

# 浅析 Cobalt Strike Team Server 扫描

## 0x01 前言

Cobalt Strike 是由 Strategic Cyber 公司开发的一款商业化渗透测试工具。该软件具有简单易用、可扩展性高等优点，并且具备团队协作等特点，因此被广大黑客、白帽子和安全研究人员等大量装备使用。网络空间测绘就是利用扫描发掘互联网中一切可发掘的资产和目标。Cobalt Strike 的发掘，是 360 Quake 团队一直以来的核心目标之一。

在本文中我们将站在蓝方角度思考，通过研究 Cobalt Strike 客户端与服务端交互的代码逻辑，来发掘出 Cobalt Strike Team Server 特征，并且进行进一步探测与分析。

## 0x02 逻辑分析

Cobalt Strike 的是 C/S (Client-Server) 架构，有一个客户端与 Team Server 进行通信。首先，简单看下代码。在完成 Swing 组件的加载后，用户输入用户名、密码等信息后，点击 Connect 按钮，触发 aggressor.dialogs 下的 Connet 类的 dialogAction 方法。如图所示，代码逻辑会将密码传入一个 SecureSocket 实例的 authenticate 方法中。

```
try {  
    //----- 客户端代码-----  
    SecureSocket var7 = new SecureSocket(var4, Integer.parseInt(var5), armitageTrustListener: this);  
    var7.authenticate(var6); // 进行密码验证  
    TeamSocket var8 = new TeamSocket(var7.getSocket());  
    this.tqueue = new TeamQueue(var8);  
    this.tqueue.call("aggressor.authenticate", CommonUtils.args(var3, var6, Aggressor.VERSION), this);  
}
```

然后创建数据输出流实例，将\x00\x00\xbe\xef+ 密码长度 + 密码 + 填充字符等数据传给 Team Server，如果返回 51966 (\x00\x00\xca\xfe) 则证明密码正确。

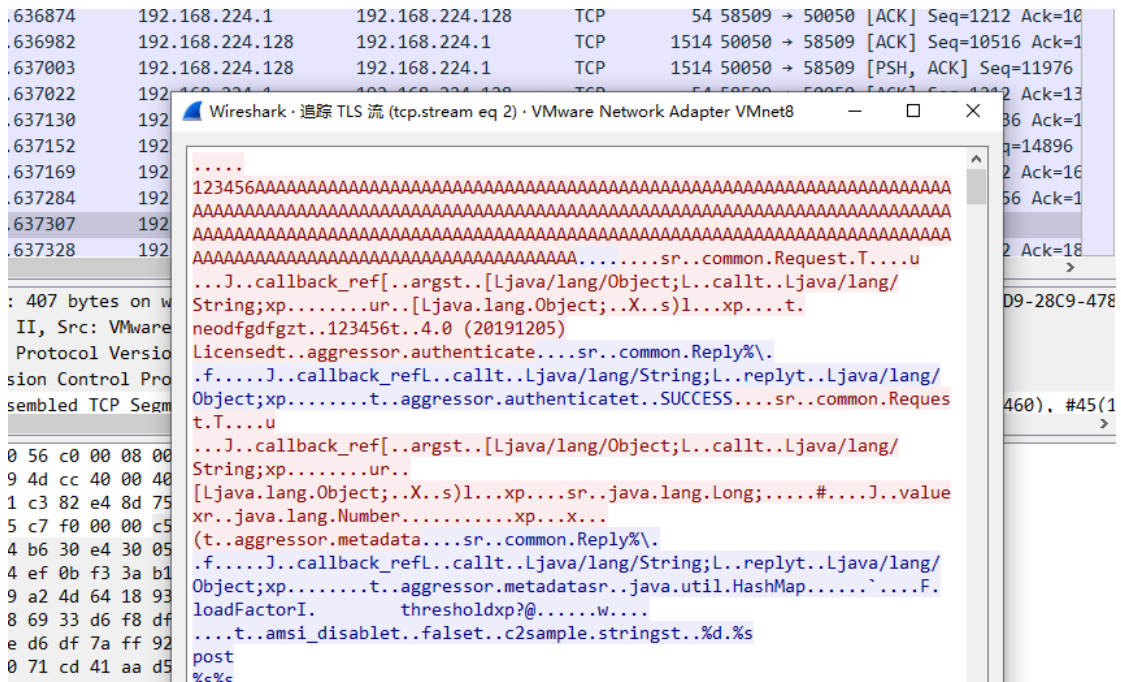
```
var3.writeInt(48879); // 写入\x00\x00\xbe\xef  
var3.writeByte(var1.length()); // 写入密码长度  
// 填充密码  
int var4;  
for(var4 = 0; var4 < var1.length(); ++var4) {  
    var3.writeByte((byte)var1.charAt(var4));  
}  
//----- 客户端代码-----  
// 填充字符串  
for(var4 = var1.length(); var4 < 256; ++var4) {  
    var3.writeByte(65);  
}  
  
var3.flush();  
var4 = var2.readInt();  
if (var4 != 51966) { // 验证正确 返回\x00\x00\xca\xfe  
    // 验证失败 authentication failure!  
    throw new RuntimeException("authentication failure!");  
}
```

