JAVA代码审计的一些Tips(附脚本)

cryin / 2017-11-22 14:39:00 / 浏览数 8863

JAVA代码审计的一些Tips(附脚本)

概述

本文重点介绍JAVA安全编码与代码审计基础知识，会以漏洞及安全编码示例的方式介绍JAVA代码中常见Web漏洞的形成及相应的修复方案，同时对一些常见的漏洞函数进行

XXE

介绍

XML文档结构包括XML声明、DTD文档类型定义（可选）、文档元素。文档类型定义(DTD)的作用是定义 XML 文档的合法构建模块。DTD 可以在 XML 文档内声明，也可以外部引用。

- 内部声明DTD:

  <!DOCTYPE 根元素 [元素声明]>

- 引用外部DTD:

  <!DOCTYPE 根元素 SYSTEM "文件名">

当允许引用外部实体时，恶意攻击者即可构造恶意内容访问服务器资源,如读取passwd文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE replace [
<!ENTITY test SYSTEM "file:///ect/passwd">]>
<msg>&test;</msg>
```

漏洞示例

此处以org.dom4j.io.SAXReader为例,仅展示部分代码片段：

```
String xmldata = request.getParameter("data");
SAXReader sax=new SAXReader();//■■■■SAXReader■■
Document document=sax.read(new ByteArrayInputStream(xmldata.getBytes()));//■■document■■,■■■■■■■■■■■■Exception■■■■
Element root=document.getRootElement();//■■■■■
List rowList = root.selectNodes("//msg");
Iterator<?> iter1 = rowList.iterator();
if (iter1.hasNext()) {
   Element beanNode = (Element) iter1.next();
   modelMap.put("success",true);
   modelMap.put("resp",beanNode.getTextTrim());
}
...
```

审计函数

XML解析一般在导入配置、数据传输接口等场景可能会用到，涉及到XML文件处理的场景可留意下XML解析器是否禁用外部实体，从而判断是否存在XXE。部分XML解析接

```
javax.xml.parsers.DocumentBuilder
javax.xml.stream.XMLStreamReader
org.jdom.input.SAXBuilder
org.jdom2.input.SAXBuilder
javax.xml.parsers.SAXParser
org.dom4j.io.SAXReader
org.xml.sax.XMLReader
javax.xml.transform.sax.SAXSource
javax.xml.transform.TransformerFactory
javax.xml.transform.sax.SAXTransformerFactory
javax.xml.validation.SchemaFactory
javax.xml.bind.Unmarshaller
javax.xml.xpath.XPathExpression
...
```

修复方案

使用XML解析器时需要设置其属性，禁止使用外部实体，以上例中SAXReader为例，安全的使用方式如下:

```
sax.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
sax.setFeature("http://xml.org/sax/features/external-general-entities", false);
sax.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
```

其它XML解析器的安全使用可参考OWASP XML External Entity (XXE) Prevention Cheat Sheet_Prevention_Cheat_Sheet#Java)

反序列化漏洞

介绍

序列化是让 Java 对象脱离 Java 运行环境的一种手段，可以有效的实现多平台之间的通信、对象持久化存储。

Java程序使用ObjectInputStream对象的readObject方法将反序列化数据转换为java对象。但当输入的反序列化的数据可被用户控制，那么攻击者即可通过构造恶意输入，

漏洞示例

漏洞代码示例如下：

```
......
//■■■■■,■■■■■
InputStream in=request.getInputStream();
ObjectInputStream ois = new ObjectInputStream(in);
//■■■■
ois.readObject();
ois.close();
```

上述代码中，程序读取输入流并将其反序列化为对象。此时可查看项目工程中是否引入可利用的commons-collections 3.1、commons-fileupload 1.3.1等第三方库，即可构造特定反序列化对象实现任意代码执行。相关三方库及利用工具可参考ysoserial、marshalsec。

审计函数

反序列化操作一般在导入模版文件、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,在代码审计时可重点关注一些反序列化操作函数并判断输

```
ObjectInputStream.readObject
ObjectInputStream.readUnshared
XMLDecoder.readObject
Yaml.load
XStream.fromXML
ObjectMapper.readValue
JSON.parseObject
...
```

修复方案

如果可以明确反序列化对象类的则可在反序列化时设置白名单，对于一些只提供接口的库则可使用黑名单设置不允许被反序列化类或者提供设置白名单的接口，可通过Hook

