CMC 产品研发编码规范--禁止性案例库

软件规范

编写	软件质量管理工作组	编写时间	2018. 9
审批		审批时间	
文档版本	V1.0		

文档修订摘要

日期	修订号	描述	著者	审阅者	日期

目录

1. 概述

1.1. 编制说明

为了提升事业部编码工作的规范性，促进产品代码的可读性、可维护性，提升产品的信息安全能力，降低各产品线、部门之间研发沟通成本，因此组织事业部下属各部门技术专家共同制定《移动客户事业部信息安全和代码质量禁止性案例库》。

本规范体系包括 JavaHtml/CSS、JavaScript、配置文件以及数据库（Oracle）SQL 语言。

本册对事业部内产品研发中禁止出现的开发问题进行了详细阐述。

1.2. 适用范围

本规范适用于南京亚信软件 CMC 事业部各产品研发部软件产品开发项目、软件工程项目、需求服务开发项目、系统升级项目。

南京亚信软件 CMC 事业部以下人员遵守本规范：

产品研发中心 Java 代码开发人员、代码质量控制人员、设计工程师、测试工程师。

1.3. 生效时间

发布之日起生效。

1.4. 语解释

本文中使用到的专业术语解释如下：

序号	术语	解释
1	禁止性案例	在产品研发过程中严令禁止出现的代码块示例

1.5. 解释主体

本管理规范由规划设计部起草并负责解释。

2. 信息安全案例

2.1. 敏感数据处理

2.1.1. URL 及业务数据

2.1.1.1. 案例概述

案例名称	严重性	适应性	备注
URL 及业务数据	严重	通用	URL 可以扫描

2.1.1.2. 案例描述

- 案例概述

  部分敏感数据未脱敏，导致部分客户、用户、账户的敏感信息泄露，以及系统被攻击等。

- 详细描述

  URL 及业务数据未脱敏主要表现在：

  1、URL 请求参数中包含敏感信息，未做加密操作

  2、系统中涉及到客户、账户等敏感信息未进行模糊化处理

2.1.1.3. 检查点

1、对 URL 中请求参数中包含敏感信息（主要指客户、用户、账户相关的主体信息，如客户姓名、证件号码、地址、手机号码、用户名、密码等）参数化，配置化，通过配置做加

密操作；

2、URL 请求路径是否为 url 全路径，入 http://xxx/xx/x.jsp;

3、客户、账户、用户信息页面中主要敏感信息事是否特殊字符展示，且在源码中也为特殊字符；

2.1.1.4. 检查方法

1、url 中的敏感信息可以通过安全扫描工具扫描得到；

2、业务系统中的敏感数据是不能通过工具扫描得到的，只能通过：

1）、 代码评审/测试

2）、检测形式多样，工具爬虫扫描得到敏感文件的路径，从而找到敏感数据

3）、手工挖掘，根据 web 容器或者网页源代码的查看，找到敏感信息。

2.1.1.5. 禁止项

- 敏感数据不允许明文传输
- 敏感信息未模糊化

2.1.1.6. 解决方法

1、对登录 URL 中请求参数中包含敏感信息做加密操作

2、实现敏感信息模糊化

3、敏感信息的访问，需要记录相应的日志，以便后续审计、追溯

2.1.2. 密码明文保存

2.1.2.1. 案例概述

案例名称	严重性	适应性	备注
密码明文保存	严重	通用	安全工具无法扫描

2.1.2.2. 案例描述

■ 案例概述

密码信息配置为明文，使得密码信息被盗取的门槛降低，信息泄露和系统安全的大门被打开。

如：在渠道的各地项目中，密码还是明文配置。

■ 详细描述

在配置文件、配置中心，数据库密码之类被配置成了明文，容易造成密码泄露。

2.1.2.3. 检查点

检查点：

检查 JDBC 配置信息。

2.1.2.4. 检查方法

检查方法：

人工检查各配置信息，安全扫描工具无法扫描。

2.1.2.5. 禁止项

禁止配置文件/数据存储明文密码。

2.1.2.6. 解决方法

存储密码等信息前必须调研加密方法。

2.2. XSS 漏洞

2.2.1. 一般性 XSS 漏洞

2.2.1.1. 案例概述

案例名称	严重性	适应性	备注
一般性 XSS 漏洞	严重	通用	安全工具可以扫描

2.2.1.2. 案例描述

- 概述

一般性 XSS 漏洞可以造成应用系统被跨网站攻击，可能会导致系统故障、信息泄露等问题。

- 详细描述

XSS(跨站脚本)漏洞是 web 应用程序的一种常见漏洞，它允许恶意 web 用户将代码植入到提供给其它用户使用的页面中。比如这些代码包括 HTML 代码和客户端脚本。攻击者利用 XSS 漏洞旁路掉访问控制——例如同源策略(same origin policy)。这种类型的漏洞由于被黑客用来编写危害性更大的网络钓鱼(Phishing)攻击而变得广为人知。对于越重要的系统，它的危害越大。

2.2.1.3. 检查点

检查点：

通过跨站资源访问限制来控制第三方对应用资源的跨站引用和访问。通过对请求交互中传输文本的特殊字符过滤来防止可能引起 XSS 攻击的脚本关键字录入到系统中。

2.2.1.4. 检查方法

检查方法：（安全工具扫描，辅助以代码评审，深度测试）

1.对提交保存 -> 查询展现类的功能，在修改的内容中加入常见的 XSS 注入脚本(例如：''等)进行测试、常见的有资料修改类、公告消息类，修改保存后，通过查询展现，检查是否对提交的 XSS 内容做了处理。

2.对不经过服务端请求的、修改内容 -> 前端构造 HTML 并展现的功能，在修改内容中加入常见的 XSS 注入进行测试，检查动态构造上屏的内容是否对 XSS 内容做了处理。

3.使用 WEB 应用程序攻击/渗透工具(Burpsuite)来对交互请求做拦截、并修改请求中的请求参数名、参数值，加入 XSS 注入脚本、检查提交保存后的数据是否做了 XSS 安全处理。

2.2.1.5. 禁止项

禁止输入 HTML 标签等字符：

"eval\\((.*?)\\)"

"EVAL\\((.*?)\\)"

"e-xpression\\((.*?)\\)"

"E-XPRESSION\\((.*?)\\)"

"javascript:"

"JAVASCRIPT:"

"vbscript:"

"VBSCRIPT:"

"alert\\((.*?)\\)"

"ALERT\\((.*?)\\)"

"confirm\\((.*?)\\)"

"CONFIRM\\((.*?)\\)"

"onerror"

"ONERROR"

"onfocus"

"FOCUS"

"prompt"

"PROMPT"

"onmouseover"

"ONMOUSEOVER"

">"　替换为"&gt

"<"　替换为"<"

"alert"

"ALERT"

" and "

"union"

"substr"

2.2.1.6. 解决方法

1、 禁止使用禁止项字符；

2、 对于必须使用的禁止项字符，也必须强制进行转换

2.2.2. XSS 漏洞-COOKIE 属性

2.2.2.1. 案例概述

案例名称	严重性	适应性	备注
XSS 漏洞-cookie 属性	严重	通用	安全工具可以扫描

2.2.2.2. 案例描述

- 概述

  程序设置的会话 cookie 未包含 HttpOnly 属性。

- 详细描述

攻击者可结合 XSS 跨站等漏洞，通过往页面中插入恶意 JS 代码的方式获取用户的会话 cookie，进而冒用用户身份发起进一步攻击。

2.2.2.3. 检查点

检查点：cookie 是否包含 HttpOnly

针对内网，允许有特殊情况申诉。

2.2.2.4. 检查方法

检查方法：（安全工具扫描）

HttpOnly 是否设值，修改程序源代码，为用户会话 cookie 设置 HttpOnly 属性：response.addHeader("Set-Cookie", "uid=112; Path=/; HttpOnly");

电渠检查方法：

2.1、请求时是否调用客户端指纹生成方法：getUUID()

2.2、服务端是否调用了指纹验证方法：checkCrc(HttpServletRequest request)

2.2.2.5. 禁止项

禁止性条款：未包含 HttpOnly

2.2.2.6. 解决方法

电渠解决方案

1、对 cookie 做安全限制，防止第三方读取和窃取。

检查配置项：

<bean id="sessionIdCookie" class="org.apache.shiro.web.servlet.SimpleCookie">

```
    <constructor-arg name="name" value="wap.session.id"/>
    <property name="path"       value="/"></property>
    <property name="httpOnly" value="true"></property>
</bean>
```

2、采用客户端指纹校验：通过指纹算法计算出浏览器的唯一指纹，与服务器中用户生成会话时第一次保存的指纹进行校验，如果指纹不一致，就有可能被劫持。

2.3. 越权访问

2.3.1. 系统越权访问

2.3.1.1. 案例概述

案例名称	严重性	适应性	备注
系统越权访问	严重	通用	安全扫描工具无法扫描

2.3.1.2. 案例描述

- 案例概述

  应用界面未做反向权限校验导致越权访问。

- 详细描述

  系统前台仅对菜单展现做了权限过滤，但在打开具体的应用界面时，没有对界面访问权限做反向校验，导致无访问权限的操作员在知道某个菜单 URL 地址的情况下可以通过浏览器开发者工具或其它手段直接访问应用界面进行越权操作。

