic.html https://docs.oracle.com/cd/E24329_01/web.1211/e24446/security.htm#INTRO232
3	WebSphere	https://www.ibm.com/developerworks/cn/websphere/ https://www.ibm.com/developerworks/websphere/zones/was/security/index.html
4	Domino	https://www.ibm.com/us-en/marketplace/ibm-domino https://www.ibm.com/security/secure-engineering/bulletins.html

3.4.4.依赖库版本扫描

对系统调用的各依赖 jar 包及组件需要通过 OWASP Dependency Check 做版本漏洞扫描，确保当前使用的各依赖包版本最新的无漏洞的版本，OWASP Dependency Check 使用方法如下:

dependency-check.bat --project test -s "C:\repository"

然后会在当前目录生成报告文件，扫描结果做为测试报告的一部分。

3.5. 通用规范

规则：通用的安全测试规范包括测试工具介绍、测试项明确、测试方法定义、测试点明确、测试报告输出、回归测试报告输出、测试结果解释等，安全测试规范将针对业务系统安全测试过程中的落地进行具体要求。

3.5.1.测试工具

静态应用安全测试工具：Static Application Security Testing (SAST) 静态应用安全测试工具，也就是 Source Code Analysis Tools (SCA) 源代码分析工具。

动态应用安全测试工具：Dynamic Application Security Testing (DAST) 动态应用安全测试工具，也就是 Vulnerability Scanning Tools (VST) 漏洞扫描工具。

依赖库检测工具：Dependency-Check (DC) 依赖库检测工具，以及平台化的 Dependency-Track (DT) 依赖库跟踪工具。

针对 Web 安全测试的工具繁多，安全测试主要测试安全漏洞为主，因此安全测试至少经过以下工具的扫描和验证。

序号	测试工具	工具描述
1	Eclipse	用于导入代码做源代码分析，官方下载地址：https://www.eclipse.org/downloads/
2	Find Security Bugs	用于执行源码安全漏洞扫描，官方下载地址：https://find-sec-bugs.github.io/download.htm
3	Fortify Static Code Analyzer	用于执行源码安全漏洞扫描，可以申请试用：https://www.microfocus.com/en-us/products/static-code-analysis-sast/overview
4	OWASP Dependency-Check	用于执行依赖库安全漏洞扫描，官方下载地址：https://www.owasp.org/index.php/OWASP_Dependency_Check
5	Nessus	用于执行系统和应用服务器安全漏洞扫描，可以申请试用（已购买）：https://www.tenable.com/products/nessus/nessus-professional
6	IBM Security AppScan	用于执行黑盒 Web 安全漏洞扫描，可以申请试用（已购买）：https://www.ibm.com/developerworks/cn/downloads/r/appscan/
7	Burp Suite Scanner	用于执行动态 Web 安全漏洞测试，可以申请试用：https://portswigger.net/burp/

8	OWASP Zed Attack Proxy (ZAP)	用于执行黑盒 Web 安全漏洞扫描，官方下载地址：https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
9	Telerik Fiddler	用于执行动态 Web 安全漏洞测试，官方下载地址：https://www.telerik.com/fiddler
10	sqlmap	用于验证 SQL 注入漏洞测试，官方下载地址：http://sqlmap.org/

3.5.2.测试清单

针对 Web 的安全测试项主要包括：

序号	测试清单	清单项描述
1	依赖库扫描	编码开始阶段即可执行安全组件识别和依赖扫描
2	安全需求验证	针对立项时提的安全需求项逐项测试安全需求满足情况，参考《安全需求评审表》
3	安全漏洞测试	通过源码扫描工具、黑盒扫描工具、动态测试工具进行安全漏洞测试，参考《安全测试指南》
4	框架和组件统计	通过源码分析识别使用的主要框架和组件版本

5	互联网泄露信息	针对该项目信息搜索互联网上代码仓库泄露情况

3.5.3.测试方法

针对 Web 的安全测试方法主要包括:

序号	测试方法	方法描述
1	依赖库扫描	更新 OWASP Dependency-Check 漏洞库后，对项目完整依赖包执行扫描，输出 HTML 原始报告，并人工整理验证结果到 Word 报告。
2	安全需求验证	通过 Eclipse 导入源码和搭建的测试环境进行具体安全需求项验证，并人工整理验证结果到 Word 报告。
3	安全漏洞测试	通过 Find Security Bugs、Fortify（可选）、AppScan、Burp Suite、ZAP、sqlmap 等进行安全漏洞扫描和人工截取数据包测试，各工具输出 HTML 原始报告，并人工整理验证结果到 Word 报告。
4	框架和组件统计	通过查看 pom.xml 依赖包版本信息，结合具体源码分析，整理出主要框架版本和组件信息，并人工整理验证结果到 Word 报告。
5	互联网泄露信息	通过人工搜索互联网上开源代码托管仓库 GitHub 中可能存在的项目相关关键字，并人工整理验证结果到 Word 报告。

3.5.4.测试报告

测试报告主要以 Word 版本提交，包括详细测试过程和测试结果:

序号	报告章节	报告描述
1	基本信息	统一为 xxxx 报告封面、报告标题为：安全测试报告、报告描述、报告日期、测试参考依据等
2	测试目标	测试目标系统名称、应用服务器版本、主要框架和组件列表、IP 域名、测试账号、源码包名称等信息
3	结果汇总	对测试结果，包括：依赖库扫描、安全需求验证、安全漏洞测试、框架和组件统计、应用服务器扫描、互联网泄露信息等结果进行汇总，并综合评定风险等级
4	详细结果	对测试结果进行详细描述，包括：需求条目满足情况、安全漏洞触发点、漏洞利用测试、框架和组件版本等，漏洞利用效果必须截图，并保留漏洞利用的原始和测试修改的数据包
5	修复建议	对安全漏洞测试结果，包括：Web 安全漏洞、业务场景漏洞、攻击行为风险提供安全加固建议
6	报告附件	对安全测试过程中产生的过程文件、工具原始测试报告、测试小工具、测试脚本等做为附件一起提供

3.5.5.回归测试

开发团队对漏洞修复后通知回归测试，并输出回归测试报告，报告主要以 Word 版本提交，包括详细测试过程和测试结果：

序号	报告章节	报告描述
1	基本信息	统一为 xxxx 报告封面、报告标题为：回归测试报告、报告描述、报告日期、测试参考依据等

2	测试目标	测试目标系统名称、应用服务器版本、主要框架和组件列表、IP 域名、测试账号、源码包名称等信息
3	结果汇总	对回归测试结果，包括：依赖库扫描、安全需求验证、安全漏洞测试、框架和组件统计、应用服务器扫描、互联网泄露信息等结果进行汇总，并综合评定风险等级
4	详细结果	对回归测试结果进行详细描述，包括：需求条目满足情况、安全漏洞触发点、漏洞利用测试、框架和组件版本等，漏洞利用效果必须截图，并保留漏洞利用的原始和测试修改的数据包
5	修复建议	对安全漏洞测试结果，包括：Web 安全漏洞、业务场景漏洞、攻击行为风险提供安全加固建议
6	报告附件	对回归测试过程中产生的过程文件、工具原始测试报告、测试小工具、测试脚本等做为附件一起提供

3.5.6.结果解释

信息安全工作组负责对安全测试结果做解答，主要包括：

序号	测试结果	结果解释
1	依赖库扫描	针对引用的依赖库版本有漏洞或是版本过低的情况，建议更新到安全的版本
2	安全需求验证	针对立项前评审的安全需求项未满足情况，需要重新完成需求项

3	安全漏洞测试	针对源码扫描和黑盒测试发现漏洞的情况，需要修复漏洞后再回归测试，直到漏洞完全修复
4	框架和组件统计	针对框架和组件版本有漏洞或是版本过低的情况，建议更新到安全的版本
5	互联网泄露信息	针对互联网上有代码和敏感数据泄露的情况，需评估其可能的影响并跟踪删除泄露代码的进展，直到泄露代码完全删除