```
public class AntObjectInputStream extends ObjectInputStream{
   public AntObjectInputStream(InputStream inputStream)
           throws IOException {
       super(inputStream);
   }

   /**
    * ■■■■■■■SerialObject class
    */
   @Override
   protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException,
           ClassNotFoundException {
       if (!desc.getName().equals(SerialObject.class.getName())) {
           throw new InvalidClassException(
                   "Unauthorized deserialization attempt",
                   desc.getName());
       }
       return super.resolveClass(desc);
   }
}
```

也可以使用Apache Commons IO Serialization包中的ValidatingObjectInputStream类的accept方法来实现反序列化类白/黑名单控制，如果使用的是第三方库则升级到最新版本。更多修复方案可参考浅谈

SSRF

介绍

SSRF形成的原因大都是由于代码中提供了从其他服务器应用获取数据的功能但没有对目标地址做过滤与限制。比如从指定URL链接获取图片、下载等。

漏洞示例

此处以HttpURLConnection为例，示例代码片段如下:

```
String url = request.getParameter("picurl");
   StringBuffer response = new StringBuffer();

   URL pic = new URL(url);
   HttpURLConnection con = (HttpURLConnection) pic.openConnection();
   con.setRequestMethod("GET");
   con.setRequestProperty("User-Agent", "Mozilla/5.0");
   BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
   String inputLine;
   while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
   }
   in.close();
   modelMap.put("resp",response.toString());
   return "getimg.htm";
```

审计函数

程序中发起HTTP请求操作一般在获取远程图片、页面分享收藏等业务场景,在代码审计时可重点关注一些HTTP请求操作函数，如下：

```
HttpClient.execute
HttpClient.executeMethod
HttpURLConnection.connect
HttpURLConnection.getInputStream
URL.openStream
...
```

修复方案

- 使用白名单校验HTTP请求url地址
- 避免将请求响应及错误信息返回给用户
- 禁用不需要的协议及限制请求端口,仅仅允许http和https请求等

SQLi

介绍

注入攻击的本质，是程序把用户输入的数据当做代码执行。这里有两个关键条件，第一是用户能够控制输入；第二是用户输入的数据被拼接到要执行的代码中从而被执行。s

漏洞示例

此处以Mybatis框架为例，示例sql片段如下:

```
select * from books where id= ${id}
```

对于Mybatis框架下SQL注入漏洞的审计可参考Mybatis框架下SQL注入漏洞面面观

修复方案

Mybatis框架SQL语句安全写法应使用#{},避免使用动态拼接形式\${}，ibatis则使用#变量#。安全写法如下:

```
select * from books where id= #{id}
```

文件上传漏洞

介绍

文件上传过程中，通常因为未校验上传文件后缀类型，导致用户可上传jsp等一些webshell文件。代码审计时可重点关注对上传文件类型是否有足够安全的校验，以及是否限

漏洞示例

此处以MultipartFile为例，示例代码片段如下:

```
public String handleFileUpload(MultipartFile file){
       String fileName = file.getOriginalFilename();
       if (fileName==null) {
           return "file is error";
       }
       String filePath = "/static/images/uploads/"+fileName;
       if (!file.isEmpty()) {
           try {
               byte[] bytes = file.getBytes();
               BufferedOutputStream stream =
                       new BufferedOutputStream(new FileOutputStream(new File(filePath)));
               stream.write(bytes);
               stream.close();
               return "OK";
           } catch (Exception e) {
               return e.getMessage();
           }
       } else {
           return "You failed to upload " + file.getOriginalFilename() + " because the file was empty.";
       }
   }
```

审计函数

java程序中涉及到文件上传的函数，比如：

```
MultipartFile
...
```

修复方案

- 使用白名单校验上传文件类型、大小限制

Autobinding

介绍

Autobinding-自动绑定漏洞，根据不同语言/框架，该漏洞有几个不同的叫法，如下：

- Mass Assignment: Ruby on Rails, NodeJS
- Autobinding: Spring MVC, ASP.NET MVC
- Object injection: PHP(对象注入、反序列化漏洞)

软件框架有时允许开发人员自动将HTTP请求参数绑定到程序代码变量或对象中，从而使开发人员更容易地使用该框架。这里攻击者就可以利用这种方法通过构造http请求，

漏洞示例

示例代码以ZeroNights-HackQuest-2016的demo为例，把示例中的justiceleague程序运行起来，可以看到这个应用菜单栏有about，reg，Sign up，Forgot password这4个页面组成。我们关注的点是密码找回功能，即怎么样绕过安全问题验证并找回密码。

1）首先看reset方法，把不影响代码逻辑的删掉。这样更简洁易懂：