2.3.1.3. 检查点

检查点：

人工检查系统公共框架的配置文件 web.xml

2.3.1.4. 检查方法

检查方法：（未采用公共框架使用代码审核，采用公共框架的检查配置文件）

1、代码审查 XML 的配置文件，是否已加过滤器，过滤器要做权限的检查；

2、找到权限控制要求的菜单或页面

3、对于权限框架的，看是否对该菜单或页面有权限配置

4、对没有权限框架的，只能进行代码评审

广研：禁止配置文件没有加载过滤器直接上线

长研：公共类

南研：过滤器校验

渠道、客服：shiro 控制

电渠：shiro 控制到菜单

OSS：secframe

逻辑漏洞扫描器无法扫描到。

2.3.1.5. 禁止项

禁止项：

未授权项（菜单、按钮、接口等）禁止访问。

2.3.1.6. 解决方法

解决方案：

1、对菜单、页面、功能等增加权限校验

2、对菜单、页面、功能等增加反向权限校验

2.3.2. App 接口越权访问

2.3.2.1. 案例概述

案例名称	严重性	适应性	备注
App 接口越权访问	严重	通用	安全扫描工具无法扫描

2.3.2.2. 案例描述

- 案例概述

APP 服务端接口未校验当前用户的登录状态或访问权限，导致越权访问。项目上线前局方安全检查时发现，修正后才准上线。

- 详细描述

系统并没有判断用户登录状态或访问权限，导致越权访问，攻击者通过越权访问可能获得更多有利于入侵的条件，进而进一步的入侵系统。

2.3.2.3. 检查点

检查点：

所有面向移动端应用的 app 项目中，应在服务端当前请求用户的登录状态和访问权限，发现请求不满足登录状态和访问权限检查要求时拒绝访问。

2.3.2.4. 检查方法

检查方法：

代码审查，辅助以接口测试工具（如 postman 等），绕过登录流程直接对接口发起访问验证。

逻辑漏洞扫描器无法扫描到。

2.3.2.5. 禁止项

禁止性条款：

1、 所有面向移动端应用的 app 项目中，禁止服务端提供给 app 的 http 接口服务不对接口做登录状态和访问权限的检查；

2、 未授权项（菜单、按钮、接口等）禁止访问，代码评审，辅助以测试方法止未进行反

向验证。

2.3.2.6. 解决方法

解决方案：

1、对请求做用户身份识别，并核查用户的登录状态和访问权限，不满足接口访问要求的拒绝访问

2.4. 短信攻击

2.4.1. 短信攻击

2.4.1.1. 案例概述

案例名称	严重性	适应性	备注
短信攻击	严重	通用	安全扫描工具无法扫描

2.4.1.2. 案例描述

- 案例概述

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

2.4.1.3. 检查点

检查点：

1、检查是否能抓取发送验证码的数据包

2、检查能否进行重放攻击

3、检查能否收取到大量短信

2.4.1.4. 检查方法

检查方法：（测试方法）

1、 手工找到有关网站注册页面，认证页面，是否具有短信发送页面，如果有，则进行下一步。

2、通过利用 burp 或者其它抓包截断工具，抓取发送验证码的数据包，并且进行重放攻击，查看手机是否在短时间内连续收到 10 条以上短信，如果收到大量短信，则说明存在该漏洞。

3、逻辑漏洞扫描器无法扫描到。

2.4.1.5. 禁止项

禁止项：

同一手机号码不允许连续请求，必须有次数或时间限制策略

2.4.1.6. 解决方法

解决方案：

合理配置后台短信服务器的功能，对于同一手机号码，如发送次数不超过 3-5 次，并且可对发送的时间间隔做限制等。

2.5. 弱口令

2.5.1. 弱口令

2.5.1.1. 案例概述

案例名称	严重性	适应性	备注
弱口令	严重	通用	安全扫描工具可以扫描

2.5.1.2. 案例描述

- 案例概述

  口令强度弱，口令长度短等容易被猜测和破解的口令统一称为弱口令。

- 详细描述

  破解用户名后获取用户口令，可利用此漏洞可在后台对信息进行增删改查。

2.5.1.3. 检查点

检查点：

1、查看应用系统中是否使用了弱口令

2.5.1.4. 检查方法

检查方法：
通过安全扫描工具，辅助以代码审查。

1、设置或修改密码时有规则校验；一般的弱口令扫描器可以扫描到。

2、但口令是否加密或加密传输、密码是否定期更换以及密码修改记录等安全工具无法扫描无法扫描。

2.5.1.5. 禁止项

禁止项：

不允许简单密码策略。

2.5.1.6. 解决方法

解决方案：

1、 在应用、系统中强制密码策略为英文字母大小写、符号、数字结合的高强度密码，或者匹配弱口令字典库，避免使用弱口令；

2、 密码长度至少 8 位字符，密码复杂性要求至少包含以下 4 种类别中的 2 种：大写字母、小写字母、数字、特殊符号

3、 系统应具备对口令强度检测的能力，并对用户进行提示（尽量不要以姓名、电话号码以及出生日期等作为密码或者密码的组成部分），且不允许常见弱口令的配置
4、 应采用加密技术保存密码，不得以明文方式保存或者传输
5、 密码至少定期更换一次。修改密码时，须保留密码修改记录，包含帐号、修改时间、修改原因等，以备审计

2.6. 异常信息泄露

2.6.1. 应用异常信息泄露

2.6.1.1. 案例概述

案例名称	严重性	适应性	备注
应用异常信息泄露	严重	通用	安全扫描工具可以部分扫描

2.6.1.2. 案例描述

- 案例概述

  应用程序未屏蔽执行过程中的错误信息，直接抛出了异常，造成敏感信息泄漏；项目上线前局方安全检查时发现，修正后才准上线。

- 详细描述

  攻击者可从程序的错误信息中获得程序开发框架名称及版本、SQL 语句、SQL 数据库表名、绝对路径等敏感信息。对错误页面没有做统一跳转，对于后端抛出的异常信息没有做封装转换。

2.6.1.3. 检查点

检查点：

1、检测抛出的异常信息中是否包含敏感字符（未封装的数据库或 JAVA 异常信息），页面中不应该显示系统直接抛出的异常

2.6.1.4. 检查方法

检查方法：

安全扫描工具，辅助以测试手段。数据库异常未经封装直接抛出，可以通过安全扫描工具扫出，Java 类的异常无法扫描。

1、检查输入错误 url，返回的页面异常信息是否有封装；

2、页面中输入错误信息，检查异常信息是否有封装展示

2.6.1.5. 禁止项

禁止项：

不允许直接返回带有敏感信息的提示，建议提示编码加统一通用描述

2.6.1.6. 解决方法

解决方案：

1、 对于 tomcat 的中间件下，常用修复方式如下：找到配置文件 web.xml,修改内容如下：

```
<error-page>
   <exception-type>java.lang.Throwable</exception-type>
   <location>/jsp/common/error.jsp</location>
 </error-page>
 <error-page>
   <error-code>500</error-code>
   <location>/jsp/common/500.jsp</location>
 </error-page>
 <error-page>
   <error-code>404</error-code>
   <location>/jsp/common/404.jsp</location>
 </error-page>
 <error-page>
   <error-code>403</error-code>
   <location>/jsp/common/403.jsp</location>
 </error-page>
```

2、对于常用的 jsp 语言开发的网站，可在业务流程中，加入异常捕获过程中预定义的错误编码，将异常输出到错误日志中，并在前台页面返回相应的错误编码，以便应用系统运维人员进行异常排查。代码参考：

```
try {
 //某业务处理流程
        ……
} catch (Exception e) {
 e.printStackTrace();
 logger.error(e.getMessage());
 resultMessage = getText("业务处理发生异常，错误编码 A-04221!");
 return "errorJsp";
}
```

appframe，可以配置异常输出内容
wade，通过公共类指定异常编码

2.6.2. Web 容器异常信息泄露

2.6.2.1. 案例概述

案例名称	严重性	适应性	备注
Web 容器异常信息泄露	严重	通用	安全扫描工具可以扫描

2.6.2.2. 案例描述

- 案例概述

  web 容器开启浏览目录容易导致系统信息泄露。如：Nginx 禁止 autoindex，多个省份安全扫描时要求禁止 autoindex。

- 详细描述

  Nginx 允许列出整个目录的。Autoindex 功能如果不关闭， 整个文件目录都被列出。

2.6.2.3. 检查点

检查点：

使用了 Nginx、Tomcat、Apache 的应用

2.6.2.4. 检查方法

可以通过安全扫描工具扫出。

2.6.2.5. 禁止项

禁止项：

禁止开启浏览目录，检查配置文件

2.6.2.6. 解决方法

解决方案：

如：关闭 autoindex 功能：在 nginx 的配置文件的 server 或 location 段里添加上 autoindex off;来关闭目录浏览，下面会分情况进行说明。（1）关闭整个虚拟主机目录浏览

在 server 段添加

location / {

autoindex off;

autoindex_localtime off;

}

（2）关闭单个目录的虚拟主机目录浏览

在 server 段添加

location /ECOP/ {

autoindex off;

autoindex_localtime off;

}

2.6.3. 存在 apache tomcat 的测试文件

2.6.3.1. 案例概述

案例名称	严重性	适应性	备注
存在 apache tomcat	严重	通用	安全扫描工具可以扫描

的测试文件			

2.6.3.2. 案例描述

- 案例概述

apache tomcat 安装后留下的测试文件，这些文件可被攻击者恶意利用构造管理员 session，泄露敏感信息。

2.6.3.3. 检查点

检查点：

tomcat 安装后的测试文件

2.6.3.4. 检查方法

检查方法：

访问 tomcat 主 URL 是否出现 tomcat 管理界面(http://localhost:*)或者服务器 webapps 下是否存在 examples

可以通过安全扫描工具扫出。

2.6.3.5. 禁止项

禁止项：

禁止存在 examples 目录

2.6.3.6. 解决方法

解决方案：

删除 examples 目录。

2.6.4. nginx 版本信息泄露

2.6.4.1. 案例概述

案例名称	严重性	适应性	备注
nginx 版本信息泄露	严重	通用	安全扫描工具可以扫描

2.6.4.2. 案例描述

- 案例概述

  对外部域机器进行了安全扫描，发现能开的 nginx 会暴露版本信息 nginx 版本信息泄露。

2.6.4.3. 检查点

检查点：

安装 Nginx 的服务器

2.6.4.4. 检查方法

检查方法：

在 Nginx 编译前修改相关源码,进行版本信息屏蔽，如，访问错误的 url http://**/a.jsp

可以通过安全扫描工具扫出。

2.6.4.5. 禁止项

禁止项：

禁止性条款:显示的 Nginx 版本号，（修改配置，输入错误 url 检查）

2.6.4.6. 解决方法

解决方案：

修改配置。

2.6.5. Apache 版本信息泄露

2.6.5.1. 案例概述

案例名称	严重性	适应性	备注
Apache 版本信息泄露	严重	通用	安全扫描工具可以扫描

2.6.5.2. 案例描述

- 案例概述

  上线后，经安全检查发现 tomcat 版本信息泄露,引用 tomcat 问题地址后 apache 版本信息泄露。

- 详细描述

  通过此漏洞可在用户服务器上执行任意代码，从而导致数据泄露或获取服务器权限，存在高安全风险。

2.6.5.3. 检查点

检查点：是否关闭 tomcat 版本漏洞

2.6.5.4. 检查方法

检查方法：

设置统一的报错页面 ，即在 web.xml 设置 404 配置；屏蔽中间件版本号；访问一个不存在地址,比如：http://localhost:8080/测试.jsp。