3.6. Web 安全漏洞

针对 Web 安全漏洞的测试，参考以下要点：

3.6.1.SQL 注入

漏洞产生点
业务系统中存在数据库调用的功能点都有可能存在 SQL 注入漏洞，不限于所有表单和参数等，都需要测试
风险等级-严重
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 sqlmap 验证数据读取 浏览器直接打开请求的 URL

3.6.2.跨站脚本

漏洞产生点
业务系统存在执行 JS 脚本前端或接收用户输入的功能点都有可能存在跨站脚本（XSS）漏洞，不限于所有表单和参数等，都需要测试

风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 浏览器直接打开请求的 URL

3.6.3.任意文件上传

漏洞产生点
业务系统存在接收用户上传文件或图片等功能点都有可能存在任意文件上传漏洞，不限于上传页面和上传接口等，都需要测试
风险等级-严重
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改上传数据包 浏览器直接打开上传的页面

3.6.4.XML 实体注入

漏洞产生点
业务系统存在接收 XML 传输数据的功能点时都有可能存在 XML 实体注入漏洞，不限于接口页面和 API 等，都需要测试
风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改 XML 数据包

3.6.5.EL 表达式注入

漏洞产生点

业务系统存在表达式计算执行功能点时都有可能存在 EL 表达式注入漏洞，不限于框架本身和应用接口页面等，都需要测试
风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改表达式数据包

3.6.6.账号弱口令

漏洞产生点
业务系统存在用户体系和后台管理等功能点时都有可能存在账号弱口令漏洞，不限于登录页面和登录接口等，都需要测试
风险等级-严重
测试方法
浏览器直接打开登录页面 Burp Suite 动态修改登录数据包

3.6.7.跨站请求伪造

漏洞产生点
业务系统存在会话认证机制时都有可能存在跨站请求伪造（CSRF）攻击，不限于会话状态和验证机制，都需要测试
风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的页面

3.6.8.文件包含

漏洞产生点

业务系统存在文件调用功能点时都有可能存在文件包含漏洞，不限于本地文件路径和远程文件路径，都需测试
风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的页面

3.6.9.目录遍历

漏洞产生点
业务系统存在文件读取功能点时都有可能存在目录遍历漏洞，不限于应用文件路径和系统文件路径，都需测试
风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的页面

3.6.10. 服务端请求伪造

漏洞产生点
业务系统中存在调用其他业务系统地址时都可能存在服务器端请求伪造（SSRF）攻击，不限于调用页面和接口等，都需要测试
风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的页面

3.6.11. Java 反序列化

漏洞产生点
业务系统调用的 Java 框架和组件存在反序列化功能时都有可能存在 Java 反序列化漏洞，不限于框架和应用接口，都需要测试
风险等级-严重
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包

3.6.12. 有漏洞的组件

漏洞产生点
业务系统调用核心框架和大量的第三方依赖库时都有可能存在有漏洞的组件，不限于核心框架和依赖组件，都需要测试
风险等级-严重
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包

3.6.13. 敏感信息泄露

漏洞产生点
业务系统存在自开发源码时都有可能存在敏感信息泄露问题，不限于备份文件和 SVN 版本控制文件等，都需要测试
风险等级-严重
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的页面

3.6.14. 不安全加密算法

漏洞产生点
业务系统存在加密解密功能和网络传输时都可能存在使用了不安全的加密算法，不限于加密机制和验证接口，都需要测试
风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包

3.6.15. 任意文件下载

漏洞产生点
业务系统存在接收用户下载文件请求等功能点时都有可能存在任意文件下载漏洞，不限于下载页面和下载接口等，都需要测试
风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改下载数据包 浏览器直接打开下载的页面

3.6.16. 任意密码找回

漏洞产生点
业务系统存在找回密码功能点时都有可能存在任意密码找回漏洞，不限于找回密码页面和接口，都需要测试
风险等级-严重
测试方法
Burp Suite 动态修改业务逻辑数据包 浏览器直接打开找回密码的页面

3.6.17. HTML 注入

漏洞产生点
业务系统存在执行 JS 脚本前端或接收用户输入的表单都有可能存在 HTML 注入漏洞，不限于所有表单和参数等，都需要测试
风险等级-中
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 浏览器直接打开请求的 URL

