静态代码扫描 (二)——PMD 自定义规则实践

丁老九 · May 27, 2016 · Last by 丁老九 replied at August 23, 2017 · 1077 hits

> 上篇文章用一个简单的规则举例，帮助大家快速了解如何自定义规则。接下来我们开始使用更为复杂的例子进行实践。

我还是以自定义规则的思路逐步讲解，但是不会和上篇一样解释的非常详细，很多细节希望大家能够进行思考后获得恍然大悟的感觉：

1. 明确想要自定义的规则。

需要自定义的规则大多来自开发的需求。这次举例的规则是：
Log日志文件中不要输出敏感信息。根据过往的代码样本总结出可定义为敏感信息的字段包括：
- pid
- uid
- imei
- classname
- getLocalClassName
- getPackageCodePath
- getPackagePath
- android.os.Process.myPid
- android.os.Process.myUid
- android.os.Process.getUidForName

其中有些是字符串，有些是获取特定字段的方法，所涉及的主要是手机相关信息、类名、代码路径等。

2. 列举会触犯这种规则的所有不同的写法。

错误样例的代码：

```
public class TestLog{
    static Logger Log = Logger.getLogger("log");
    static boolean DEBUG = true;
    static boolean DEBUG1 = false;
     public static void main(String []args){
        Context cont = activity.getApplicationContext();
        String classname = activity.getLocalClassName();
        String pcodeName = cont.getPackageCodePath();
```

```
        int id= android.os.Process.myPid();
        String pid =String.valueOf(id);
        int uicd= android.os.Process.myUid();
        String uid = String.valueOf(uicd);
        int idname= android.os.Process.getUidForName("pay");
        String imei = ((TelephonyManager)getSystemService(TELEPHONY_SERVICE)).getDeviceId();
        int bbq=activity.getLocalClassName();

        Log.i("classname", classname);//触发规则
        Log.i("pcodeName", pcodeName);//触发规则
        Log.i("pid", pid);//触发规则
        Log.i("uid", uid);//触发规则
        Log.i("imei", imei); //触发规则
        Log.i("imei", imei.length);
        Log.i("imei", imei.size());
        Log.i("imei:", activity.getLocalClassName());//触发规则
        Log.i("imei:", MYUUID);
        Log.i("imei:", imei.toString());//触发规则
        Log.i("imei:", ab.imei.toString());//触发规则
        Log.i("imei:", bbq);//触发规则
        Log.i("imei:", idname);//触发规则
        Log.i("imei:", id);//触发规则
        Log.i("imei:", uicd);//触发规则
        Log.i("imei:", pcodeName);//触发规则
        Log.i("imei:", 101);

        if (DEBUG) {
            Log.i("imei", imei);//触发规则
        }
        if (DEBUG1) {
            Log.i("imei", imei);
        }
    }
}
```