安全扫描工具可以扫描。

2.6.5.5. 禁止项

禁止项：

禁止显示的 Nginx 版本号，（修改配置，输入错误 url 检禁止性条款:禁止显示版本号

2.6.5.6. 解决方法

解决方案：

修改配置。

2.7. 上传下载文件校验

2.7.1. 任意文件上传

2.7.1.1. 案例概述

案例名称	严重性	适应性	备注
任意文件上传	严重	通用	安全扫描工具无法扫描

2.7.1.2. 案例描述

- 案例概述

  文件上传逻辑,没有对文件格式做限制，导致可以上传恶意脚本或恶意可执行文件。

- 详细描述

  上传文件功能没有针对文件真实的内容扫描匹配，导致可执行文件或脚本修改一下扩展名就可以上传了。

2.7.1.3. 检查点

检查点：

针对文件头做文件上传限制,不仅仅只判断文件扩展名。

2.7.1.4. 检查方法

检查方法：（测试方法）

验证将 exe 改成 txt 的后缀上传，以及其它后缀，主要是针对 dll，exe 格式的可执行文件

安全扫描工具无法扫描。

2.7.1.5. 禁止项

禁止项：

1、 禁止 exe、dll、com 类文件上传

2、 未调用文件合法性校验的方法（代码审查）

2.7.1.6. 解决方法

解决方案：

1、 禁止 exe、dll、com 类文件上传

2、 调用文件合法性校验的方法

2.8. APP 源码混淆加密

2.8.1. APP 源码混淆加密

2.8.1.1. 案例概述

案例名称	严重性	适应性	备注
APP 源码混淆加密	严重	通用	安全扫描工具无法扫描

2.8.1.2. 案例描述

- 案例概述

  APP 的源代码未混淆加密，通过逆向工程可以直接看到 APP 的主要源代码；项目上线前局方安全检查时发现，修正后才准上线

- 详细描述

  攻击者通过逆向反汇编等技术可以直接看到 APP 的代码结构和主要源代码，进而对 APP 源代码进行分析，获得 APP 的敏感信息加密算法，并进行后门植入、广告插入等攻击。

2.8.1.3. 检查点

检查点：

所有面向移动端应用的 app 项目中，检查项目源码是否已加密处理

2.8.1.4. 检查方法

检查方法：

可通过逆向反汇编等技术（如 jd 反编译工具）检查 app 源码是否混淆加密。

安全扫描工具无法扫描。

2.8.1.5. 禁止项

禁止项：

所有面向移动端应用的 app 项目中，禁止对项目源码在不进行加固或加密操作前直接将应用发布到互联网应用平台。

2.8.1.6. 解决方法

解决方案：

应使用如 360 加固等第三方工具，进行混淆加密 APP 源代码。

2.9. 账号枚举

2.9.1. 账号枚举

2.9.1.1. 案例概述

案例名称	严重性	适应性	备注
账号枚举	严重	通用	安全扫描工具无法扫描

2.9.1.2. 案例描述

- 案例概述

  应用程序在用户名/密码验证错误时会分别输出不同的错误信息，攻击者可能利用此缺陷进行暴力猜解攻击，局方安全组在上线前就通过 sonar 工具扫描出来该问题，修改后才准上线

- 详细描述

  应用程序在用户名/密码验证错误时会分别输出不同的错误信息，攻击者可能利用此缺陷进行暴力猜解攻击。

2.9.1.3. 检查点

检查点：各种类型的登录页面或接口；

2.9.1.4. 检查方法

检查方法：

分别输入错误的用户名和密码，如果输出不同的错误，则需要进行整改；

安全扫描工具无法扫描。

2.9.1.5. 禁止项

禁止项：

1、禁止详细提示登录信息，模糊化提示登录信息提示，如用户或密码错误，统一提示“用户或密码错误”，减少被攻击的可能；

2、不允许同一时段同一地址频繁请求（代码审查辅助测试）。

2.9.1.6. 解决方法

解决方案：

登录提示信息模糊化处理。

2.10. 并发控制

2.10.1. 并发控制

2.10.1.1. 案例概述

案例名称	严重性	适应性	备注
并发控制	严重	电渠产品线	安全扫描工具无法扫描

2.10.1.2. 案例描述

■ 案例概述

没有对并发情况控制，导致多条记录生成。

2.10.1.3. 检查点

检查点：

1、 并发活动业务；

2、 与金钱相关业务；

3、 库存增减业务。

2.10.1.4. 检查方法

检查方法：

代码评审和测试。

分别输入错误的用户名和密码，如果输出不同的错误，则需要进行整改。

安全扫描工具无法扫描。

2.10.1.5. 禁止项

禁止项：

1、禁止详细提示登录信息，模糊化提示登录信息提示，如用户或密码错误，统一提示“用户或密码错误”，减少被攻击的可能；

2、不允许同一时段同一地址频繁请求。

2.10.1.6. 解决方法

解决方案：

1、 使用分布式全局锁对红包加锁，实现并发控制。

2、 检查在活动业务、库存扣减业务、与金钱有关的高并发业务场景中是否先调用加锁方法

redis.getLock()加锁成功后再进行业务操作

2.11. 敏感词过滤

2.11.1. 敏感词过滤

2.11.1.1. 案例概述

案例名称	严重性	适应性	备注
敏感词过滤	严重	电渠产品线	安全扫描工具无法扫描

2.11.1.2. 案例描述

- 案例概述

  加入关键字过滤，导致对下发内容不可控。任何进去数据库的非法、煽动、谣言短信均存在下发给用户的可能多个省份要求对敏感词进行过滤

2.11.1.3. 检查点

检查点：

1、 短信下发；

2、 微信下发。

2.11.1.4. 检查方法

检查方法：

测试提交敏感字和下发短信场景，查看是否能控制敏感字，(由于敏感字库一般有第三方提供，具体测试的使用由第三方提供）

安全扫描工具无法扫描。

2.11.1.5. 禁止项

禁止项：

1、禁止条款：禁止不校验内容就下发短信、微信（代码审查辅助以测试）。

2.11.1.6. 解决方法

解决方案：

1.目前敏感字库一般由第三方提供检查是否调用第三方提供的敏感字校验方法（第三方接口，由实际项目的敏感字负责方提供）

2.下发程序在进行下发模版或者内容配置时，调用敏感词库中的方法进行验证。判断其是否合法

2.12. 数据加密与验签

2.12.1. 数据加密与验签

2.12.1.1. 案例概述

案例名称	严重性	适应性	备注
数据加密与验签	严重	电渠产品线	安全扫描工具无法扫描

2.12.1.2. 案例描述

- 案例概述

  通过浏览器对访问报文进行截获，多个省份安全扫描时增暴露出该问题。

- 详细描述

  浏览者通过对请求报文截获后，进行分析，拼装成需要的结构，或者对请求串修改模拟请求。

2.12.1.3. 检查点

检查点：

1、 客户端、服务器端是否有加解密操作；

2.12.1.4. 检查方法

检查方法：（代码审查）

1.1、服务端是否配置了统一拦截器进行统一拦截：

<filter-name>securiteFilter</filter-name>
<filter-class>com.ai.ecs.websecurite.filter.DesSecuriteFilter</filter-class>
<filter-mapping>
<filter-name>securiteFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>

1.2、服务调是否调用了：checkSingnal 方法对数据进行验签

1.3、服务端是否调用了：getDesString 对请求的参数进行解密

1.4 、 服 务 端 是 否 调 用 了 正 则 ^[a-z0-9A-Z 一 - 龥 豈 - 鶴 \|\ \ （ \ ） \ 【 \ 】 \{\}_\?\\-\&\$\#\=\(\)\!\@\%\^*\{\}\[\]\!\~\:\：\`\;\；\'\,\，\.\。\<\>\?]+$

对特殊字符进行验证

1.5、是否有错误码配置：

web.xml 中配置项<init-param>

<param-name>httpcode</param-name>
<param-value>403</param-value>
</init-param>

安全扫描工具无法扫描。

2.12.1.5. 禁止项

禁止项：

1、禁止条款：禁止报文未加密传输。

2.12.1.6. 解决方法

解决方案：

1、对 ajax 数据进行加密，服务端进行解密加密的秘钥每次动态生成.防止使用者猜出秘钥后进行加密模拟访问。

检查客户端 ajax 请求时是否调用了：securiteAjax 方法对请求数据进行加密、加签处理。

2、客户端对数据进行加密后，对整体的参数进行一个 MD5 摘要。将摘要信息放入到 http 头中。服务端接收到请求后对数据进行整体验签，防止请求串被篡改。电渠的实现方式是：服务端重写 ServletHttpRequest 对象与过滤器，统一在过滤器中对数据进行验签和解密操作。

2.13. http 头攻击

2.13.1. http 头攻击

2.13.1.1. 案例概述

案例名称	严重性	适应性	备注
http 头攻击	严重	电渠产品线	安全扫描工具可以扫描

2.13.1.2. 案例描述

- 案例概述

  通过设置 Httpt 头中的 host 导致重定向挂马网站，多个省份安全扫描时增暴露出该问题。

2.13.1.3. 检查点

检查点：

1、 检查外网业务

2、 对内网业务（CRM、客服、渠道、BOMC、OSS）不校验

2.13.1.4. 检查方法

检查方法：（代码审查）

1.1、检查是否有调用获取头方法：

```
public String getHeader(String name) {
        return StringEscapeUtils.escapeHtml4(super.getHeader(name));
    }
```

1.2、检查是否有调用 host 合法性验证方法：checkHost(String host)

1.3、测试将 http 请求的头修改为其他网站域名或者含有脚本的数据，服务端是否会返回 403

安全扫描工具可以扫描。

2.13.1.5. 禁止项

禁止项：

禁止业务后台不对 host 值进行校验。

2.13.1.6. 解决方法

解决方案：

1、后台统一对 host 值进行处理或者校验。

2.14. CSRF 攻击

2.14.1. CSRF 攻击简单预防

2.14.1.1. 案例概述

案例名称	严重性	适应性	备注
CSRF 攻击简单预防	严重	电渠产品线	安全扫描工具可以扫描

2.14.1.2. 案例描述

- 案例概述

跨站请求伪造（CSRF）攻击里面，攻击者通过用户的浏览器来注入额外的网络请求，来破坏一个网站会话的完整性。。而浏览器的安全策略是允许当前页面发送到任何地址的请求，因此也就意味着当用户在浏览他/她无法控制的资源时，攻击者可以控制页面的内容来控制浏览器发送它精心构造的请求。