3.6.18. 隐藏暗链

漏洞产生点
业务系统被搜索引擎收录后，当某些地区 DNS 遭劫持后可能会污染其页面内容，隐藏恶意地址，不限于搜索引擎结果，都需要测试
风险等级-中
测试方法
浏览器直接打开搜索结果的 URL 使用 Telerik Fiddler 抓取搜索结果的请求数据包

3.6.19. 未授权访问

漏洞产生点
业务系统存在授权访问功能点时都有可能存在未授权访问漏洞，不限于功能页面和接口等，都需要测试
风险等级-高
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开功能页面

3.6.20. 越权访问

漏洞产生点
业务系统存在授权访问功能点时都有可能存在越权访问漏洞，不限于功能页面和接口等，都需要测试
风险等级-严重
测试方法
Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开功能页面

3.6.21. SessionID 固化

漏洞产生点
业务系统存在会话认证机制时都有可能存在 Session ID 固化漏洞，不限于会话状态和验证机制，都需要测试
风险等级-中
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的页面

3.6.22. URL 重定向

漏洞产生点
业务系统存在 URL 跳转功能时都可能存在 URL 被恶意重定向漏洞，不限于成功或失败跳转等，都需要测试
风险等级-中
测试方法
Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的页面

3.6.23. Flash 访问

漏洞产生点
业务系统存在 Flash 组件时都有可能产生 Flash 跨站脚本漏洞，不限于 Flash 引用和 crossdomain.xml 配置，都需要测试。
风险等级-中
测试方法
AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的页面

3.7. 业务场景漏洞

针对业务场景或功能漏洞的安全测试，参考以下要点：

3.7.1.用户登录

风险产生点
业务系统存在用户登录功能时都可能存在暴力破解用户名密码、用户撞库、验证码爆破和绕过、手机号撞库、账户权限绕过、任意用户登录(如:修改 uid 的值)、弱口令、匿名登录、未授权登录、动态 key 绕过、任意号码登录（如：注册手机号码和收短信的号码可以篡改）、验证码绕过等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开功能页面

3.7.2.用户注册

风险产生点

业务系统存在用户注册功能时都有可能存在恶意用户批量注册、恶意验证注册账户、存储型 XSS、注册任意账号（如：绕过验证码）等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开功能页面

3.7.3.密码找回

风险产生点
业务系统存在密码找回功能时都有可能存在重置任意用户账户密码、批量重置用户密码、新密码劫持、短信验证码长度（如：6 位数，1 分钟内失效）、用户邮箱劫持篡改、短信验证码 response 回显、修改响应包重置任意账号密码（如：修改 response 的值）、跳过验证步骤重置任意修改账号密码、重置密码链接中 token 值未验证或不失效导致任意账号密码重置等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开功能页面

3.7.4.后台管理

风险产生点
业务系统存在后台管理功能时都可能存在管理员用户名密码绕过(如:万能密码)、目录遍历、url 跳转、后台地址暴露等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开功能页面

3.7.5.接口调用

风险产生点
业务系统存在接口调用机制时都可能存在接口调用重放、接口调用遍历、接口调用参数篡改、接口未授权访问/调用、Callback 自定义、WebService 未授权访问/调用等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.6.会员系统

风险产生点
业务系统存在会员系统功能时都可能存在用户越权访问、个人资料信息、个人资料遍历、用户图片上传、csrf 攻击（表单处）、任意跳转等漏洞，都需要测试 ssrf 攻击
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.7.业务办理

风险产生点
业务系统存在业务办理功能时都可能存在顶替办理、绕过业务流程办理、篡改其他办理人信息、办理人信息泄漏等漏洞，都需要测试

风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.8.业务查询

风险产生点
业务系统存在业务查询功能时都可能存在恶意查询、办理人信息泄漏等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.9.业务传输

风险产生点
业务系统存在业务传输过程时都可能存在 COOKIE 注入、COOKIE 跨站、COOKIE 劫持、明文传输等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.10. 发表评论

风险产生点
业务系统存在发表评论功能时都可能存在 POST 注入、CSRF、存储型 XSS、遍历用户名、遍历留言等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 AppScan 或 Burp Suite 黑盒扫描 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.11. 购买支付