查看服务端的代码认证逻辑(ssl.SecureServerSocket.java)，发现在密码正确

后返回 51966 (\x00\x00\xca\xfe)，密码错误返回 0 (\x00\x00\x00\x00)。

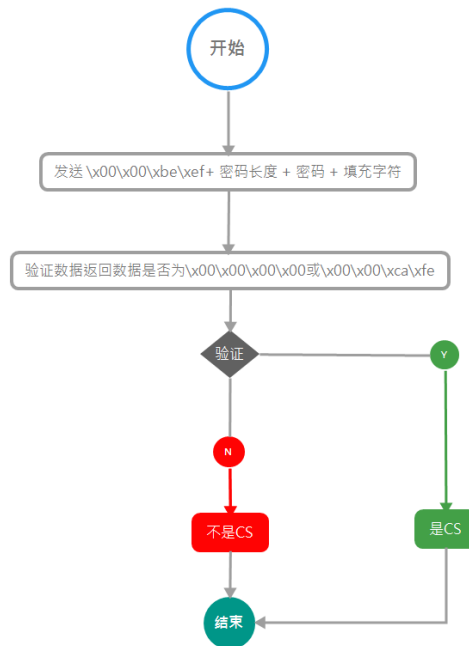
```
// -----服务端代码-----  
if (var8.toString().equals(var2)) {  
    var5.writeInt(51966); // \x00\x00\xca\xfe  
    return true;  
} else {  
    var5.writeInt(0); // \x00\x00\x00\x00  
    CommonUtils.print_error("rejected client from " + var3 + ": invalid password");  
    return false;  
}
```

通过 Wireshark 抓包如图所示，可以看到客户端发送的密码及填充字符串。



Wireshark 抓包图显示了网络流量，其中包含客户端发送的密码及填充字符串。图中展示了数据包列表、数据包详情以及数据包字节内容。数据包详情部分显示了 TLS 流的解密过程，包括解密后的明文内容。

通过以上代码逻辑，可以找到一个识别 Cobalt Strike Team Server 的方法，流程图及部分代码如下图所示：



```

res = authenticate(password="123456", host="192.168.1.1", port=50050)
if res == bytearray(b'\x00\x00\x00\x00'):
    print("检测到Cobalt Strike")
elif res == bytearray(b"\x00\x00\xca\xfe"):
    print("检测到Cobalt Strike, 密码为123456")
else:
    print("目标不是Cobalt Strike")
  
```

为了利用该特征获取更多资产，我们希望把 Cobalt Strike 中被控制的 IP 给提取出来，就是 Cobalt Strike 登陆后的 Session Table 提取出来，如图所示。

Cobalt Strike					
Cobalt Strike View Attacks Reporting Help					
external	internal	listener	user	computer	note
192.168.224.133	192.168.224.133	aaaaaa	Administrator *	WIN-RAGK00Q0AKD	
Session Table					

同样先看下代码，在通过密码验证后，会创建一个 TeamQueue 的实例。

```

try {
    //----- 客户端代码-----
    SecureSocket var7 = new SecureSocket(var4, Integer.parseInt(var5), armitageTrustListener: this);
    var7.authenticate(var6); // 进行密码验证
    TeamSocket var8 = new TeamSocket(var7.getSocket());
    this.tqueue = new TeamQueue(var8);
    this.tqueue.call("aggressor.authenticate", CommonUtils.args(var3, var6, Aggressor.VERSION), this);
  }
  
```

通过查看该类的构造方法，发现在创建该实例的时候，同时启动了两个线程来对 TeamQueue 中请求和响应进行监控。

```
public TeamQueue(TeamSocket var1) {  
    this.socket = var1;  
    this.reader = new TeamReader();  
    this.writer = new TeamWriter();  
    (new Thread(this.writer, name: "TeamQueue Writer")).start();  
    (new Thread(this.reader, name: "TeamQueue Reader")).start();  
}
```

如图所示，TeamQueue 类的 call 方法中会根据传入的参数实例化一个 Request 对象，并添加进队列，之后 TeamQueue Writer 线程从队列获取请求对象，使用 socket 进行发送。