2.14.1.3. 检查点

检查点：

1、 对所有请求过来的 http 请求。

2、 不适用内网应用

2.14.1.4. 检查方法

检查方法：

安全扫描工具可以扫描。

2.14.1.5. 禁止项

禁止项：

禁止不对站点进行校验（检查服务器端是否有该校验）

2.14.1.6. 解决方法

解决方案：

1.对所有请求过来的 http 请求进行 referer 校验，只有授信的站点才能才能容许请求，未授信直接返回 403

1.1、检查代码是否有校验 referer 的代码

```
if(refer.contains(domains)){
    if(logger.isDebugEnabled()){
        logger.debug("refer  校验通过");
    }
    doRealFilter(req,reps,chain);
}
```

1.2、检查是否配置了授信的域名库：

配置文件 web.xml:

```
<init-param>
  <param-name>domains</param-name>
  <param-value>127.0.0.1,wap.10086.cn</param-value>
</init-param>
```

2.给每次请求头中加入 token，用户请求过来后对 token 进行验证，是否是合法。Token 保存在 cookie 中并且设置了失效性。

检查代码是否调用校验 token 是否合法的方法：

```
HttpSession session = request.getSession(true);
String laserandom =(String)session.getAttribute(FILE_RANDOM);
if(!StringUtils.isBlank(laserandom)){
    if(laserandom.equals(random)) return true;#random 是每次前台传来的
}
```

2.15. 合法性校验

2.15.1. 业务合法性校验

2.15.1.1. 案例概述

案例名称	严重性	适应性	备注

业务合法性校验	严重	电渠产品线	安全扫描工具无法扫描

2.15.1.2. 案例描述

- 案例概述

  微信号和手机号没有做双重校验，导致搞活动微信时曾出现过只校验微信号导致校验不严的问题。

- 详细描述

  对绑定、充值送业务进行微信号与手机号双重校验，防止一个手机号注册多个微信号重复参与活动。

2.15.1.3. 检查点

检查点：

业务合法性校验。

2.15.1.4. 检查方法

检查方法：

代码审查。

2.15.1.5. 禁止项

禁止项：

禁止在微信绑定业务、充值送等活动中不校验手机号。

2.15.1.6. 解决方法

解决方案：

1、业务加强逻辑校验，同时校验微信号与手机号

1.1、审查微信绑定业务逻辑是否调用微信号与手机号双重校验方法

1.2、审查充值送业务逻辑是否调用微信号与手机号双重校验方法

2.16. 微服务鉴权

2.16.1. 微服务鉴权

2.16.1.1. 案例概述

案例名称	严重性	适应性	备注
微服务鉴权	严重	COMS 产品线	安全扫描工具无法扫描

2.16.1.2. 案例描述

- 案例概述

  微服务未鉴权。未发生故障，在代码 review 中发现。

- 详细描述

  微服务提供给外部调用时，未鉴权，导致外部可以随意调用。

2.16.1.3. 检查点

检查点：

不登录系统，直接使用 postman 工具访问服务接口，检查结果是否为不允许访问。

2.16.1.4. 检查方法

检查方法：（测试方法）

在 postman 等工具中，向服务器端发送正常请求

检查返回结果是否不允许访问

2.16.1.5. 禁止项

禁止项：

禁止无权限用户访问服务接口。

2.16.1.6. 解决方法

解决方案：

1、禁止无权限用户访问服务接口。

2.17. 请求篡改

2.17.1. 请求篡改

2.17.1.1. 案例概述

案例名称	严重性	适应性	备注
请求篡改	严重	CRM 长研	安全扫描工具可以扫描

2.17.1.2. 案例描述

- 案例概述

通过浏览器插件篡改请求，模拟调用。

2.17.1.3. 检查点

检查点：

云盾浏览器，定制 IE 和 Chrome 内核，限制浏览器插件，对请求加密。

2.17.1.4.检查方法

检查方法：（测试方法）

在 postman 等工具中，向服务器端发送正常请求

检查返回结果是否不允许访问

使用安全扫描工具扫描。

2.17.1.5. 禁止项

禁止性条款：

实施的省份，禁止使用其他外部浏览器，只能使用云盾。

2.17.1.6. 解决方法

解决方案：

使用云盾浏览器。

2.18. 操作系统权限控制

2.18.1. 操作系统权限控制

2.18.1.1.案例概述

案例名称	严重性	适应性	备注
操作系统权限控制	严重	CRM 长研	安全扫描工具无法扫描

2.18.1.2.案例描述

- 案例概述

  针对类 Unix 系统进行多层的操作权限控制，避免应用程序不可控导致主机故障

- 详细描述

针对操作系统的账号、运行级别、远程操作等设定对应的安全控制，避免运维误操作以及应用程序非法操作。

2.18.1.3. 检查点

检查点：

1、检查 root 下是否有 java 、数据库进程；

主机命令行下执行 "top | grep root"，查看是否有 root 启动的进程信息，检查最右侧的 "systemd" 列，是否有应用启动句柄(java、mysqld、oracle、redis、memcache)

2、检查操作系统运行级别

主机命令行下执行 "runlevel"， 查看主机运行级别，不能是 5

3、检查主机是否启用 telnet 服务，检查非文件服务器主机是否启用了 ftp 服务；

1）telnet 服务检查，主机命令行下使用 nc 命令探测本机端口，执行 "nc -v -w 5 本机 IP 23" 探测 Telnet 服务端口是否启用，启用时会返回"Connected" 字样

2) FTP 服务检查，主机命令行下使用 nc 命令探测本机端口，执行 "nc -v -w 5 本机 IP 21" 探测 FTP 服务端口是否启用，启用时会返回"Connected" 字样

2.18.1.4.检查方法

检查方法：

人工检查。

安全扫描工具无法扫描。

2.18.1.5. 禁止项

禁止性条款：

1.禁止在 root 用户下运行业务应用

2.禁止系统运行在图形界面（level 5 级别下

3.禁止启用 telnet 服务，禁止非文件服务器启用 ftp 服务

2.18.1.6. 解决方法

解决方案：

加强系统权限管理。

2.19. IP 白名单限制

2.19.1. IP 白名单限制

2.19.1.1.案例概述

案例名称	严重性	适应性	备注
IP 白名单限制	严重	CRM 长研	安全扫描工具无法扫描

2.19.1.2.案例描述

- 案例概述

  针对 crm 的所有中台组件（app、memcache、redis、Kafka、Zookeeper、MySQL、监控平台等）设置允许访问的 IP，以增强其安全性

- 详细描述

  在 tcp 的网络层控制，限制接入 IP，能有效避免内部人员的非法操作，并在集团安全扫描时提高门槛。

2.19.1.3. 检查点

检查点：

修改组件的网络层逻辑，统一配置文件 acl.conf，统一配置格式。

2.19.1.4.检查方法

检查方法：

使用测试方法进行检查。

在必须访问这些中台服务之外的主机上，使用这些中台服务的客户端或者相应的协议对中台服务发起连接或探测，检查是否能连接上服务或者获取到服务端信息。

安全扫描工具无法扫描。

2.19.1.5. 禁止项

禁止性条款：

只允许必须且合法的 IP 来源对这些中台服务进行访问。禁止非法的访问和探测。（有相应的校验方法，代码审核）

2.19.1.6. 解决方法

解决方案：

1、 统一配置文件 acl.conf，统一配置格式。

2、 加强日常检查。

2.20. 文件泄漏

2.20.1. 实名制以及实名制生成的文件泄漏

2.20.1.1. 案例概述

案例名称	严重性	适应性	备注
实名制以及实名制生成的文件泄漏	严重	CRM 长研	安全扫描工具无法扫描

2.20.1.2. 案例描述

- 案例概述

  实名业务经常会涉及证件照片、签名文件、头像文件等，在传输过程中绕开权限控制，容易被截取。

- 详细描述

  1、处理图片文件采用 Base64 时未对内容加密；

2、文件存放在 FTP 服务器上未加密存放；

3、提供下载的资源为提升性能未做访问限制，未登录用户可通过链接直接下载，导致信息泄漏。

2.20.1.3. 检查点

检查点：
1.检查敏感图片、文件信息是否加密存储
2.检查敏感图片、文件信息是否能被越权访问。

2.20.1.4.检查方法

检查方法：

1.直接从文件服务器上下载涉敏的文件、图片文档，检查是否能直接查看内容（能直接查看则未加密）。

2.先使用合规的账号登录系统，打开查看涉敏文档，通过抓包工具得到相应的请求 URL 路径。然后在未登录或登录不合规的账号两种情况下，是否能通过文档访问的 URL 获得文档内容。（参见越权访问案例）

安全扫描工具无法扫描。

2.20.1.5. 禁止项

禁止性条款：

1.对于敏感的文件、图片文档，需要对其进行加密传输和存储

2.对于敏感类的文件、图片文档，需要对其访问进行权限的合规控制。

2.20.1.6. 解决方法

解决方案：

1、 对涉敏文档进行加密

2.21. 流量控制访问

2.21.1. 流量控制访问

2.21.1.1. 案例概述

案例名称	严重性	适应性	备注
流量控制访问	严重	OSS	安全扫描工具无法扫描

2.21.1.2. 案例描述

- 案例概述

出现外部渠道大量访问用户信息，并将用户隐私信息进行交易 能力开放平台针对外部渠道大并发接入进行流量控制，避免外部渠道恶意访问能力开放平台。

2.21.1.3. 检查点

检查点：

三户资料访问量查询.。将最近 1 个月中最大值的 130%设为渠道访问额度，通过 redis 提供的计数器记录渠道访问量。

2.21.1.4. 检查方法

检查方法：

代码审查，检查是否有访问量统计及限制方法。

安全扫描工具无法扫描。

2.21.1.5. 禁止项

禁止性条款：

超过最高值 30%，break 程序跳出，终止服务

。

2.21.1.6. 解决方法

解决方案：

统计生产上每个渠道每天三户资料查询接口，用户套餐查询，宽带信息查询，用户信息查询等关键业务数据，将最近 1 个月中最大值的 130%设为渠道访问额度，通过 redis 提供的计数器记录渠道访问量，当渠道使用量超过此额度时能力开放平台发送告警短信给运维人员，超过额度后继续上升到额度的 50%时系统禁止外围渠道使用此业务。

