基于WEB的 JAVA灰盒安全测试技术分享

演讲人：吴卓群

职务：安恒安全研究院负责人

日期：2014年09月

Why

- 目前常用的WEB应用自动化测试程序
  - 白盒测试(源码审计系统)
    - 需要获得源代码，对人员的安全专业知识要求高
    - 误报率高
    - 逻辑顺序关联的问题无法测试
  - 黑盒测试(自动化WEB扫描器)
    - 依靠页面响应，很多漏洞无法检测
      - 存储跨站
      - 页面无变化的注入(update, insert 等)
      - 大部分的代码注入
      - 很多文件操作相关的漏洞
    - 需要依靠爬虫的能力，测试覆盖面比较低
    - 为减少漏报率，需要编写大量的测试向量

灰盒测试

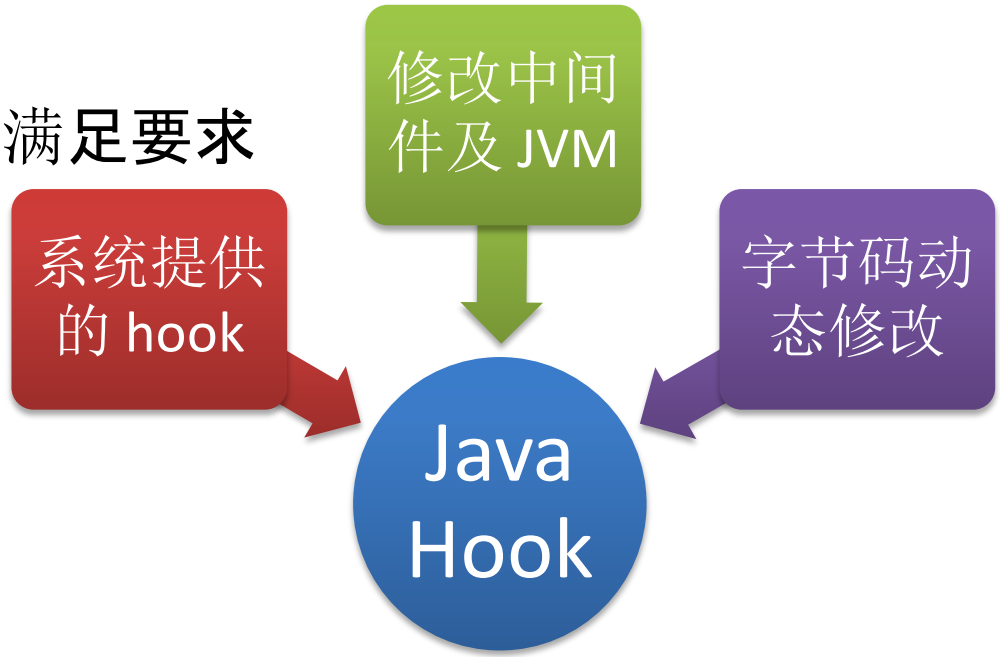
- 介于白盒和黑盒之间，对应用进行深度的测试
  - 不需要源代码支持
  - 能检测到黑盒无法检测的无回显差异的漏洞
  - 能检测带白盒无法测试逻辑顺序复杂的漏洞
  - 对测试人员要求低