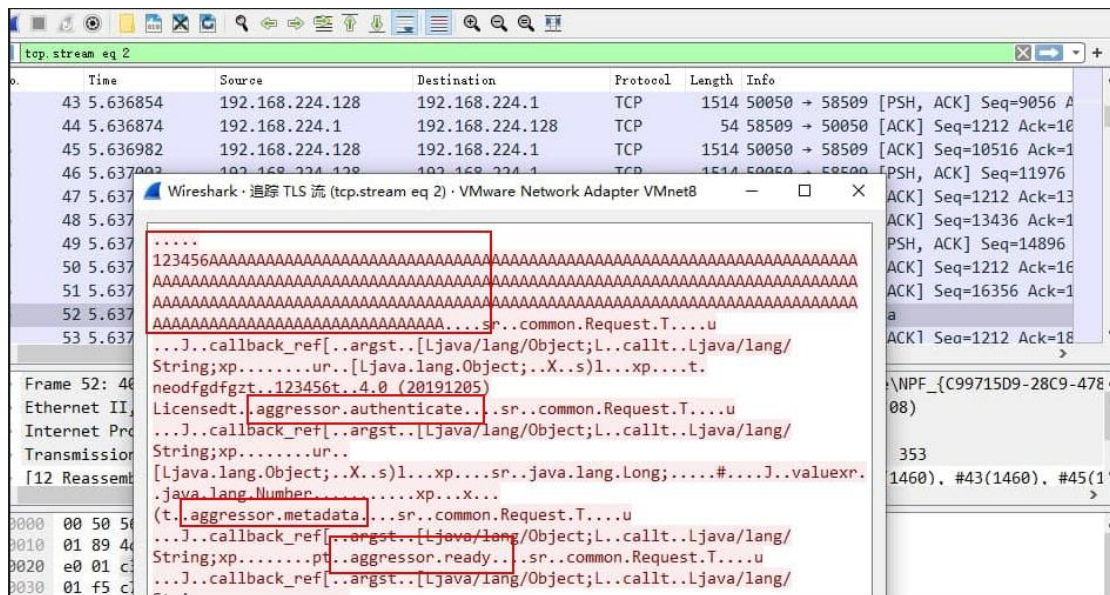
```
public void call(String var1, Object[] var2, Callback var3) {  
    if (var3 == null) {  
        Request var4 = new Request(var1, var2, 0L);  
        this.writer.addRequest(var4);  
    } else {  
        synchronized(this.callbacks) {  
            ++this.reqno;  
            this.callbacks.put(new Long(this.reqno), var3);  
            Request var5 = new Request(var1, var2, this.reqno);  
            this.writer.addRequest(var5);  
        }  
    }  
}  
  
public void run() {  
    while(TeamQueue.this.socket.isConnected()) {  
        Request var1 = this.grabRequest();  
        System.out.println(var1);  
        if (var1 != null) {  
            TeamQueue.this.socket.writeObject(var1);  
            Thread.yield();  
        } else {  
            try {  
                Thread.sleep(25L);  
            } catch (InterruptedException var3) {  
                MudgeSanity.logException("teamwriter sleep", var3, false);  
            }  
        }  
    }  
}
```

TeamQueue Reader 时刻监听着来自 Team Server 的响应。

```
public void run() {
    while(true) {
        try {
            if (TeamQueue.this.socket.isConnected()) {
                Reply var1 = (Reply)TeamQueue.this.socket.readObject();
                TeamQueue.this.processRead(var1);
                Thread.yield();
                continue;
            }
        } catch (Exception var2) {
            MudgeSanity.logException("team reader", var2, false);
            TeamQueue.this.close();
        }

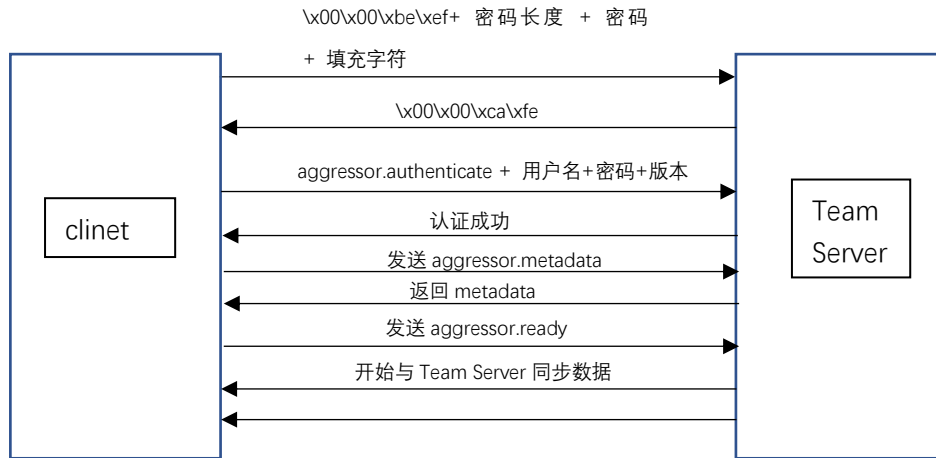
        return;
    }
}
```