2.22. 超长登录

2.22.1. 超长登录

2.22.1.1. 案例概述

案例名称	严重性	适应性	备注
超长登录	严重	OSS	安全扫描工具可以扫描

2.22.1.2. 案例描述

- 案例概述

超时时间过长的应用系统很可能被攻击者捕获到敏感信息，进而进行进一步的利用。

2.22.1.3. 检查点

检查点：

session-timeout

2.22.1.4. 检查方法

检查方法：

安全扫描工具可以扫描。

1：通过 Sonar 扫描，查看 session-timeout 是否大于 12 个小时

2：手工挖掘，根据 web 容器查看 session-timeout 时间

2.22.1.5. 禁止项

禁止性条款：

禁止 session-timeout 超过 12 个小时。

2.22.1.6. 解决方法

解决方案：

session-timeout<12 小时。

3. 代码质量案例

3.1. JAVA 基础

3.1.1. 写死常量问题

3.1.1.1. 案例概述

案例名称	严重性	适应性	备注
写死常量问题	中等	通用	

3.1.1.2. 案例描述

将应该使用业务常量的代码写死成了固定字符串，导致后续配置失效。

3.1.1.3. 错误代码示例

```
if (4 == Integer.parseInt(acceptType)){

    queryCustAttachAccessModeRequest.setBusiType(CtrlConstants.BusinessType.DATAINFO);
                        }
else if (5 == Integer.parseInt(acceptType)){
```

```
        queryCustAttachAccessModeRequest.setBusiType(CtrlConstants.BusinessType.VOICEINFO);
                                }
else if (6 == Integer.parseInt(acceptType)){

        queryCustAttachAccessModeRequest.setBusiType(CtrlConstants.BusinessType.MOBILEINFO);
                        }
```

3.1.1.4. 检查点

所有使用业务常量进行比较判定的代码。

3.1.1.5. 检查方法

SONAR 支持：
Magic numbers should not be used.　　squid:S109

3.1.1.6. 禁止项

禁止在使用业务常量进行比较时写死固定值。

3.1.1.7. 解决方法

定义并使用业务常量。

3.1.2. 对象比较判定问题

3.1.2.1. 案例概述

案例名称	严重性	适应性	备注
对象比较判定问题	中等	通用	

3.1.2.2. 案例描述

使用==或！=操作符进行对象是否相同的判定。

3.1.2.3. 错误代码示例

```
Object obj1=new Object();
Object obj2=new Object();
if(obj1==obj2){
    ……
}
```

3.1.2.4. 检查点

对象判定的代码。

3.1.2.5. 检查方法

SONAR 支持：
"==" and "!=" should not be used when "equals" is overridden. squid:S1698

3.1.2.6. 禁止项

禁止在对象比较的时候使用==或！=操作符。

3.1.2.7. 解决方法

```
Object obj1=new Object();
Object obj2=new Object();
if(obj1.equals(obj2)){
    ……
}
```

3.1.3. 字符串或常量与变量比较问题

3.1.3.1. 案例概述

案例名称	严重性	适应性	备注
字符串或常量与变量比较问题	中等	通用	

3.1.3.2. 案例描述

在变量与字符串或常量进行比较时没有进行对象为空的判定。
在字符串或常量与变量比较时，将变量放在 equals 前面，字符串或常量放在 equals 后面，造成空指针错误。

3.1.3.3. 错误代码示例

```
if (result.equals("xxxx")){
        syncStatus = TransatelConstant.SyncFromOut.status.INVALID_STATUS;
        logsStatus = "1";
}
```

3.1.3.4. 检查点

变量对象与字符串或常量进行比较的代码。

3.1.3.5. 检查方法

SONAR 支持：Null pointers should not be dereferenced. squid:S2259

3.1.3.6. 禁止项

变量对象与字符串或常量进行比较的时候，禁止不进行对象为空的判定，在比较时，变量对象应该在 equal 后面。

3.1.3.7. 解决方法

```
if （result ! = null）{
  if ("xxxx".equals(result)){
      syncStatus = TransatelConstant.SyncFromOut.status.INVALID_STATUS;
      logsStatus = "1";
  }
}
```

3.1.4. 循环语句中的字符串拼接问题

3.1.4.1. 案例概述

案例名称	严重性	适应性	备注
循环语句中的字符串拼接问题	严重	通用	

3.1.4.2. 案例描述

在循环语句中进行字符串拼接时，没有使用 StringBuilder 对象，直接使用字符串赋值，造成内存溢出。

3.1.4.3. 错误代码示例

```
String s = "";
for (Person p : persons) {
      s += ", " + p.getName();
}
```

3.1.4.4. 检查点

查询所有在 loop 语句中进行字符串拼接的代码。

3.1.4.5. 检查方法

SONAR 支持：
Performance - Method concatenates strings using + in a loop
findbugs:SBSC_USE_STRINGBUFFER_CONCATENATION
Inefficient String Buffering：pmd:InefficientStringBuffering

3.1.4.6. 禁止项

禁止在循环迭代中使用+进行字符串拼接

3.1.4.7. 解决方法

```
StringBuilder sb = new StringBuilder(persons.size() * 16);
for (Person p : persons) {
    if (sb.length() > 0) sb.append(", ");
    sb.append(p.getName);
}
```

3.1.5. switch 语句缺少跳出语句问题

3.1.5.1. 案例概述

案例名称	严重性	适应性	备注
switch 语句缺少跳出语句。	严重	通用	

3.1.5.2. 案例描述

switch 语句中的 case 缺少跳出语句，造成多个分支被执行。每个 case 都需要 break、return、throw 或 continue 等跳出语句，而且即使我们相信代码已经覆盖掉了所有的可能分支，也应当编写 default 语句。

3.1.5.3. 错误代码示例

```
switch(变量){
case  变量值 1:
    //;
case  变量值 2:
    //...;
case default:
    //...;
}
```

3.1.5.4. 检查点

所有使用 switch 语句的代码。

3.1.5.5. 检查方法

SONAR 支持：

"switch" statements should end with a "default" clause：squid:SwitchLastCaseIsDefaultCheck

Switch cases should end with an unconditional "break" statement.　　squid:S128

3.1.5.6. 禁止项

禁止 case 分支没有 break 退出语句。

3.1.5.7. 解决方法

```
switch(变量){
case  变量值 1:
      //;
      break;
case  变量值 2:
      //...;
      break;
   ...
case default:
      //...;
      break;
}
```

3.1.6. finally 代码块包含了跳转语句问题

3.1.6.1. 案例概述

案例名称	严重性	适应性	备注
finally 代码块包含了跳转语句	严重	通用	

3.1.6.2. 案例描述

finally 代码块包含了跳转语句，如 return，程序接着执行 finally 块，1.没有异常，返回 finally 块中的 return 结果，2.有异常，执行 catch 块中位于“return”之前的代码，紧跟着执行 finally 块，覆盖了前边的 return，所以并不会返回异常，同理也适用于 continue 和 break。即如果有 finally 的情况下，try 中的 break 或 continue 都不会立即执行，程序会紧跟着把 finally 中的语句执行完。

3.1.6.3. 错误代码示例

```
package test;
public class TryTest {
    public static void main(String[] args) {
        try {
            System.out.println(TryTest.test());// 返回结果为 true 其没有任何异常
        } catch (Exception e) {
            System.out.println("Exception from main");
            e.printStackTrace();
        }
    }
    public static boolean test() throws Exception {
        try {
            throw new Exception("Something error");// 1.抛出异常
        } catch (Exception e) {// 2.捕获的异常匹配(声明类或其父类)，进入控制块
            System.out.println("Exception from e");// 3.打印
            return false;// 4. return 前控制转移到 finally 块,执行完后再返回
        } finally {
            return true; // 5. 控制转移，直接返回，吃掉了异常
        }
    }
}
```

3.1.6.4. 检查点

所有 finally 语句代码。

3.1.6.5. 检查方法

SONAR 支持："return" statements should not occur in "finally" blocks. squid:S1143

3.1.6.6. 禁止项

禁止 finally 里有跳转语句（break,continue,return,throw,goto）。

3.1.6.7. 解决方法

```
package test;
public class TryTest {
```

```
    public static void main(String[] args) {
        try {
            System.out.println(TryTest.test());// 返回结果为 true 其没有任何异常
        } catch (Exception e) {
            System.out.println("Exception from main");
            e.printStackTrace();
        }
    }
    public static boolean test() throws Exception {
        try {
            throw new Exception("Something error");// 1.抛出异常
        } catch (Exception e) {// 2.捕获的异常匹配(声明类或其父类），进入控制块
            System.out.println("Exception from e");// 3.打印
            return false;// 4. return 前控制转移到 finally 块,执行完后再返回
        } finally {
            //return true;
          .....
        }
    }
}
```

3.1.7.精度计算时采用 BigDecimal 类型进行计算问题

3.1.7.1. 案例概述

案例名称	严重性	适应性	备注
精度计算时采用 BigDecimal 类型进	中等	通用	

行计算问题			

3.1.7.2. 案例描述

在对费用等进行精度计算时应采用 BigDecimal 类型进行计算。

3.1.7.3. 错误代码示例

```
float b1= Float.parseFloat(v1)
float b2 =Float.parseFloat(v2);
return b1+b2;
```

3.1.7.4. 检查点

进行精度计算的代码块。

3.1.7.5. 检查方法

SONAR 支持：

"BigDecimal(double)" should not be used： squid:S2111

Math should not be performed on floats： squid:S2164

3.1.7.6. 禁止项

禁止在精度计算时使用非 BigDecimal 类型。

3.1.7.7. 解决方法

```
double d = 1.1;
BigDecimal bd1 = BigDecimal.valueOf(d);
BigDecimal bd2 = BigDecimal.valueOf(1.1);
return b1.add(b2).doubleValue();
```

3.1.8. 在 FOR 循环中对 LIST 循环变量进行 REMOVE 或 ADD 操作问题

3.1.8.1. 案例概述

案例名称	严重性	适应性	备注
在 FOR 循环中对 LIST 循环变量进行 REMOVE 或 ADD 操作问题	中等	通用	

3.1.8.2. 案例描述

在循环变量中对 LIST 对象进行 REMOVE 或 ADD 操作，会导致数组长度变更，以致业务逻辑出错。

3.1.8.3. 错误代码示例

```
//元素删除后 list.size()会变
IDataset list=new DatasetList();
for(int i=0;i<list.size();i++)
{
	list.remove(i);
}
```

3.1.8.4. 检查点

所有 FOR 循环语句。

3.1.8.5. 检查方法

SONAR 支持：规则待开发。

3.1.8.6. 禁止项

禁止在 FOR 循环中对 LIST 循环变量进行 REMOVE 或 ADD 操作。

3.1.8.7. 解决方法

```
for (Iterator<String> ite = list.iterator(); ite.hasNext();) {
        String str = ite.next();
        System.out.println(str);
        if (str.contains("b")) {
            ite.remove();
        }
}
```

3.1.9. 使用 Map 作为接口出入参问题

3.1.9.1. 案例概述

案例名称	严重性	适应性	备注
使用 Map 作为接口出入参问题	中等	渠道	

3.1.9.2. 案例描述

大量开发人员在开发服务接口时，使用 Map 作为出参和入参，导致接口自测、运维、后续开发非常困难。

3.1.9.3. 错误代码示例

Private static final Logger logger = LoggerFactory.getLogger(EnquiryApproveController.class);