风险产生点
业务系统存在购买支付功能时都可能存在商品金额篡改、商品数量篡改、交易信息泄漏、支付订单遍历等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.12. 账号充值

风险产生点
业务系统存在账号充值功能时都可能存在虚假充值金额、充值数量篡改、篡改充值账户等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包

浏览器直接打开构造的请求

3.7.13. 抽奖活动

风险产生点
业务系统存在抽奖活动功能时都可能存在刷取活动奖品、盗刷积分、刷取活动奖品都漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.14. 代金优惠

风险产生点
业务系统存在代金优惠功能时都可能存在批量刷取代金卷/优惠卷、更改代金卷金额、更改优惠卷数量等漏洞，都需要测试
风险等级-高
漏洞危害
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.15. 订单信息

风险产生点
业务系统存在订单功能时都可能存在订单信息泄漏、用户信息泄漏、订单遍历订单请求重放测试（如：大量发送订单，有些订单变为 0）、订单其他参数干扰测试（如：正变负）等漏洞，都需要测试

风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.16. 运费账单

风险产生点
业务系统存在运费账单功能时都可能存在运费篡改、货到付款、免运费（退货，换货，第三方可以陪一定金额运费）、包邮、运费免邮卷等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.7.17. 第三方商家

风险产生点
业务系统存在第三方商家功能时都可能存在盗号、商家账户遍历、越权访问其他商家用户等漏洞，都需要测试
风险等级-高
测试方法
Eclipse 导入源码分析 Find Security Bugs 或 Fortify 扫描源码 Burp Suite 动态修改请求数据包 浏览器直接打开构造的请求

3.8. 攻击行为风险

针对攻击行为风险可能产生的攻击方式安全测试，参考以下要点：

3.8.1.短信验证

风险产生点
存在短信验证机制的业务系统，在生成验证码的长度和时效性未做安全考虑，短信验证码是以 4-6 位数字验证，而且定义的失效时间过长，比如 1 个小时，可能会导致攻击者在失效时间内进行高频测试，暴力验证码，从而绕过短信验证码验证机制。
风险等级-高
测试方法
Eclipse 导入源码分析 Burp Suite 动态修改请求数据包

3.8.2.用户撞库

风险产生点
撞库是黑客通过收集互联网已泄露的用户和密码信息，生成对应的字典表，尝试批量登录其他网站后，得到一系列可以登录的用户名和密码组合。由于很多用户在不同网站使 用的是相同的账号和密码，因此黑客可以通过获取用户在 A 网站的账户从而尝试登录 B 网站，这就可以理解为撞库攻击。（存在风险点的位置：登录处，找回密码等）
风险等级-高
测试方法
Eclipse 导入源码分析 Burp Suite 动态修改请求数据包

3.8.3.爆破密码

风险产生点

暴力破解测试是指针对应用系统用户登录账号与密码进行的穷举测试，针对账号或密 码进行逐一比较，直到找出正确的账号与密码。 一般分为以下三种情况： • 在已知账号的情况下，加载密码字典针对密码进行穷举测试； • 在未知账号的情况下，加载账号字典，并结合密码字典进行穷举测试； • 在未知账号和密码的情况下，利用账号字典和密码字典进行穷举测试。
风险等级-高
测试方法
Eclipse 导入源码分析 Burp Suite 动态修改请求数据包

3.8.4.ID 遍历

风险产生点
用户在请求操作（增、删、查、改）某条数据时，Web 应用程序没有判断该数据的 所属人，或者在判断数据所属人时直接从用户提交的表单参数中获取（如用户 ID），导 致攻击者可以自行修改参数（用户 ID），操作不属于自己的数据，实现 ID 遍历。
风险等级-严重
测试方法
Eclipse 导入源码分析 Burp Suite 动态修改请求数据包

3.8.5.风控缺失

风险产生点
业务系统对用户注册和登录未做基于来源的前置条件验证，可以通过脚本和编程实现批量用户注册和登录，并进行具体转账等操作。
风险等级-严重
测试方法
Eclipse 导入源码分析 Burp Suite 动态修改请求数据包