利用Hook的方式进行数据劫持，并进行测试

Java Hook方式

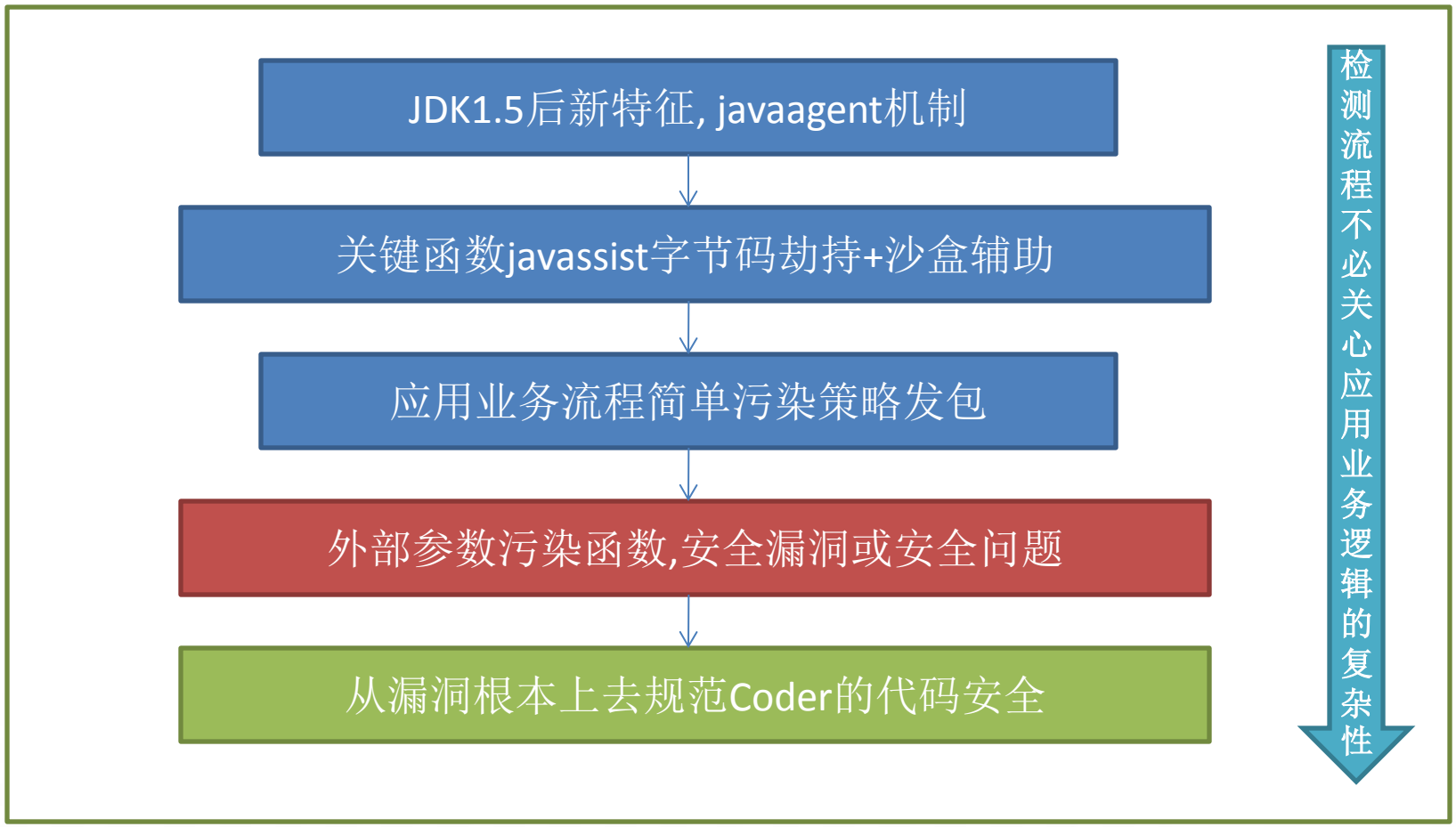
太过繁琐，兼容性不好
无法劫持上层调用函数

无法满足要求

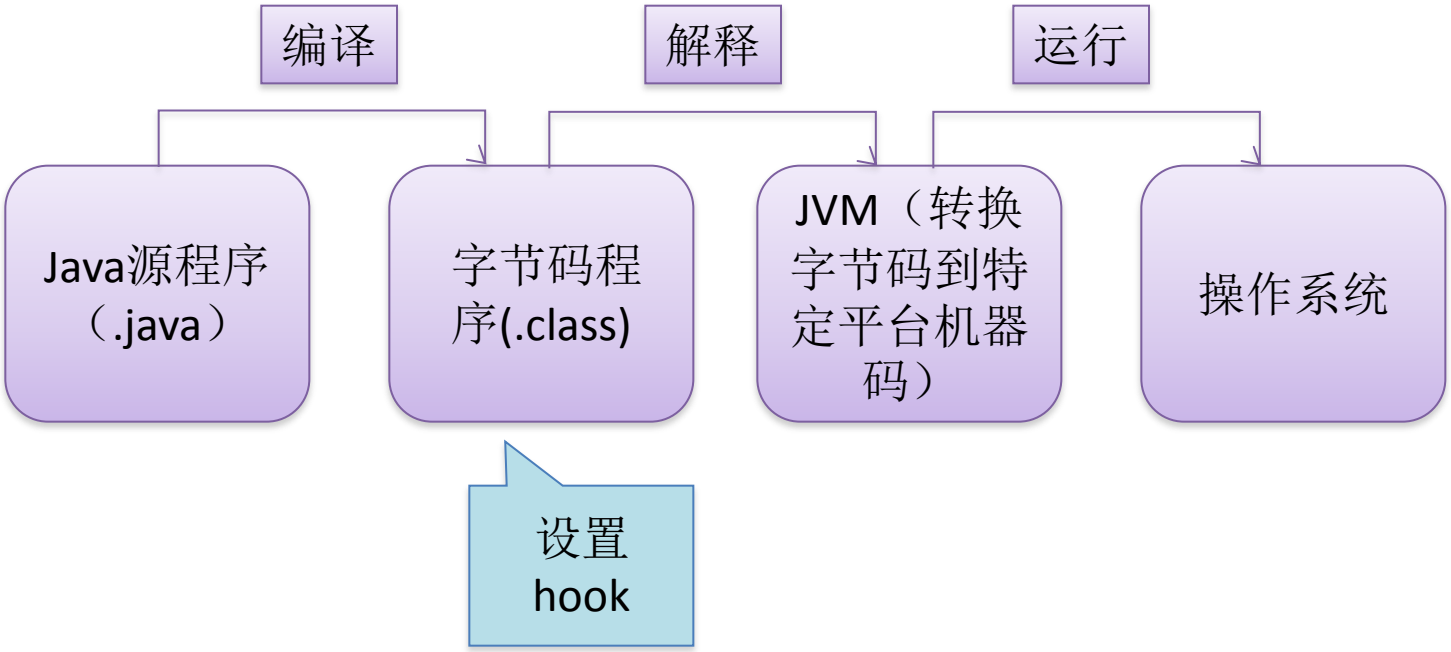


灰盒测试框架

- 适应灰盒测试概念，关注J2EE中间件关键函数污染表现



JAVA 编译执行代码的过程



Javassist

- 强大的 Javassist
  - Javassist 是一个开源的分析、编辑和创建Java字节码的类库。Javassist 是 jboss 的一个子项目，其主要的优点，在于简单，而且快速。直接使用java编码的形式，而不需要了解虚拟机指令，就能动态改变类的结构，或者动态生成类。
  - 利用 javassist 对目标函数动态注入字节码代码

动态修改字节码


```
public byte[] transform(java.lang.ClassLoader loader, String className,
        @SuppressWarnings("rawtypes")
        Class redefiningClass, ProtectionDomain domain, byte[] paramArrayOfByte)
        throws IllegalClassFormatException{
        try{
            String classFullName = className.replace("/", ".");
            ClassPool pool = ClassPool.getDefault();
            pool.insertClassPath(new ByteArrayClassPath(classFullName, paramArrayOfByte));


            for(int i = 0; i < classInjects.length; i ++){
                if(className.equals(classInjects[i].getPath()))
                {

                    JagentInjectMethod[] methods = classInjects[i].getMethods();

                    CtClass ctClass = pool.get(classFullName);

                    for(int j = 0; j < methods.length; j++){
                        try{
                            ctClass = ClassTools.classChange(ctClass, methods[j].getMethod(), methods[j].g
                                    methods[j].getCallBackFunc(), paramArrayOfByte, methods[j].getCal Back
                        }catch(Exception e){
                            JagentLogger.printLog(JagentLogger.ERROR, "inject function failed", e);
                        }
                    }
                    return ctClass.toBytecode();
```