@ResponseBody

@RequestMapping(value = "/submitSales",method = RequestMethod.POST)

@ApiOperation(value = "审批提交", httpMethod = "POST")

public Response<SaveSalesVO> submitSales(@RequestBody Map reqMap, Principal principal)

3.1.9.4. 检查点

所有接口方法的声明代码。

3.1.9.5. 检查方法

SONAR 支持：规则待开发。

3.1.9.6. 禁止项

禁止在接口定义时使用 Map 作为出入参。

3.1.9.7. 解决方法

```
@RequestMapping("findSaleTaskDistArea")
    public BaseResponse<List<SaleTaskDistAreaVO>>
findSaleTaskDistArea(@RequestBody SaleTaskDistAreaRequst request)
```

3.1.10. 手动 GC 问题

3.1.10.1.案例概述

案例名称	严重性	适应性	备注
手动 GC 问题	严重	通用	

3.1.10.2.案例描述

手动 System.gc 主动进行垃圾回收是一个非常危险的动作。因为它要停止所有的响应，才能检查内存中是否有可回收的对象，这对一个应用系统风险极大。

3.1.10.3.错误代码示例

System.gc

3.1.10.4.检查点

全工程搜索 System.gc。

3.1.10.5.检查方法

SONAR 支持：

Do not call garbage collection explicitly：pmd:DoNotCallGarbageCollectionExplicitly

Execution of the Garbage Collector should be triggered only by the JVM： squid:S1215

3.1.10.6.禁止项

禁止手动 GC。

3.1.10.7.解决方法

删除 System.gc 的调用代码。

3.1.11. 使用流输入、输出时未指定字符编码问题

3.1.11.1.案例概述

案例名称	严重性	适应性	备注
使用流输入、输出时未指定字符编码问题	中等	通用	

3.1.11.2.案例描述

未指定字符编码，由于不同的平台可能使用的是不同的默认字符编码，这样的代码不具有跨平台可移植性，会导致乱码问题。

3.1.11.3.错误代码示例

```
Reader r = new FileReader(file);
Writer w = new FileWriter(file);
```

Reader r = new InputStreamReader(inputStream);
Writer w = new OutputStreamWriter(outputStream);
String s = new String(byteArray); // byteArray is a byte[]
byte[] a = string.getBytes();

3.1.11.4.检查点

所有使用流进行输入输出的代码（InputStreamReader、OutputStreamWriter）

3.1.11.5.检查方法

SONAR 支持：Correctness - Method encodes String bytes without specifying the character encoding（检查是否使用了编码，但没有统一要求检查编码）

3.1.11.6.禁止项

禁止使用不设置字符编码的流。

3.1.11.7.解决方法

```
Reader r = new InputStreamReader(new FileInputStream(file), "UTF-8");
Writer w = new OutputStreamWriter(new FileOutputStream(file), "UTF-8");
Reader r = new InputStreamReader(inputStream, "UTF-8");
Writer w = new OutputStreamWriter(outputStream, "UTF-8");
String s = new String(byteArray, "UTF-8");
byte[] a = string.getBytes("ASCII");
```

3.2. 多线程

3.2.1. 线程池的创建问题

3.2.1.1. 案例概述

案例名称	严重性	适应性	备注
线程池的创建问题	严重	通用	

3.2.1.2. 案例描述

线程池不允许使用 Executors 去创建，而是通过 ThreadPoolExecutor 的方式，这样的处理方式让写的同学更加明确线程池的运行规则，规避资源耗尽的风险。

说明： Executors 返回的线程池对象的弊端如下：

1） FixedThreadPool 和 SingleThreadPool 允许的请求队列长度为 Integer.MAX_VALUE ，可能会堆积大量的请求，从而导致 OOM 。

2）CachedThreadPool 和 ScheduledThreadPool 允许的创建线程数量为 Integer.MAX_VALUE ，可能会创建大量的线程，从而导致 OOM 。

3.2.1.3. 错误代码示例

```
Executors executors = newFixedThreadPool(int nThreads);
Executors executors = SingleThreadPool(int nThreads);
```

3.2.1.4. 检查点

所有多线程操作代码。

3.2.1.5. 检查方法

全工程搜索 newCachedThreadPool、newFixedThreadPool、newScheduledThreadPool、newSingleThreadExecutor 等关键字。

3.2.1.6. 禁止项

禁止使用 Executors 创建线程池。

3.2.1.7. 解决方法

```
ThreadPoolExecutor mThreadPool = new ThreadPoolExecutor(SIZE_CORE_POOL,
SIZE_MAX_POOL, 0L,TimeUnit.MILLISECONDS, new
LinkedBlockingQueue<Runnable>());
mThreadPool.submit();
```

3.2.2. 线程池资源不释放问题

3.2.2.1. 案例概述

案例名称	严重性	适应性	备注
线程池资源不释放问题	严重	通用	

3.2.2.2. 案例描述

线程池资源使用时申请，但是忘记释放。

3.2.2.3. 错误代码示例

```
XResource resource = pool.getResource();
//resource.dosomething();
……
```

3.2.2.4. 检查点

所有使用线程池进行多线程操作的代码。

3.2.2.5. 检查方法

SONAR 支持：

Correctness - ExecutorService field doesn't ever get shutdown : fb-contrib:HES_EXECUTOR_NEVER_SHUTDOWN

3.2.2.6. 禁止项

在使用线程池进行多线程操作时，禁止不进行释放操作。

3.2.2.7. 解决方法

```
//jdk1.7 开始的资源自动释放模式：
try (XResource resource =pool.getResource()) {
      resource.dosomething();
} catch (Exception e) {
    ……
}
//传统的 try 里申请，finally 释放模式:
XResource resource = null;
try  {
    resource = pool.getResource();
    resource.dosomething();
} catch (Exception e) {
    ……
}finally {
    if(null != jedis){
        pool.returnResource(resource);
    }
}
```

3.2.3.线程同步问题

3.2.3.1. 案例概述

案例名称	严重性	适应性	备注
线程同步问题	中等	长研	

3.2.3.2. 案例描述

使用 CountDownLatch 进行异步转同步操作，每个线程退出前必须调用 countDown 方法，线程执行代码注意 catch 异常，确保 countDown 方法被执行到，避免主线程无法执行至 await 方法，直到超时才返回结果。 说明：注意，子线程抛出异常堆栈，不能在主线程 try-catch 到。

3.2.3.3. 错误代码示例

```
//主线程代码：
ThreadPoolExecutor pool = new ThreadPoolExecutor(5, 5, 0L, TimeUnit.MILLISECONDS, blockQueue);
CountDownLatch latch = new CountDownLatch(iMaxTaskSize);
BreTask task = new BreTask(latch);
try{
 pool.execute(task);
}catch (Exception e){
 logger.error("往规则线程池投放任务发生异常!", e);
}
latch.await(20, TimeUnit.SECONDS);
pool.shutdown();

//子线程代码：
final class BreTask implements Runnable
{
 private CountDownLatch latch;
 public BreTask(CountDownLatch latch){
  this.latch = latch;
 }

 @Override
        public void run()
        {

            try{
  //Todo
     }catch(Exception ex){
  ex.printStackTrace();
     }
 }
}
```

3.2.3.4. 检查点

使用 CountDownLatch 进行异步转同步操作的多线程代码。

3.2.3.5. 检查方法

手工搜索 CountDownLatch 关键字。

3.2.3.6. 禁止项

主线程使用 CountDownLatch 来同步子线程执行状态时，必须在子线程逻辑执行完毕后调用 CountDownLatch.countDown 方法通知主线程

3.2.3.7. 解决方法

```
//主线程代码：
ThreadPoolExecutor pool = new ThreadPoolExecutor(5, 5, 0L,
TimeUnit.MILLISECONDS, blockQueue);
CountDownLatch latch = new CountDownLatch(iMaxTaskSize);
BreTask task = new BreTask(latch);
try{
 pool.execute(task);
}catch (Exception e){
 logger.error("往规则线程池投放任务发生异常!", e);
}
latch.await(20, TimeUnit.SECONDS);
pool.shutdown();

//子线程代码：
final class BreTask implements Runnable
{
 private CountDownLatch latch;
 public BreTask(CountDownLatch latch){
```

```
    this.latch = latch;
  }

  @Override
        public void run()
        {

            try{
   //Todo
      }catch(Exception ex){
   ex.printStackTrace();
      }finally
              {
                  latch.countDown();
              }
  }
}
```

3.2.4. 线程对象定义问题

3.2.4.1. 案例概述

案例名称	严重性	适应性	备注
线程对象定义问题	中等	通用	

3.2.4.2. 案例描述

对于线程内的变量尽量不要设置为 static 变量，如设置为 static 变量，则操作时一定要加锁。

3.2.4.3. 错误代码示例

```
public class MyClass {
```

```
private static SimpleDateFormat format = new SimpleDateFormat("HH-mm-ss");
static private Calendar calendar = Calendar.getInstance();
}
```

3.2.4.4. 检查点

多线程操作时变量定义部分的代码。

3.2.4.5. 检查方法

SONAR 支持：
Non-thread-safe fields should not be static. squid:S2885

3.2.4.6. 禁止项

线程内的变量禁止定义为 static 变量，如设置为 static 变量，则操作时一定要加锁。

3.2.4.7. 解决方法

```
public class MyClass {
 private SimpleDateFormat format = new SimpleDateFormat("HH-mm-ss");
 private Calendar calendar = Calendar.getInstance();
}
```

3.3. 数据库操作

3.3.1. SQL 注入问题

3.3.1.1. 案例概述

案例名称	严重性	适应性	备注
SQL 注入问题	严重	通用	

3.3.1.2. 案例描述

把 SQL 命令插入到 Web 表单提交或输入域名或页面请求的查询字符串，最终欺骗服务器，执行恶意的 SQL 命令。我们需要通过绑定变量来防止 SQL 注入。

3.3.1.3. 错误代码示例

```
public User getUser(Connection con, String user) throws SQLException {
   Statement stmt1 = null;
   Statement stmt2 = null;
   PreparedStatement pstmt;
   try {
      stmt1 = con.createStatement();
      // Compliant; parameters not used here
      ResultSet rs1 = stmt1.executeQuery("GETDATE()");
      stmt2 = con.createStatement();
      // Noncompliant; parameter concatenated directly into query
      ResultSet rs2 = stmt2.executeQuery("select FNAME, LNAME, SSN " +
                        "from USERS where UNAME=" + user);
      // Noncompliant; parameter concatenated directly into query
      pstmt = con.prepareStatement("select FNAME, LNAME, SSN " +
                        "from USERS where UNAME=" + user);
      ResultSet rs3 = pstmt.executeQuery();
      //...
}
public User getUserHibernate(org.hibernate.Session session, String userInput) {
   // Noncompliant; parameter binding should be used instead
   org.hibernate.Query query = session.createQuery( "FROM students where fname = " +
   userInput);
   // ...
}
```