我在代码中标记出了 `触发规则` 的代码，大家可以根据代码前后文理解这些代码语句输出了什么敏感信息。

3. 使用designer.bat分析所有写法的抽象语法树的特点。

在语法树分析阶段，就得把这些触发错误的各种代码进行归类。

a. 在日志中直接输出敏感信息字符串。

b. 日志中直接输出敏感信息字符串做了处理。

c. 日志中输出的变量在前文中被赋值敏感信息。

d. 日志被日志开关包裹，开关为true时，才会输出敏感信息。

4. 编写规则代码捕捉这种特点。

```
import java.util.ArrayList;
```

```
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import org.jaxen.JaxenException;
import net.sourceforge.pmd.lang.ast.Node;
import net.sourceforge.pmd.lang.java.ast.ASTBlock;
import net.sourceforge.pmd.lang.java.ast.ASTBlockStatement;
import net.sourceforge.pmd.lang.java.ast.ASTBooleanLiteral;
import net.sourceforge.pmd.lang.java.ast.ASTCompilationUnit;
import net.sourceforge.pmd.lang.java.ast.ASTExpression;
import net.sourceforge.pmd.lang.java.ast.ASTIfStatement;
import net.sourceforge.pmd.lang.java.ast.ASTMethodDeclaration;
import net.sourceforge.pmd.lang.java.ast.ASTName;
import net.sourceforge.pmd.lang.java.ast.ASTPrimaryPrefix;
import net.sourceforge.pmd.lang.java.ast.ASTStatementExpression;
import net.sourceforge.pmd.lang.java.ast.ASTUnaryExpressionNotPlusMinus;
import net.sourceforge.pmd.lang.java.ast.ASTVariableDeclarator;
import net.sourceforge.pmd.lang.java.ast.ASTVariableDeclaratorId;
import net.sourceforge.pmd.lang.java.ast.ASTVariableInitializer;
import net.sourceforge.pmd.lang.java.rule.AbstractJavaRule;

/**
 * @author yuanwei
 * @date 2017年07月06日
 * @category log中敏感信息泄露检测
 */
public class LogBlockRule extends AbstractJavaRule {
    private static Set<String> SensitiveStrings = new HashSet<String>();
    private List<ASTName> astNamewithLog =(List<ASTName>)new ArrayList<ASTName>();
    private List<ASTName> SASTNames =(List<ASTName>)new ArrayList<ASTName>();
    private List<ASTVariableDeclaratorId> SensitiveFieldVariables =(List<ASTVariableDeclaratorId>)new A
rrayList<ASTVariableDeclaratorId>();
    private List<String> BooleanTrueStrings=new ArrayList<String>();
    private List<String> BooleanFalseStrings=new ArrayList<String>();

    static {
        SensitiveStrings.add("classname");
        SensitiveStrings.add("pid");
        SensitiveStrings.add("uid");
        SensitiveStrings.add("imei");
        SensitiveStrings.add("getLocalClassName");
        SensitiveStrings.add("getPackagePath");
        SensitiveStrings.add("android.os.Process.myPid");
        SensitiveStrings.add("android.os.Process.myUid");
        SensitiveStrings.add("android.os.Process.getUidForName");
    }


    @Override
    public Object visit(ASTCompilationUnit node, Object data) {
        startInitData(node);//初始化数据
        checkLogRule(node,data);
        return super.visit(node, data);
    }
```

```
    @SuppressWarnings("unchecked")
    private void checkLogRule(Node node,Object data) {
        if(!this.astNamewithLog.isEmpty()){
            try {
                List<ASTName> xpathLogNames = this.astNamewithLog; //将过滤以后的ASTname节点赋值给它，不再使用xpath
                if(xpathLogNames.size()>0){
                    for(ASTName name:xpathLogNames){
                        ASTIfStatement ifStatement=name.getFirstParentOfType(ASTIfStatement.class);
                        ASTBlockStatement blockStatement=name.getFirstParentOfType(ASTBlockStatement.class);
                        List<ASTName> names2=(List<ASTName>) blockStatement.findDescendantsOfType(ASTName.class);
                        if(names2.size()>0){
                            for(ASTName name2:names2){
                                String imageString2=name2.getImage();
                                boolean sflag=checkIsSensitiveString(imageString2);

                                //当前没发现包含敏感信息，把该ASTName节点存储后续解析
                                if (!sflag) {
                                    this.SASTNames.add(name2);
                                }
                                //当前发现包含敏感信息，确认是否被if包围
                                if (sflag) {
                                    //被if判断包围，确认判断条件是否为true
                                    if(ifStatement!=null){
                                        if (checkConditionTrue(ifStatement)) {
                                            addViolation(data, name2);
                                        }
                                    }
                                    //没有被if判断包围，判断是否在执行到敏感信息语句前return
                                    if (ifStatement==null) {
                                        Boolean returnFlag=checkIfReturnStatement(name2);
                                        //没有在之前return，触发规则
                                        if (!returnFlag) {
                                            addViolation(data, name2);
                                        }
                                    }
                                }
                            }
                        }
                    }
                    //新增第二层敏感信息监测
                    secendCheckLog(node,data);
                }
            }finally{
                this.astNamewithLog.clear();
                this.SASTNames.clear();
                this.BooleanTrueStrings.clear();
                this.BooleanFalseStrings.clear();
                this.SensitiveFieldVariables.clear();
            }
        }
```

```
    }

    //新增第二层敏感信息监测
    private void secendCheckLog(Node node, Object data) {
        //先找出带有敏感信息的全局变量
        try {
            List<ASTVariableDeclaratorId> variableDeclaratorIds=(List<ASTVariableDeclaratorId>)node.fin
dChildNodesWithXPath("//FieldDeclaration//VariableDeclarator/VariableDeclaratorId");