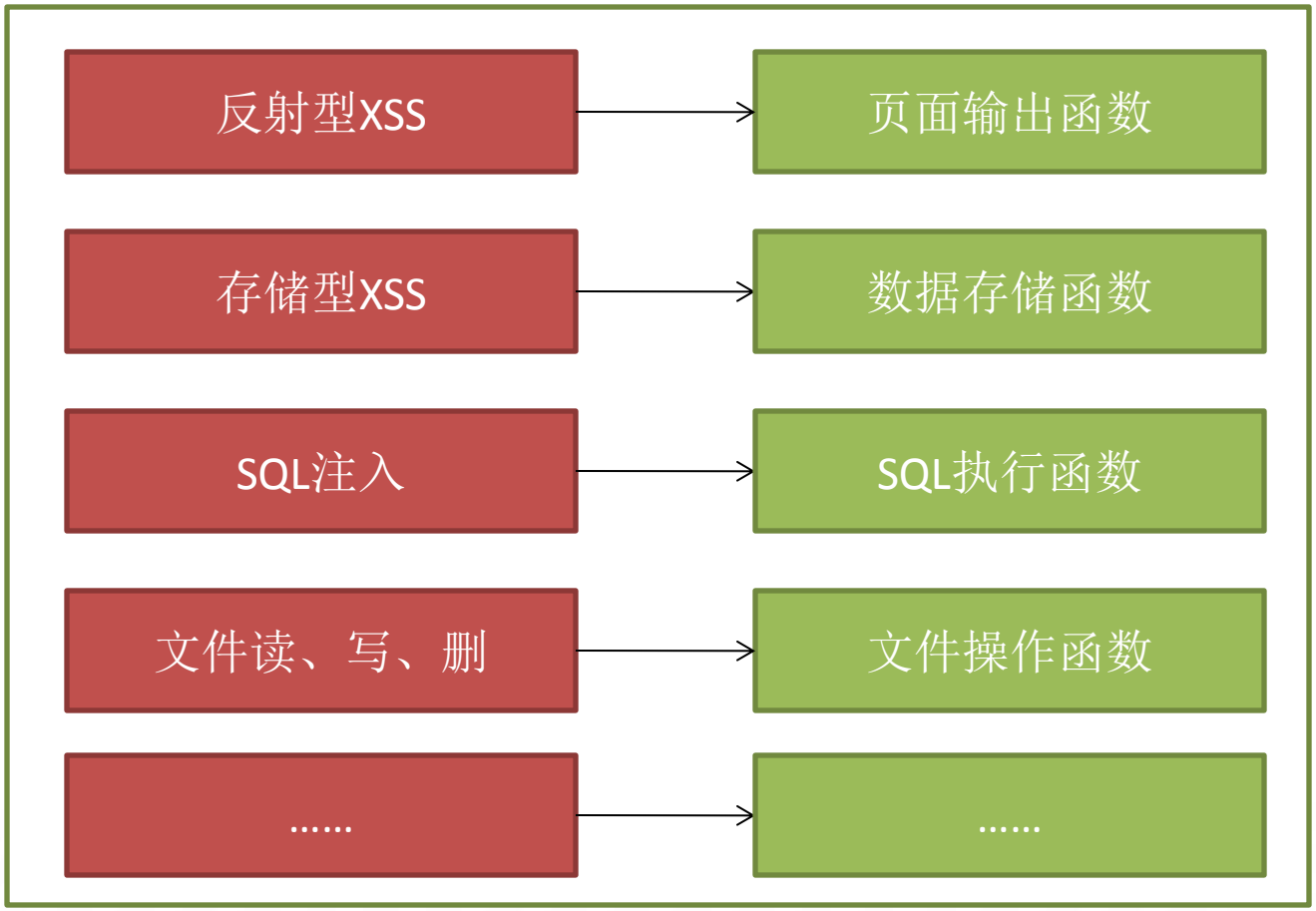
动态修改字节码

- Javassist 实现代码动态修改

```
    if(argNum == 0){
        body = String.format("{" +
                    "%s myclass = new %s();" +
                    "return myclass.%s((Object)this, \"%s\", null);" +
                    "}", callbackClass, callbackClass, callBackFunc, methodName);

    }else{
         body = String.format("{" +
                    "%s myclass = new %s();" +
                    "Object[] objs = $args;" +
                    "return myclass.%s((Object)this, \"%s\", objs);" +
                    "}", callbackClass, callbackClass, callBackFunc, methodName);
    }

}

ctMethod.insertBefore(body);
return ctClass;
```

Hook 函数

- **Web在代码层面的常见安全漏洞,来自参数在关键函数的污染**



Tomcat Hook 函数

- Tomcat 为例，劫持的关键函数
  - Request session请求初始化函数
  - Request session销毁函数
  - 数据库查询函数
  - 页面输出函数
  - ……

Tomcat Hook 函数

- Request请求初始化函数
  - 只需要能在执行执行其他劫持函数前获得request 请求的函数都可以
  - Org.apache.catalina.connector.Request 类
  - setRequestedSessionId函数

- Request请求销毁函数
  - 其他函数执行结束后request销毁前执行的函数都可以
  - org.apache.catalina.connector.Request 类
  - recycle