以上就是 Cobalt Strike 客户端发包和接收响应的大致逻辑。在创建 TeamQueue 实例后开始调用 call 方法来发送不同阶段的请求。我们进一步抓包分析。如图所示



结合抓包和查看代码发现客户端与服务端交互的流程如下图所示。





在客户端发送 aggressor.ready 请求后，表示一切准备就绪，开始和服务端进行数据同步，这其中就有 session table 的数据。根据以上的逻辑，然后手动代码实现这几个请求，就能够在识别出 Cobalt Strike 后进一步爆破密码，在爆破出密码后提取出目标的受控 IP。如图所示。

```
[kali@kali:~]$- [02:29:02 AM]
Picked up JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Cobalt Strike 版本为: 4.0 (20191205) Licensed
SUCCESS
Sync To Team Server. 同步中...
受控 IP
{note=, charset=windows-1252, internal=172.16.1.103, alive=true, session=beacon, listener=mycompute, pid=2096, lastf=0ms, computer=WIN-QIUREVVK5FL, host=172.16.1.103, is64=1, id=559928342, process=5ff2fced4d398072b3666cf7ede34a6, ver=6.1, last=0, os=Windows, barch=x64, opened=1607398814887, phint=0, external=36.99.136.129, port=0, pbid=, arch=x64, user=Administrator *, accent=)
{note=, charset=mswin-936, internal=192.168.243.1, alive=true, session=beacon, listener=mycompute, pid=8492, lastf=0ms, computer=DESKTOP-A9J3PLT, host=192.168.243.1, is64=1, id=151667470, process=notepad.exe, ver=10.0, last=0, os=Windows, barch=x86, opened=1603978297896, phint=0, external=124.77.36.97, port=0, pbid=, arch=x86, user=lingf, accent=)
{note=, charset=windows-1252, internal=10.130.225.183, alive=true, session=beacon, listener=mycompute, pid=1912, lastf=0ms, computer=WIN-500TMIODGJX, host=10.130.225.183, is64=1, id=1089619838, process=calc.exe, ver=6.1, last=0, os=Windows, barch=x64, opened=1605179631377, phint=0, external=95.26.55.35, port=0, pbid=, arch=x64, user=Dm421X *, accent=)
{note=, charset=mswin-936, internal=172.18.9.177, alive=true, session=beacon, listener=mycompute, pid=1676, lastf=0ms, computer=DESKTOP-PJIO8V5, host=172.18.9.177, is64=1, id=187839728, process=notepad.exe, ver=10.0, last=0, os=Windows, barch=x86, opened=1605015024652, phint=0, external=124.77.36.97, port=0, pbid=, arch=x86, user=Boolu, accent=)
{note=, charset=windows-1252, internal=10.13.206.168, alive=true, session=beacon, listener=mycompute, pid=2652, lastf=0ms, computer=WIN-FFW5EAPRJR V, host=10.13.206.168, is64=1, id=1958367154, process=txjujuvsky.exe, ver=6.1, last=0, os=Windows, barch=x64, opened=1605178913855, phint=0, external=89.288.29.205, port=0, pbid=, arch=x64, user=Jni5d *, accent=)
{note=, charset=windows-1252, internal=192.168.127.10, alive=true, session=beacon, listener=mycompute, pid=888, lastf=0ms, computer=PC-4A095E27CB, host=192.168.127.10, is64=1, id=282942592, process=calc.exe, ver=6.1, last=0, os=Windows, barch=x64, opened=1605195663951, phint=0, external=72.12.194.90, port=0, pbid=, arch=x64, user=STRAZNIICA.GRUBUTT *, accent=)
{note=, charset=mswin-936, internal=192.168.142.33, alive=true, session=beacon, listener=mycompute, pid=10496, lastf=0ms, computer=DESKTOP-PJIO8V5, host=192.168.142.33, is64=1, id=140704656, process=notepad.exe, ver=10.0, last=0, os=Windows, barch=x86, opened=1603978297896, phint=0, external=124.77.36.97, port=0, pbid=, arch=x86, user=lingf, accent=)}
```