            //检查ASTVariableDeclaratorId节点是否包含敏感信息
            checkIsSensitiveFieldVariable(variableDeclaratorIds);
        } catch (JaxenException e) {
            // TODO 自动生成的 catch 块
            e.printStackTrace();
        }
        //依次找出各个日志输出语句方法体内的变量
        if (this.SASTNames.size()>0) {
            for(ASTName SecondastName:this.SASTNames){
                ASTMethodDeclaration astMethodDeclaration=SecondastName.getFirstParentOfType(ASTMethodD
eclaration.class);
                if (astMethodDeclaration==null) {
                    continue;
                }
                List<ASTVariableDeclaratorId> methodVariableDeclaratorIds=astMethodDeclaration.findDesc
endantsOfType(ASTVariableDeclaratorId.class);
                List<ASTVariableDeclaratorId> SensitiveLocalVariables =
                checkIsSensitiveLocalVariable(methodVariableDeclaratorIds);
                if(this.SensitiveFieldVariables.size()>0){
                    checkSecondViolation(this.SensitiveFieldVariables,SecondastName,data);
                }
                if (SensitiveLocalVariables.size()>0) {
                    checkSecondViolation(SensitiveLocalVariables,SecondastName,data);
                }
            }
        }
    }

    //检查带有敏感信息的变量有没有在日志中输出
    private void checkSecondViolation(
            List<ASTVariableDeclaratorId> sensitiveFieldVariables2,
            ASTName secondastName, Object data) {
        String astNameimage=secondastName.getImage();
        for(ASTVariableDeclaratorId SensitiveVariable:sensitiveFieldVariables2){
            if(!(hasNullInitializer(SensitiveVariable)) && astNameimage!=null&&SensitiveVariable.getIma
ge().equalsIgnoreCase(astNameimage)){
                //被if判断包围
                ASTIfStatement ifStatement=secondastName.getFirstParentOfType(ASTIfStatement.class);
                if(ifStatement!=null){
                    if (checkConditionTrue(ifStatement)) {
                        addViolation(data, secondastName);
                    }
                }
                //没有被if判断包围，判断是否在执行到敏感信息语句前return
                if (ifStatement==null) {
                    Boolean returnFlag=checkIfReturnStatement(secondastName);
```

```
                    //没有在之前return，触发规则
                    if (!returnFlag) {
                        addViolation(data, secondastName);
                    }
                }
            }
        }
    }

    //检查全局变量ASTVariableDeclaratorId节点是否包含敏感信息
    private List<ASTVariableDeclaratorId> checkIsSensitiveLocalVariable(
            List<ASTVariableDeclaratorId> methodVariableDeclaratorIds) {
        List<ASTVariableDeclaratorId> SensitiveLocalVariables = (List<ASTVariableDeclaratorId>)new Arra
yList<ASTVariableDeclaratorId>();
        if (methodVariableDeclaratorIds.size()>0) {
            for(ASTVariableDeclaratorId variableDeclaratorId:methodVariableDeclaratorIds){
                if(variableDeclaratorId.jjtGetParent() instanceof ASTVariableDeclarator){
                    ASTName astName=variableDeclaratorId.getFirstParentOfType(ASTVariableDeclarator.cla
ss).getFirstDescendantOfType(ASTName.class);
                    if(astName!=null){
                        if (checkIsSensitiveString(astName.getImage())) {
                            SensitiveLocalVariables.add(variableDeclaratorId);

                        }
                    }
                }
            }
        }
        return SensitiveLocalVariables;
    }
    //检查本地变量ASTVariableDeclaratorId节点是否包含敏感信息
    private void checkIsSensitiveFieldVariable(
            List<ASTVariableDeclaratorId> variableDeclaratorIds) {
        if (variableDeclaratorIds.size()>0) {
            for(ASTVariableDeclaratorId variableDeclaratorId:variableDeclaratorIds){
                if(variableDeclaratorId.jjtGetParent() instanceof ASTVariableDeclarator){
                    ASTName astName=variableDeclaratorId.getFirstParentOfType(ASTVariableDeclarator.cla
ss).getFirstDescendantOfType(ASTName.class);
                    if(astName!=null){
                        if (checkIsSensitiveString(astName.getImage())) {
                            this.SensitiveFieldVariables.add(variableDeclaratorId);

                        }
                    }
                }
            }
        }
    }

    //判断if条件是否为true
    private boolean checkConditionTrue(ASTIfStatement ifStatement) {
        // TODO 自动生成的方法存根
        ASTExpression astExpression=ifStatement.getFirstDescendantOfType(ASTExpression.class);
        ASTName astName =astExpression.getFirstDescendantOfType(ASTName.class);
```

```
        ASTUnaryExpressionNotPlusMinus astUENotPlusMinus=ifStatement.getFirstDescendantOfType(ASTUnaryE
xpressionNotPlusMinus.class);
        if(astName!=null){
            String astNameString=astName.getImage();
            //存在非！符号时
            if (astUENotPlusMinus!=null&&"!".equals(astUENotPlusMinus.getImage())) {
                if(this.BooleanFalseStrings.size()>0 && this.BooleanFalseStrings.contains(astNameString
)){
                    return true;
                }
            }else {
                if(this.BooleanTrueStrings.size()>0 && this.BooleanTrueStrings.contains(astNameString))
{
                    return true;
                }
            }
        }
        return false;
    }

    //初始化数据
    private void startInitData(Node node) {
        pickUpLogMethods(node); //遍历找出源代码中的Log.*代码
        pickUpBooleanNames(node);//获取所有布尔型的值的名称
    }

    //获取所有布尔型的值的名称
    private void pickUpBooleanNames(Node node) {