Tomcat Hook 函数

- 数据库查询函数
  - 各种 jdbc 的 class 库中的执行 sql 语句的函数
  - 如:

    com.mysql.jdbc.StatementImpl 类

    executeQuery 函数

    可检测存储跨站或注入漏洞

Tomcat Hook 函数

- 页面输出函数
  - Org.apache.jasper.runtime.JspWriterImpl
    write 函数

检测跨站脚本、信息泄露等漏洞

Tomcat Hook 函数

- 使用反射解决变量类型问题

```
public String getRequestURL(){
    Method m = getMethod("getRequestURL");
    if(m == null) return null;
    try{
        StringBuffer sb = (StringBuffer)m.invoke(this.request);
        return sb.toString();
    }catch(Exception e){
        return null;
    }
}
public String getParameter(String name){
    Method m = getMethod("getParameter", name.getClass());
    if(m == null) return null;
    try{
        return (String)m.invoke(this.request, name);
    }catch(Exception e){
        e.printStackTrace();
        return null;
    }
}
```

系统函数的检测

- 系统函数的 HOOK
  - 部分漏洞的操作实习并非中间件，如
    - 文件操作的漏洞
  - 无法通过hook中间件实现，只能hook系统函数完成

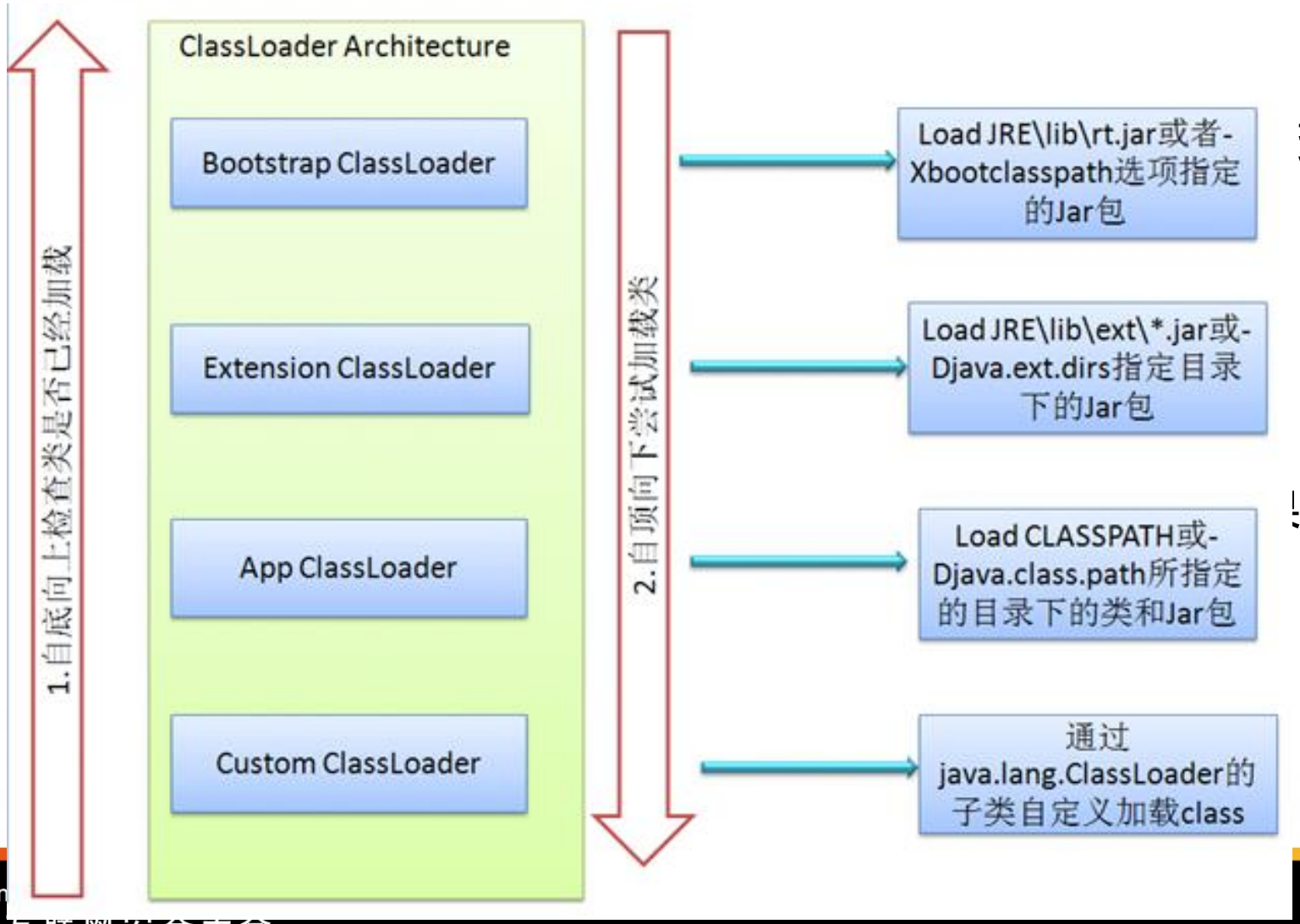
系统函数的检测

- Java.lang 包的处理
  - JVM启动是加载Runtime，File等类加载优先于 premain 函数，所以无法劫持
  - Java.lang 中的 class 出于安全考虑无法 redefine 或重新加载