```
@Controller
@SessionAttributes("user")
public class ResetPasswordController {

private UserService userService;
...
@RequestMapping(value = "/reset", method = RequestMethod.POST)
public String resetHandler(@RequestParam String username, Model model) {
       User user = userService.findByName(username);
       if (user == null) {
           return "reset";
       }
       model.addAttribute("user", user);
       return "redirect: resetQuestion";
   }
```

这里从参数获取username并检查有没有这个用户，如果有则把这个user对象放到Model中。因为这个Controller使用了@SessionAttributes("user")，所以同时也会自动把

2）resetQuestion密码找回安全问题校验页面有resetViewQuestionHandler这个方法展现

```
@RequestMapping(value = "/resetQuestion", method = RequestMethod.GET)
   public String resetViewQuestionHandler(@ModelAttribute User user) {
       logger.info("Welcome resetQuestion ! " + user);
       return "resetQuestion";
   }
```

这里使用了@ModelAttribute User
user，实际上这里是从session中获取user对象。但存在问题是如果在请求中添加user对象的成员变量时则会更改user对象对应成员的值。
所以当我们给resetQuestionHandler发送GET请求的时候可以添加"answer=hehe"参数，这样就可以给session中的对象赋值，将原本密码找回的安全问题答案修改成"heh

审计函数

这种漏洞一般在比较多步骤的流程中出现，比如转账、找密等场景，也可重点留意几个注解如下：

```
@SessionAttributes
@ModelAttribute
...
```

更多信息可参考Spring MVC Autobinding漏洞实例初窥

修复方案

Spring MVC中可以使用@InitBinder注解，通过WebDataBinder的方法setAllowedFields、setDisallowedFields设置允许或不允许绑定的参数。

URL重定向

介绍

由于Web站点有时需要根据不同的逻辑将用户引向到不同的页面，如典型的登录接口就经常需要在认证成功之后将用户引导到登录之前的页面，整个过程中如果实现不好就可

漏洞示例

此处以Servlet的redirect 方式为例，示例代码片段如下:

```
String site = request.getParameter("url");
   if(!site.isEmpty()){
       response.sendRedirect(site);
   }
```

审计函数

java程序中URL重定向的方法均可留意是否对跳转地址进行校验、重定向函数如下：

```
sendRedirect
setHeader
forward
...
```

修复方案

- 使用白名单校验重定向的url地址
- 给用户展示安全风险提示，并由用户再次确认是否跳转

CSRF

介绍

跨站请求伪造（Cross-Site Request
Forgery，CSRF）是一种使已登录用户在不知情的情况下执行某种动作的攻击。因为攻击者看不到伪造请求的响应结果，所以CSRF攻击主要用来执行动作，而非窃取用户数

漏洞示例

由于开发人员对CSRF的了解不足，错把"经过认证的浏览器发起的请求"当成"经过认证的用户发起的请求"，当已认证的用户点击攻击者构造的恶意链接后就"被"执行了相应的

```
GET http://blog.com/article/delete.jsp?id=102
```

当攻击者诱导用户点击下面的链接时，如果该用户登录博客网站的凭证尚未过期，那么他便在不知情的情况下删除了id为102的文章，简单的身份验证只能保证请求发自某个

漏洞审计

此类漏洞一般都会在框架中解决修复，所以在审计csrf漏洞时。首先要熟悉框架对CSRF的防护方案，一般审计时可查看增删改请求重是否有token、formtoken等关键字以及

修复方案

- Referer校验，对HTTP请求的Referer校验，如果请求Referer的地址不在允许的列表中，则拦截请求。
- Token校验，服务端生成随机token，并保存在本次会话cookie中，用户发起请求时附带token参数，服务端对该随机数进行校验。如果不正确则认为该请求为伪造请求
- Formtoken校验，Formtoken校验本身也是Token校验，只是在本次表单请求有效。
- 对于高安全性操作则可使用验证码、短信、密码等二次校验措施
- 增删改请求使用POST请求

命令执行

介绍

由于业务需求，程序有可能要执行系统命令的功能，但如果执行的命令用户可控，业务上有没有做好限制，就可能出现命令执行漏洞。

漏洞示例

此处以getRuntime为例，示例代码片段如下:

```
String cmd = request.getParameter("cmd");
   Runtime.getRuntime().exec(cmd);
```

审计函数

这种漏洞原理上很简单，重点是找到执行系统命令的函数，看命令是否可控。在一些特殊的业务场景是能判断出是否存在此类功能，这里举个典型的实例场景,有的程序功能需

java程序中执行系统命令的函数如下：

```
Runtime.exec
ProcessBuilder.start
GroovyShell.evaluate
...
```

修复方案

- 避免命令用户可控
- 如需用户输入参数，则对用户输入做严格校验，如&&、|、;等

权限控制

介绍

越权漏洞可以分为水平、垂直越权两种,程序在处理用户请求时未对用户的权限进行校验，使的用户可访问、操作其他相同角色用户的数据，这种情况是水平越权；如果低权限

漏洞示例

```
@RequestMapping(value="/getUserInfo",method = RequestMethod.GET)
   public String getUserInfo(Model model, HttpServletRequest request) throws IOException {
       String userid = request.getParameter("userid");
       if(!userid.isEmpty()){
           String info=userModel.getuserInfoByid(userid);
           return info;
       }
       return "";
   }
```

审计函数

水平、垂直越权不需关注特定函数，只要在处理用户操作请求时查看是否有对当前登陆用户权限做校验从而确定是否存在漏洞

修复方案

获取当前登陆用户并校验该用户是否具有当前操作权限，并校验请求操作数据是否属于当前登陆用户，当前登陆用户标识不能从用户可控的请求参数中获取。

批量请求

介绍

业务中经常会有使用到发送短信校验码、短信通知、邮件通知等一些功能，这类请求如果不做任何限制，恶意攻击者可能进行批量恶意请求轰炸，大量短信、邮件等通知对正

除了短信、邮件轰炸等，还有一种情况也需要注意，程序中可能存在很多接口，用来查询账号是否存在、账号名与手机或邮箱、姓名等的匹配关系，这类请求如不做限制也会

漏洞示例

```
@RequestMapping(value="/ifUserExit",method = RequestMethod.GET)
   public String ifUserExit(Model model, HttpServletRequest request) throws IOException {
       String phone = request.getParameter("phone");
       if(! phone.isEmpty()){
           boolean ifex=userModel.ifuserExitByPhone(phone);
           if (!ifex)
               return "■■■■■■";
       }
       return "■■■■■■■";
   }
```

修复方案

- 对同一个用户发起这类请求的频率、每小时及每天发送量在服务端做限制，不可在前端实现限制

第三方组件安全

介绍

这个比较好理解，诸如Struts2、不安全的编辑控件、XML解析器以及可被其它漏洞利用的如commons-collections:3.1等第三方组件，这个可以在程序pom文件中查看是否

修复方案

- 使用最新或安全版本的第三方组件

危险函数自动化搜索脚本javaid.py

审计一个工程一般是需要通篇阅读代码，但是有的时候也需要简单粗暴的方法，就是关注一些可能产生漏洞的危险函数，这里分享一个自己编写的自动化脚本JavaID。通过正

总结

除了上述相关的漏洞，在代码审计的时候有时会遇到一些特别的漏洞，比如开发为了测试方便关闭掉了一些安全校验函数、甚至未彻底清除的一些预留后门及测试管理接口等

参考

- https://github.com/Cryin/JavaID

上一篇：企业安全应急响应 下一篇：JAVA中常见数据库操作API

1. 10 条回复

cover 2017-11-22 15:46:59

整理挺好的资料，算是入门吧

0 回复Ta

hades 2017-11-22 16:22:29

@cover 欢迎━(` ロ´)ノイ !补充

0 回复Ta


theone 2017-11-22 17:41:23

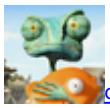
赞一个

0 回复Ta


xxlegend 2017-11-23 11:00:54

赞一个

0 回复Ta


orich1 2017-12-15 17:53:15

学习学习

0 回复Ta


applychen 2017-12-25 16:13:24

RequestDispatcher.forward可读取文件

0 回复Ta


reborn 2018-02-12 11:18:50

总结得很好，给一个赞。

0 回复Ta


71428****@qq.com 2018-09-28 17:06:01

csrf、权限验证、批量请求偏业务不好判断，其他可以参考findsecbugs规则来跑一跑

0 回复Ta


niexinming 2018-11-11 01:37:06

马克

0 回复Ta


Lr0me 2019-02-13 09:25:35

刚开始代码审计，很有帮助

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板