        // TODO 自动生成的方法存根
        String xpathBooleanTrue = ".//FieldDeclaration/VariableDeclarator/VariableInitializer/Expressio
n/PrimaryExpression"
                +"/PrimaryPrefix/Literal/BooleanLiteral[@True='true']";
        String xpathBooleanFalse = ".//FieldDeclaration/VariableDeclarator/VariableInitializer/Expressi
on/PrimaryExpression"
                +"/PrimaryPrefix/Literal/BooleanLiteral[@True='false']";
        List<ASTBooleanLiteral> xpathBooleanTrueNames;
        List<ASTBooleanLiteral> xpathBooleanFalseNames;
        try {
            xpathBooleanTrueNames = (List<ASTBooleanLiteral>) node.findChildNodesWithXPath(xpathBoolean
True);
            if(xpathBooleanTrueNames.size()>0){
                for(ASTBooleanLiteral booleanLiteral:xpathBooleanTrueNames){
                    ASTVariableDeclarator variableDeclarator = booleanLiteral.getFirstParentOfType(ASTV
ariableDeclarator.class);
                    ASTVariableDeclaratorId variableDeclaratorId = variableDeclarator.getFirstChildOfTy
pe(ASTVariableDeclaratorId.class);
                    this.BooleanTrueStrings.add(variableDeclaratorId.getImage());
                }
            }
            xpathBooleanFalseNames = (List<ASTBooleanLiteral>) node.findChildNodesWithXPath(xpathBoolea
nFalse);
            if(xpathBooleanFalseNames.size()>0){
                for(ASTBooleanLiteral booleanLiteral:xpathBooleanFalseNames){
                    ASTVariableDeclarator variableDeclarator = booleanLiteral.getFirstParentOfType(ASTV
```

```
ariableDeclarator.class);
                    ASTVariableDeclaratorId variableDeclaratorId = variableDeclarator.getFirstChildOfTy
pe(ASTVariableDeclaratorId.class);
                    this.BooleanFalseStrings.add(variableDeclaratorId.getImage());
                }
            }
        } catch (JaxenException e) {
            // TODO 自动生成的 catch 块
            e.printStackTrace();
        }
    }

    // 检查是否在执行敏感信息输出语句前return
    // 返回true说明使用了return语句在之前返回
    private Boolean checkIfReturnStatement(ASTName name) {
        int a = name.getEndLine();
        //System.out.println("a="+a);
        ASTMethodDeclaration astMethodDeclaration=name.getFirstParentOfType(ASTMethodDeclaration.class)
;
        ASTBlock astBlock=astMethodDeclaration.getFirstChildOfType(ASTBlock.class);
        List<Node> returnList;
        try {
            returnList = astBlock.findChildNodesWithXPath("//IfStatement//ReturnStatement");
            if (returnList.size()>0) {
                //System.out.println("returnList.size="+returnList.size());
                for (Node node : returnList) {
                    int b=node.getEndLine();
                    //System.out.println("b="+b);
                    if (b<a) {
                        ASTIfStatement ifStatement=node.getFirstParentOfType(ASTIfStatement.class);
                        if (ifStatement!=null) {
                            if (checkConditionTrue(ifStatement)) {
                                return true;
                            }
                        }
                    }
                }
            }
        } catch (JaxenException e) {
            // TODO 自动生成的 catch 块
            e.printStackTrace();
        }
        return false;
    }

    //判断是否包含敏感信息
    private boolean checkIsSensitiveString(String imageString2) {
        // TODO 自动生成的方法存根
        for(String SensitiveString:SensitiveStrings){
            if(imageString2.equalsIgnoreCase(SensitiveString)){
                return true;
            }
            if (imageString2!=null&&imageString2.contains(".")) {
                String[] partStrings=imageString2.split("\\.");
```

```
                int LastIndex=partStrings.length-1;
                if (partStrings[LastIndex].equals("length")||partStrings[LastIndex].equals("size")) {
                    return false;
                }else {
                    for (int i = 0; i < partStrings.length; i++) {
                        String partString = partStrings[i];
                        if (partString.equalsIgnoreCase(SensitiveString)) {
                            return true;
                        }
                    }
                }
            }
        }
        return false;
    }

    //判断变量是否为null，如果初始化为null则剔除
    private boolean hasNullInitializer(ASTVariableDeclaratorId var) {
        ASTVariableInitializer init = var.getFirstDescendantOfType(ASTVariableInitializer.class);
        if (init != null) {
            try {
                List<?> nulls = init.findChildNodesWithXPath("Expression/PrimaryExpression/PrimaryPrefi
x/Literal/NullLiteral");
                return !nulls.isEmpty();
            } catch (JaxenException e) {
                return false;
            }
        }
        return false;
    }

    //遍历找出源代码中的Log.*代码
    private void pickUpLogMethods(Node node){
        List<ASTStatementExpression> pexs =  node.findDescendantsOfType(ASTStatementExpression.class);
        for(ASTStatementExpression ast : pexs){
            ASTPrimaryPrefix primaryPrefix = ast.jjtGetChild(0).getFirstDescendantOfType(ASTPrimaryPref
ix.class);
            if (primaryPrefix != null) {
                ASTName name = primaryPrefix.getFirstChildOfType(ASTName.class);
                if (name != null) {
                    String imageString = name.getImage();
                    if(imageString.startsWith("Log.")){
                        astNamewithLog.add(name);
                    }
                }
            }
        }
    }
}
```