系统函数的检测

- Xbootclass 和 SecurityManager
  - -Xbootclasspath:bootclasspath 让 jvm 从指定路径（可以是分号分隔的目录、jar、或者zip）中加载bootclass，用来替换jdk的rt.jar
  - SecurityManager，java的安全管理器（沙盘）

ClassLoader加载流程



- 到8,
- 找

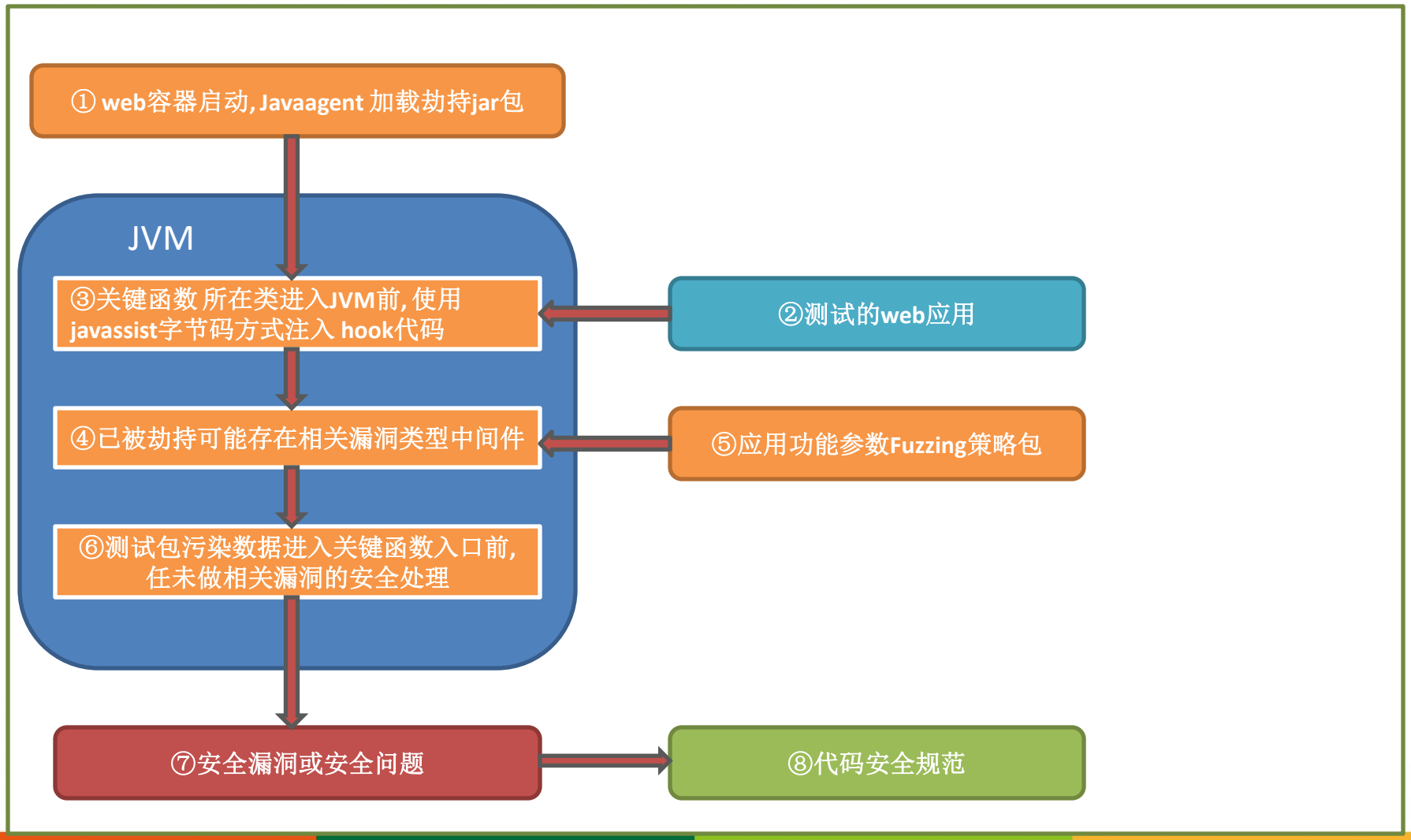
SecurityMananger

- SecurityMananger

```
public static void premain(String paramString,
        Instrumentation paramInstrumentation){
```