3.3.1.4. 检查点

DAO 层拼接 SQL 语句的代码。

3.3.1.5. 检查方法

SONAR 支持：Values passed to SQL commands should be sanitize

3.3.1.6. 禁止项

禁止使用字符串拼接的方式拼装 SQL，必须使用绑定变量的方式来防止 SQL 注入。

3.3.1.7. 解决方法

```
public User getUser(Connection con, String user) throws SQLException {
  Statement stmt1 = null;
  PreparedStatement pstmt = null;
  String query = "select FNAME, LNAME, SSN  from USERS where UNAME=?"
  try {
    stmt1 = con.createStatement();
    ResultSet rs1 = stmt1.executeQuery("GETDATE()");
    pstmt = con.prepareStatement(query);
    // Compliant; PreparedStatements escape their inputs.
    pstmt.setString(1, user);
     ResultSet rs2 = pstmt.executeQuery();
    //...
  }
}
public User getUserHibernate(org.hibernate.Session session, String
userInput) {
  org.hibernate.Query query =  session.createQuery("FROM students where
  fname = ?");
 // Parameter binding escapes all input
  query = query.setParameter(0,userInput);
  // ...
```

3.3.2. 数据库链接泄漏问题

3.3.2.1. 案例概述

案例名称	严重性	适应性	备注
数据库链接泄漏问题	严重	长研以外所有 SRD	

3.3.2.2. 案例描述

数据库链接泄漏，链接池标记链接被占用，处于激活状态，不能被其它线程使用，但数据加层看到的是未激活状态

3.3.2.3. 错误代码示例

```
DBConnection conn = new DBConnection("cen1", true, false);
    try
    {
        inData.put("UPDATE_STAFF_ID", CSBizBean.getVisit().getStaffId());
        inData.put("UPDATE_DEPART_ID", CSBizBean.getVisit().getDepartId());

        SQLParser parser = new SQLParser(inData);
        parser.addSQL(" UPDATE TF_B_CTRM_RETURN SET ");
        parser.addSQL(" STATUS          = :STATUS, ");
        parser.addSQL(" TRADE_ID     = :TRADE_ID, ");
        parser.addSQL(" UPDATE_STAFF_ID     = :UPDATE_STAFF_ID, ");
        parser.addSQL(" UPDATE_DEPART_ID     = :UPDATE_DEPART_ID, ");
        parser.addSQL(" IS_SYNC     = :IS_SYNC, ");
        parser.addSQL(" RESULT_INFO     = :RESULT_INFO, ");
        parser.addSQL(" RESULT_CODE     = :RESULT_CODE, ");
        parser.addSQL(" RESULT_DETAIL     = :RESULT_DETAIL, ");
        parser.addSQL(" REMARK     = :REMARK, ");
        parser.addSQL(" UPDATE_TIME = SYSDATE ");
        parser.addSQL(" WHERE ROWID = :ROWID ");

    CrmDAO dao = DAOManager.createDAO(CrmDAO.class, Route.CONN_CRM_CEN);
        dao.executeUpdate(conn, parser.getSQL(), parser.getParam());
        conn.commit();
    }
    catch (Exception e)
```

```
{
  conn.rollback();
  CSAppException.apperr(CrmCommException.CRM_COMM_103, e.getMessage());
}
```

3.3.2.4. 检查点

数据库操作部分的代码。

3.3.2.5. 检查方法

sonar 支持：Resources should be closed.

3.3.2.6. 禁止项

禁止在数据库操作后不释放数据库链接。

3.3.2.7. 解决方法

```
DBConnection conn = new DBConnection("cen1", true, false);
    try
    {

 inData.put("UPDATE_STAFF_ID",CSBizBean.getVisit().getStaffId());

inData.put("UPDATE_DEPART_ID",CSBizBean.getVisit().getDepartId());
        SQLParser parser = new SQLParser(inData);
        parser.addSQL(" UPDATE TF_B_CTRM_RETURN SET ");
        parser.addSQL(" STATUS       = :STATUS, ");
        parser.addSQL(" TRADE_ID    = :TRADE_ID, ");
        parser.addSQL(" UPDATE_STAFF_ID    = :UPDATE_STAFF_ID, ");
        parser.addSQL("UPDATE_DEPART_ID    =:UPDATE_DEPART_ID,");
        parser.addSQL(" IS_SYNC    = :IS_SYNC, ");
        parser.addSQL(" RESULT_INFO    = :RESULT_INFO, ");
```

```
            parser.addSQL(" RESULT_CODE    = :RESULT_CODE, ");
            parser.addSQL(" RESULT_DETAIL    = :RESULT_DETAIL, ");
            parser.addSQL(" REMARK    = :REMARK, ");
            parser.addSQL(" UPDATE_TIME = SYSDATE ");
            parser.addSQL(" WHERE ROWID = :ROWID ");
             CrmDAO dao =
             DAOManager.createDAO(CrmDAO.class,Route.CONN_CRM_CEN);
            dao.executeUpdate(conn, parser.getSQL(),
            parser.getParam());
            conn.commit();
        }
        catch (Exception e)
        {
            conn.rollback();
            CSAppException.apperr(CrmCommException.CRM_COMM_103,
            e.getMessage());
        }finally{
            conn.close();
        }
```

3.3.3. 在 LOOP 语句中申请数据库链接问题

3.3.3.1. 案例概述

案例名称	严重性	适应性	备注
在 LOOP 语句中申请数据库链接问题	严重	通用	

3.3.3.2. 案例描述

资源申请，比如 DB 的连接的申请是需要代价的，尽量避免在 LOOP 语句中进行申请，

以免长期占用大量资源。

3.3.3.3. 错误代码示例

```
for(int i=0;i<size;i++){
   DbCon = ResouceFactory.NewInst();
   ....
}
```

3.3.3.4. 检查点

所有的 LOOP 语句前后。

3.3.3.5. 检查方法

SONAR 支持：规则待开发。

3.3.3.6. 禁止项

禁止在 LOOP 语句中申请数据库链接。

3.3.3.7. 解决方法

```
DbCon = ResouceFactory.NewInst();
for(int i=0;i<size;i++){
  ....
}
```

3.3.4. 大批量数据单条提交问题

3.3.4.1. 案例概述

案例名称	严重性	适应性	备注
大批量数据单条提交问题	严重	通用	

3.3.4.2. 案例描述

对于后台批量作业，频繁的 COMMIT 会引起大量 Redo Log 的物理 I/O，会极大的限制数据库的性能。可根据实际业务需要以及当前主机性能参数，分批提交事务（如: 1K-3K 为一批次进行分段 COMMIT），这样即可提高业务逻辑处理性能，还能及时释放占用的资源。

3.3.4.3. 错误代码示例

```
BEGIN
        FOR cur IN (SELECT * FROM t_ref) LOOP
          INSERT INTO t VALUES cur;
        COMMIT;
       END LOOP;
 END;
```

3.3.4.4. 检查点

大批量数据提交的 JAVA 代码及存储过程。

3.3.4.5. 检查方法

SONAR 支持：规则待开发。

3.3.4.6. 禁止项

禁止在大批量数据提交的 JAVA 代码及存储过程中使用单条提交模式。

3.3.4.7. 解决方法

禁止存储过程和 JAVA 代码在 LOOP 语句中单条提交，批量提交需判定数量。

```
v_count NUMBER;
BEGIN
 FOR cur IN (SELECT * FROM t_ref) LOOP
   INSERT INTO t VALUES cur;
   v_count := v_count + 1;
   IF v_count >= 100 THEN
```

```
        COMMIT;
      END IF;
    END LOOP;
    COMMIT;
  END;
```

3.3.5. 存储过程的异常处理问题

3.3.5.1. 案例概述

案例名称	严重性	适应性	备注
存储过程的异常处理问题	中等	长研	

3.3.5.2. 案例描述

如果 java 程序中调用存储过程，需要注意：
1.存储过程中的异常需要传递到前台，否则应用无法捕获异常回滚
2.过程里不要有事务控制，交给应用控制，否则出现事务不一致（需要独立事务除外）

3.3.5.3. 错误代码示例

```
BEGIN
……
END;
```

3.3.5.4. 检查点

所有存储过程。

3.3.5.5. 检查方法

手工检查。

3.3.5.6. 禁止项

禁止在存储过程里不使用 EXCEPTION 进行异常处理。

3.3.5.7. 解决方法

```
BEGIN
...
EXCEPTION
  WHEN OTHERS THEN
    v_resultcode := '-1';
    v_resulterrinfo := 'PCC 处理失败:' ||'-'||SQLERRM;
    RETURN;
END ;
```

3.3.6. WADE 框架变量绑定问题

3.3.6.1. 案例概述

案例名称	严重性	适应性	备注
WADE 框架变量绑定问题	严重	通用	

3.3.6.2. 案例描述

开发人员在使用 wade 框架方法 Dao.qryByParse(parser) 或者 Dao.qryByCodeParser(tabName, sqlref, param)进行查询的时候，sql 需要注意写法；parser 或者 sqlref 中所需要的变量在代码里如果没有绑定，框架会将对应的一行直接过滤，引发问题，例如

1.where 关键字如果与变量在同一行，没有传递对应变量会导致此行被过滤，报 sql 未正确结束；

2.如果需要走的索引变量没有传递，会导致全表扫描，影响整个性能。

3.如果条件全部丢失，导致查询全表数据返回

3.3.6.3. 错误代码示例

```
IData cond = new DataMap();
//没有将对应参数传递
return Dao.qryByCodeParser("TD_S_STATIC", "SEL_BY_TYPEID", cond);
```

3.3.6.4. 检查点

WADE 框架 DAO 层拼接 SQL 语句的代码。

3.3.6.5. 检查方法

手工搜索 Dao.qryByParse 或者 Dao.qryByCodeParser。

3.3.6.6. 禁止项

禁止在使用 parse 方法的时候不检查变量

3.3.6.7. 解决方法

```
IData cond = new DataMap();
cond.put("TYPE_ID", "ABILITY_PSPT_TYPE_CODE");
return Dao.qryByCode("TD_S_STATIC", "SEL_BY_TYPEID", cond);
```