为了方便大家的理解，代码加了注释，但还是要结合第三点提出的四条特点进行理解。

同时我想说明一下，最深刻的理解在于动手实践，我上文提供的代码无需调试，可直接复制使用，希望大家在动手实践后能有自己的收获。

5. 创建自己的xml规则文件，内容包括规则的相关信息。

在上一篇文章中已经详细的介绍过，不再赘述。

6. 运行PMD扫描错误代码，验证是否能触发自定义规则。

特别需要注意的一点，规则完成后，使用样例代码验证只是确认自己撰写的逻辑是否正确。后续一定要经过大量实际代码扫描的验证，逐渐的提高规则的准确性，降低误报率。

360Qtest团队公众号

关注公众号，第一时间收到我们推送的新文章~

2 个赞

共收到 6 条回复

时间 点赞

taki #1 · May 28, 2016

pmd误爆率咋样

—— 来自TesterHome官方 安卓客户端

丁老九 #2 · May 30, 2016 作者

#1 @taki PMD

楼 是误报率吧。 是基于模式匹配来定位问题的，所以说如果当你自定义规则时考虑的不周全时，就会出现误报。误报率的降低需要你不断的拿自定义的规则在大量代码中进行验证，寻找自己考虑的不周全之处。PMD就好比一辆车，误报率就好比事故率，当你从新手成长为老司机，误报率也就不是问题了。

taki #3 · May 30, 2016

#2楼 @oggboy 恩，就怕误报率太高导致结果不可信

 丁老九 在 静态代码扫描 (三)——FindBugs 自定义规则入门 中提及了此贴 08 May 11:53

 丁老九 在 静态代码扫描 (四)——Java 资源关闭研究 中提及了此贴 08 May 11:53

 丁老九 在 静态代码扫描 (五)——Java 资源关闭的特殊场景 中提及了此贴 15 May 11:31

liwei #7 · June 12, 2017

```
List<ASTName> names2=(List<ASTName>) blockStatement.findDescendantsOfType(ASTName.class);
```

这句话得到的names2中是不是也包含了Log.d?

丁老九 #8 · June 20, 2017 作者

对  **liwei** #7 回复

从Log.d的第一个父节点ASTBlockStatement放下找子孙节点ASTName，肯定会包括Log.d本身的。你的意思是有更好的方法可以过滤掉Log.d本身节点么?

丁老九 #9 · August 23, 2017 作者

代码结构调整更新

需要 Sign In 后方可回复, 如果你还没有账号请点击这里 Sign Up 。

相关话题

uiautomatorviewer 新增功能 compressed 之 Device 端细节

jmeter 接口自动化测试方案二 (报告优化) 增加 git 路径

IMethodInterceptor 监听器入门级教学 (通过 IMethodInterceptor 监听器实现运行指定的组)

QualityCenter 案例导出工具源码

Robotium 娱乐小工具，临门一脚把我搞吐血了

作者

oggboy (丁老九)

第 1064 位Users / 2014-06-12

VIP

“ 这家伙很懒，什么个性签名都没有留下。 ”