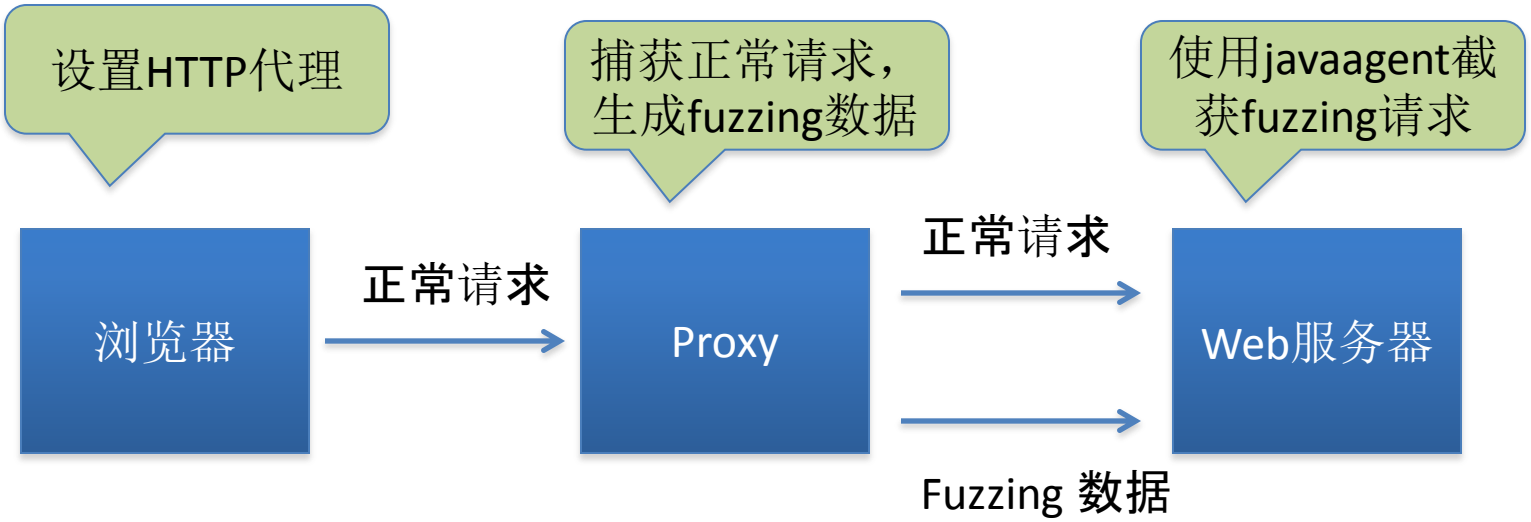
```
public class JagentSecurityManager extends SecurityManager {

    public void checkAccess(Thread paramThread){
        //System.out.println("checkAccess: " + paramThread.toString() );
    }
    public void checkRead(String file){
        FileSecCallBack.checkRead(file);
        //System.out.println("checkRead: " + file );
    }
    public void checkRead(String paramString, Object paramObject){
        FileSecCallBack.checkRead(paramString, paramObject);
        //System.out.println("checkRead: " + paramString );
    }
}
```

灰盒测试框架


① web容器启动, Javaagent 加载劫持jar包
JVM
③关键函数 所在类进入JVM前, 使用 javassist字节码方式注入 hook代码
②测试的web应用
④已被劫持可能存在相关漏洞类型中间件
⑤应用功能参数Fuzzing策略包
⑥测试包污染数据进入关键函数入口前, 任未做相关漏洞的安全处理
⑦安全漏洞或安全问题
⑧代码安全规范