最终的扫描流程图见附录。

## 0x03 扫描结果分析

网络空间测绘始于扫描，不止于扫描。在找到该特征后，我们开始在互联网中进行挖掘。

使用搜索语句：`response:"\\x00\\x00\\xca\\xfe" AND port: "50050"`，可以找到存在弱口令的 Cobalt Strike，我们只对部分主机进行了受控 IP 的提取，这步骤稍微敏感，大家可以自行操作。

服务数据 response: "\x00\x00\xca\xfe" AND port: "50050"

共 193 条 搜索结果 (68个独立IP) , 用时1.222秒

近一年 重新扫描 导出数据 排除蜜罐 忽略缓存 添加收藏 IP地址列表

检索结果 数据统计 聚合分析

端口 50050 193

产品 暂无数据

服务协议 cobaltstrike/ssl 193

运营商

103.210.238.24 2021-04-13 07:24:10

中国 - 香港 - 香港 高墙

50050 tcp ipv4

ASN: 133115  
组织: hongkong kwaifong inform...  
运营商: 香港葵芳集团有限公司

服务协议: cobaltstrike/ssl

端口响应

\x00\x00\xca\xfe

"password": "11223344"  
"info": "Cobalt Strike 版本为: 4.0 (20191205) Licensed SUCCESS  
Sync To Team Server. 同步中. . .  
监听IP  
(note=, charset=windows-1252, internal=10.66.152.115, alive=true, session=beacon, listener=必驻, pid=14848, lastf=0ms, computer=SERGEY-37826572E7, host=10.66.152.115, is64=1, id=1038783873, process=kbdqdficz.exe, ver=6.1, last=0, os=Windows, barch=x86, opened=1615252872611, phint=0, external=95.26.153.138, port=0, pbid=, arch=x86, user=komarov \*, \_accent=)

103.210.238.24

2021-04-13 07:24:10

50050

cobaltstrike/ssl

\x00\x00\xca\xfe 认证成功返回包

"password": "11223344"  
"info": "Cobalt Strike 版本为: 4.0 (20191205) Licensed SUCCESS  
Sync To Team Server. 同步中. . .  
监听IP  
(note=, charset=windows-1252, internal=10.66.152.115, alive=true, session=beacon, listener=必驻, pid=14848, lastf=0ms, computer=SERGEY-37826572E7, host=10.66.152.115, is64=1, id=1038783873, process=kbdqdficz.exe, ver=6.1, last=0, os=Windows, barch=x86, opened=1615252872611, phint=0, external=95.26.153.138, port=0, pbid=, arch=x86, user=komarov \*, \_accent=)

获取Team Server 版本

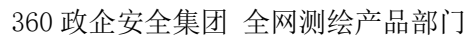
内网IP

进程

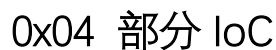
Listener

外网IP

随机登录一台如图:



通过搜索语法 `response:"\\x00\\x00\\x00\\x00" AND port: "50050" AND service: "http/ssl"` 可以找到没有爆破出密码的 Cobalt Strike，可以看到共有 5173 条扫描记录，1049 个 IP。



IP地址	端口号	证书序列号	证书主体通用名	证书主体所属国家	证书主体所属组织
198.52.125.217	50050	1044643756	Major Cobalt Strike	Earth	cobaltstrike
195.154.250.27	50050	149257128014384946781589745993101702086	orderer.users.mediso.t.com	US	
193.57.40.222	50050	1497275127	Major Cobalt Strike	Earth	cobaltstrike
193.112.10.125	50050	2023502547	Major Cobalt Strike	Earth	cobaltstrike
188.119.113.24	50050	1514727070	Major Cobalt Strike	Earth	cobaltstrike
185.207.152.86	50050	658336750	Major Cobalt Strike	Earth	cobaltstrike
185.118.166.66	50050	1503727134	use.fortawesome.com	US	Microsoft Corporation
185.161.208.234	50050	1514727070	Major Cobalt Strike	Earth	cobaltstrike
175.24.125.242	50050	924895928	Major Cobalt Strike	Earth	cobaltstrike
173.234.155.145	50050	924895928	Major Cobalt Strike	Earth	cobaltstrike



## 0x05 结论

网络空间测绘，始于资产，但不止于资产。

我们认为，主动测绘数据将会与终端行为样本数据、网络流量通信数据一样，是未来网络安全大数据&&威胁情报数据的重要源头。主动测绘数据和基于测绘数据分析后形成的知识将能够极大补充我们的视野，从而开拓出更多的攻击面和领域。更多网络空间测绘领域研究内容，敬请期待~

## 0x06 附录