3.4. 外部资源操作

3.4.1. FTP 资源未释放问题

3.4.1.1. 案例概述

案例名称	严重性	适应性	备注
FTP 资源未释放问题	严重	通用	

3.4.1.2. 案例描述

在操作 FTP 连接的时候，没有及时释放资源，导致 FTP 不可访问。

3.4.1.3. 错误代码示例

```
String ip = "10.1.234.146"; // 服务器 IP 地址
String userName = "dubbouser"; // 用户名
String userPwd = "dubbouser123"; // 密码
int port =22022; // 端口号
ChannelSftp sftp = SftpUtil.connect(ip, port, userName, userPwd);
try {
    SftpUtil.upload("/aifs01/dubbouser/","D:/JXL2003.xls",sftp);
    sftp.mkdir("/tmp/aa/bb/");
} catch (SftpException e) {
    e.printStackTrace();
}catch (Exception e1){
    e1.printStackTrace();
}
```

3.4.1.4. 检查点

所有使用 FTP 的代码。

3.4.1.5. 检查方法

sonar 支持：Resources should be closed

3.4.1.6. 禁止项

禁止在 FTP 资源使用完毕后不进行关闭操作。

3.4.1.7. 解决方法

```
String ip = ""; // 服务器 IP 地址
String userName = ""; // 用户名
String userPwd = ""; // 密码
int port =;// 端口号
ChannelSftp sftp = SftpUtil.connect(ip, port, userName, userPwd);
try {
```

```
        SftpUtil.upload("/aifs01/dubbouser/","D:/JXL2003.xls",sftp);
        sftp.mkdir("/tmp/aa/bb/");
    } catch (SftpException e) {
        e.printStackTrace();
    }catch (Exception e1){
        e1.printStackTrace();
    }finally {
         if(sftp == null){
            return;
    }
    try {
        if (sftp.getSession() != null && sftp.getSession().isConnected()) {
            sftp.getSession().disconnect();
        }
        if (sftp.isConnected()) {
            sftp.disconnect();
        }
        } catch (JSchException e) {
            logger.error("Disconnect sftp error.", e);
        }
    }
```

3.4.2. 资源文件未关闭问题

3.4.2.1. 案例概述

案例名称	严重性	适应性	备注
资源文件未关闭问题	严重	通用	

3.4.2.2. 案例描述

我们在操作文件的时候，没有及时释放资源，导致 IO 异常。

3.4.2.3. 错误代码示例

File zipFile = null;//压缩文件
//使用后没有关闭
……

3.4.2.4. 检查点

所有操作外部文件的代码

3.4.2.5. 检查方法

sonar 支持：Resources should be closed

3.4.2.6. 禁止项

禁止在资源文件使用完毕后不进行关闭操作。

3.4.2.7. 解决方法

```
File zipFile = null;//压缩文件
.......
finally{
      if(null!=zipFile){
           try{
               zipFile.close();
           }catch(Exception e){
               ……
           }
      }
}
```

3.5. 日志操作

3.5.1. LOG 变量定义问题

3.5.1.1. 案例概述

案例名称	严重性	适应性	备注
LOG 变量定义问题	中等	通用	

3.5.1.2. 案例描述

没有使用 final 关键字定义 Log 变量，后续 log 对象被修改，导致打印出的日志信息有误

3.5.1.3. 错误代码示例

```
public class MsgProducerSVImpl implements IMsgProducerSV {
    private static transient Log log = LogFactory.getLog(Order2BillingSVImpl.class);
    ……
}
```

3.5.1.4. 检查点

所有 log 对象的定义代码。

3.5.1.5. 检查方法

SONAR 支持：规则待开发。

3.5.1.6. 禁止项

禁止不使用 final 关键字定义 Log 变量。

3.5.1.7. 解决方法

```
public class MsgProducerSVImpl implements IMsgProducerSV {
    private final static transient Log log =
        LogFactory.getLog(Order2BillingSVImpl.class);
    ……
}
```

3.5.2. 直接使用 System.out 输出日志问题

3.5.2.1. 案例概述

案例名称	严重性	适应性	备注
直接使用 System.out 输出日志问题	严重	通用	

3.5.2.2. 案例描述

使用 System.out 等方式输出日志信息，很难对异常产生的原因进行定位。

3.5.2.3. 错误代码示例

System.out.println("eccpetion"+e);

3.5.2.4. 检查点

所有输出日志的代码。

3.5.2.5. 检查方法

SONAR 支持：system.out/system.err ；

3.5.2.6. 禁止项

所有日志输出采用 slf4j 或 log4j 等日志输出框架，禁止使用 System.out 等方式输出日志信息。

3.5.2.7. 解决方法

```
logger.info("eccpetion",e);
logger.erro("eccpetion",e);
```

3.5.3. 日志输出不判断日志级别问题

3.5.3.1. 案例概述

案例名称	严重性	适应性	备注
日志输出不判断日志级别	严重	通用	

3.5.3.2. 案例描述

日志输出未判断日志级别，导致系统性能下降。

3.5.3.3. 错误代码示例

```
log.debug("http"+request.getParameterMap());
System.out.println("输出为"+map);
```

3.5.3.4. 检查点

所有日志输出的代码。

3.5.3.5. 检查方法

SONAR 支持：Replace this usage of System.out or logger.debug a logger.

3.5.3.6. 禁止项

禁止在日志输出时不判断日志级别。

3.5.3.7. 解决方法

```
if(log.isDebugEnabled()){
  log.debug("http"+request.getParameterMap());
}
```

3.6. 配置文件

3.6.1. csf 服务空闲连接时间过短问题

3.6.1.1. 案例概述

案例名称	严重性	适应性	备注
csf 服务空闲连接时间过短	严重	通用	

3.6.1.2. 案例描述

csf 配置的空闲连接关闭时间比心跳检测超时时间还短，引发客户端调用超时，有一定的概率和频发性，建议采用默认配置。配置必须保证 csf 配置的空闲连接关闭时间大于心跳检测超时时间。

3.6.1.3. 错误代码示例

```
<Category name="client" description="socket 客户端的配置">
  <Item name="heart.beat.internal">
    <value>30000</value>
    <description>心跳间隔，链路空闲该时间段后发送心跳请求，默认 30s</description>
  </Item>
</Category>
```

```
<Category name="server" description="socket 服务端的配置">
 <Item name="channel.idle.time.out">
  <value>18000</value>
  <description>服务端关闭空闲连接时间，默认 70s</description>
 </Item>
</Category>
```

3.6.1.4. 检查点

检查工程中的配置文件 csf.socket.xml。

3.6.1.5. 检查方法

手工检查。

3.6.1.6. 禁止项

禁止配置项 channel.idle.time.out[服务端关闭空闲连接时间]值小于配置项 heart.beat.timeout[心跳超时时间]。

3.6.1.7. 解决方法

修改配置项 channel.idle.time.out 的值大于配置项 heart.beat.timeout 的值或使用默认配置。

```
<Category name="client" description="socket 客户端的配置">
 <Item name="heart.beat.internal">
  <value></value>
  <description>心跳间隔，链路空闲该时间段后发送心跳请求，默认
30s</description>
 </Item>
</Category>

<Category name="server" description="socket 服务端的配置">
 <Item name="channel.idle.time.out">
  <value></value>
```

<description>服务端关闭空闲连接时间，默认 70s</description>

</Item>

</Category>

3.6.2. 应用容器未配置线程池参数或参数配置过低问题

3.6.2.1. 案例概述

案例名称	严重性	适应性	备注
URL 及业务数据	严重	通用	

3.6.2.2. 案例描述

系统并发大时系统响应时间长,但 cpu,内存消耗却很低。极简单的业务也表现出耗时很长，时间消耗在等待服务器返回。可以检查下应用容器的线程池配置。应用容器的线程池配置过低，在大并发情况下会导致请求队列等待时间过长。

3.6.2.3. 错误代码示例

无

3.6.2.4. 检查点

配置文件中线程池参数配置部分。

3.6.2.5. 检查方法

手工检查

3.6.2.6. 禁止项

禁止不配置线程池参数或线程池参数配置过低。

3.6.2.7. 解决方法

线程池的最大线程数、最小线程数必须配置

配置参考如下：

AI-Container

Dwade.container.minThreads=100

Dwade.container.maxThreads=200

BES

iastool --pasport admin --port 7777 set standalone-instances.standalone-instance.config.${instance}-config.thread-pools.thread-pool.http-thread-pool.max-thread-pool-size=200

weblogic

-Dweblogic.threadpool.MinPoolSize=100

-Dweblogic.threadpool.MaxPoolSize=200

3.6.3. 系统页面或服务调用返回出现中文乱码问题

3.6.3.1. 案例概述

案例名称	严重性	适应性	备注
系统页面或服务调用返回出现中文乱码问题	严重	通用	

3.6.3.2. 案例描述

前台页面出现中文乱码，或后台服务调用返回中文乱码。

3.6.3.3. 错误代码示例

应用容器字符集参数未设置或设置错误，如：应用字符集为：
-Dcom.wade.container.util.URI.charset=GBK
但容器字符集参数为：
-Dfile.encoding=UTF-8

3.6.3.4. 检查点

按范例代码中提供的各类容器配置检查配置，确认应用容器字符集参数是否设置正确，是否与应用的字符集一致。

3.6.3.5. 检查方法

手工检查

3.6.3.6. 禁止项

禁止应用容器字符集参数与应用的字符集不一致。

3.6.3.7. 解决方法

AI-Container

-Dfile.encoding=UTF-8

BES

设置 url 中的中文是否需要编码，设置方式如下：

iastool --passport admin --host ${machine} --port 7777 --password ${bes_password} set standalone-instances.standalone-instance.config.${instance}-config.protocols.protocol.http-listener-1.http-protocol.property.useBodyEncodingForURI=true

设置 web 容器默认返回的 charset 为 UTF-8，设置方式如下：

```
iastool --passport admin --host ${machine} --port 7777 --password
${bes_password} set
standalone-instances.standalone-instance.config.${instance}-config.pr
otocols.protocol.http-listener-1.http-protocol.default-response-type=
text\/html\;charset\=UTF-8
```

设置 file.encoding 为 UTF-8

```
iastool --passport admin --host ${machine} --port 7777 --password
${bes_password} create --jvm-options --target ${instance}
-Dfile\.encoding=UTF-8
```

tomcat

```
<Connector port="8008" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443"  URIEncoding="UTF-8"/>
```

weblogic

```
 -Dweblogic.webservice.i18n.charset=utf-8
```