灰盒测试框架

- 如何获得测试用例



?检测漏洞类型覆盖

反射型XSS	存储型XSS	SQL注入
系统命令注入	文件操作类漏洞	OGNL代码注入
信息泄露	安全目录绕过	URL跳转
……		

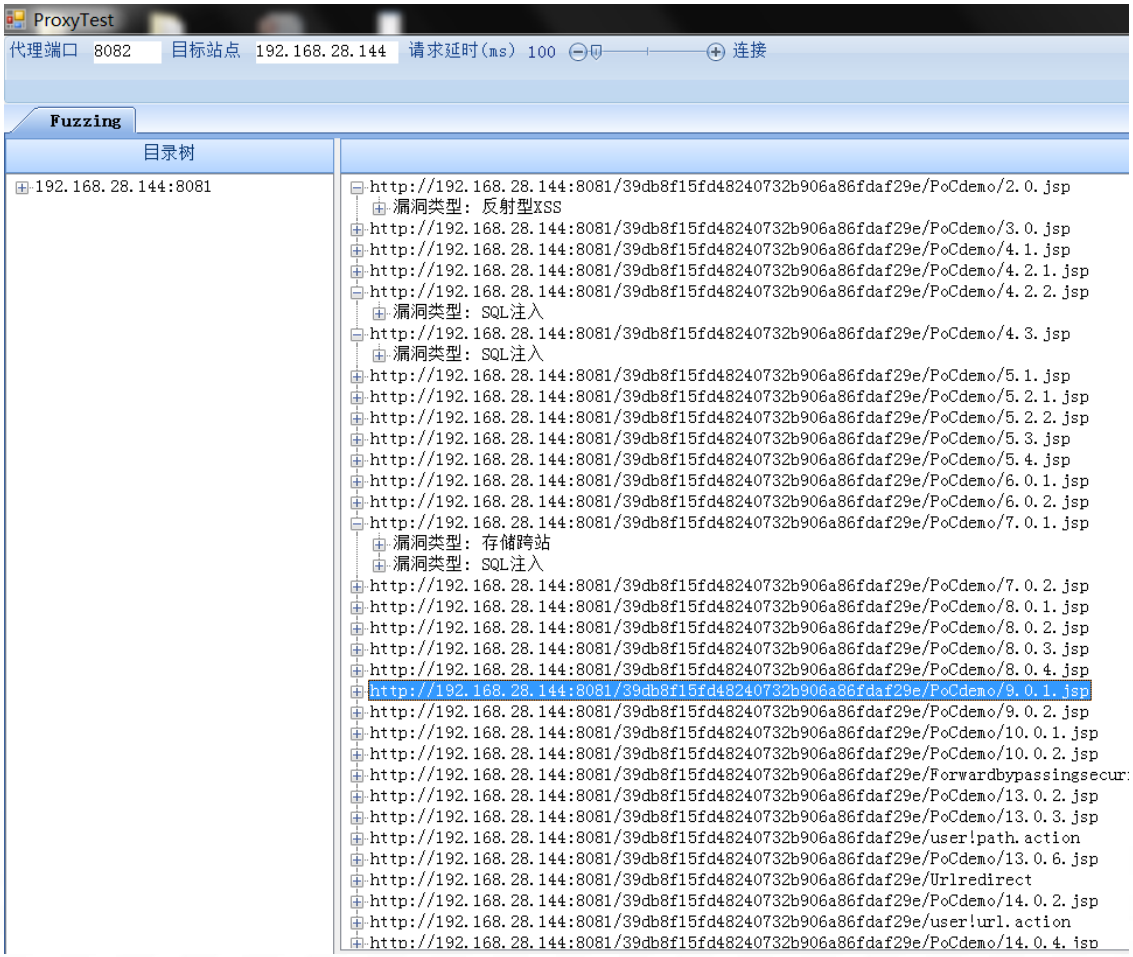
测试Demo

漏洞详情:(根据参数信息,安全测试人员帮助Coder快速定位漏洞)

漏洞详情	
项目名称:axis2　　版本名称:axis2-1.6.2	
漏洞类型:	反射型XSS
漏洞地址:	/axis2-admi[redacted]
漏洞参数名:	[redacted]
测试向量:	[redacted]
请求方式:	GET
堆栈信息:	at java.lang.Thread.dumpThreads(Thread.java:-2) at java.lang.Thread.getAllStackTraces(Thread.java:1530) at org.apache.jasper.runtime.JspWriterImpl.write(JspWriterImpl.java:-1) at org.apache.jasper.runtime.JspWriterImpl.print(JspWriterImpl.java:481) at org.apa[redacted] at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)

测试Demo

灰盒方式(被动模式):



检测分析

一.平均一个参数一个种漏洞只需要使用2个POC就可完成

二. 解决对于无回显漏洞（存储型XSS、SQL无回显注入等）的检测瓶颈

三. 可无害的探测各种漏洞，不产生垃圾数据，对业务流程不影响

四.被动探测的方式可结合测试人员进行安全测试， 解决复杂业务逻辑的爬虫问题

Thanks!