可信网络 安全世界
RELIABLE NETWORK
SAFE WORLD

APT组织研究报告

Flame2.0 样本分析

2018年5月

听风者实验室

目　录

一、概述

2012 年，Flame 被曝光后，攻击者通过下发“自杀”命令进行清理，并擦除了他们用来与之通信的命令和控制服务器，从此销声匿迹。近期，Chronicle 安全研究人员表示他们已经发现了 2014 年出现的新版火焰版本，并且可能在 2016 年之前一直保持活跃状态。在此我们对已有的样本进行分析，主要分两部分（32 位和 64 位样本各 4 个），为 Flame2.0 的部分功能模块。

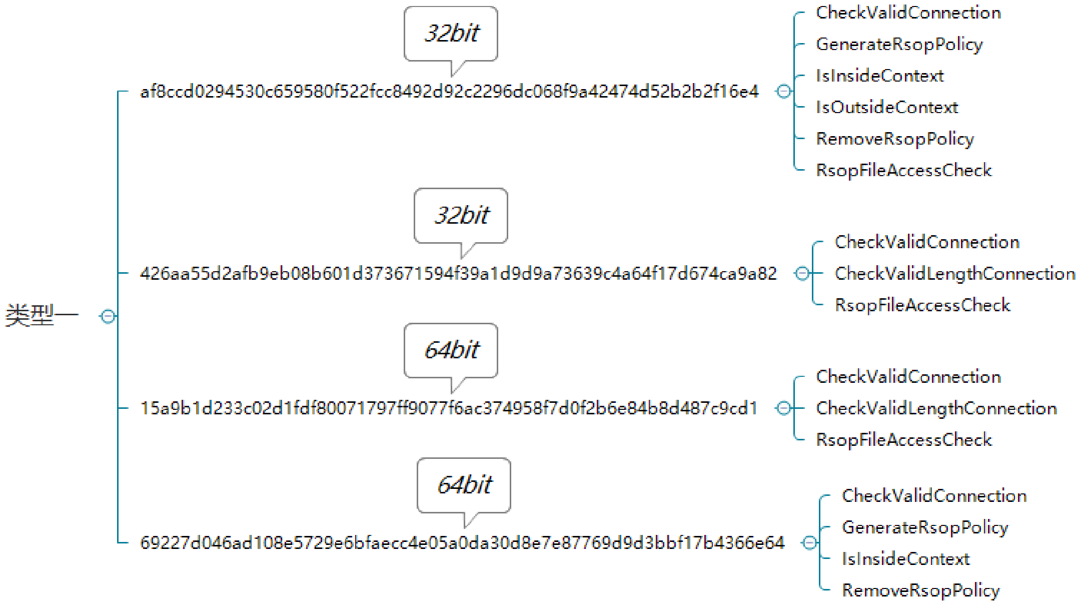
二、样本信息

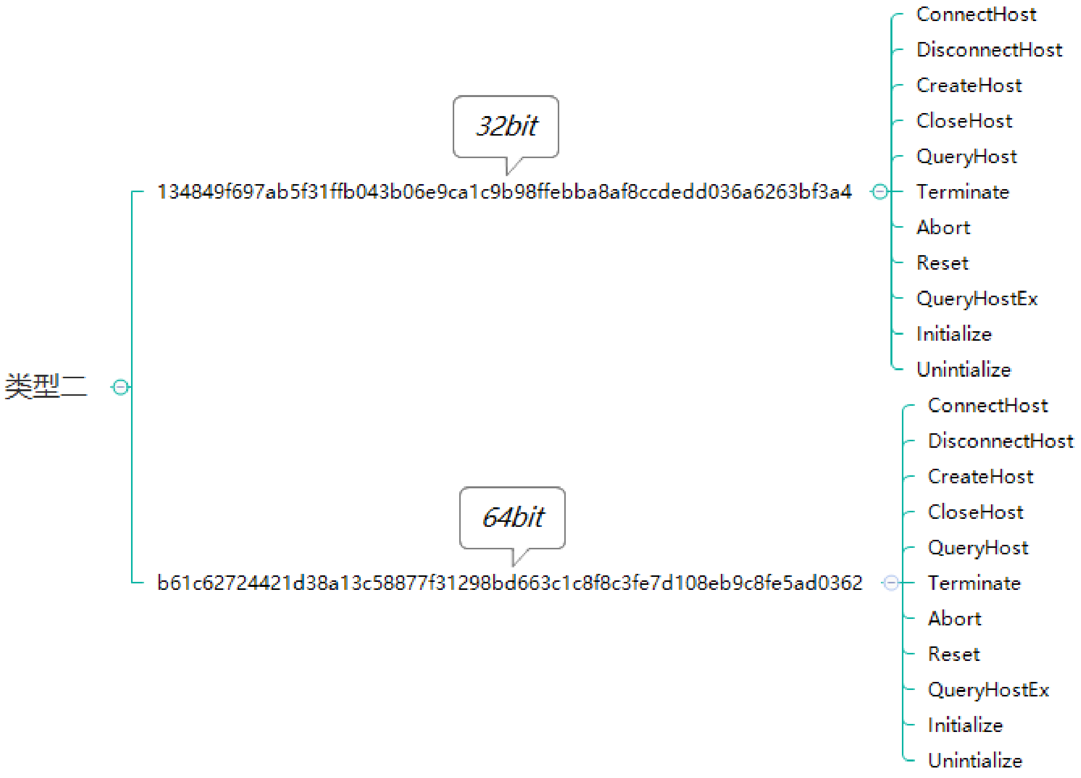
当前获取到的样本信息如下：

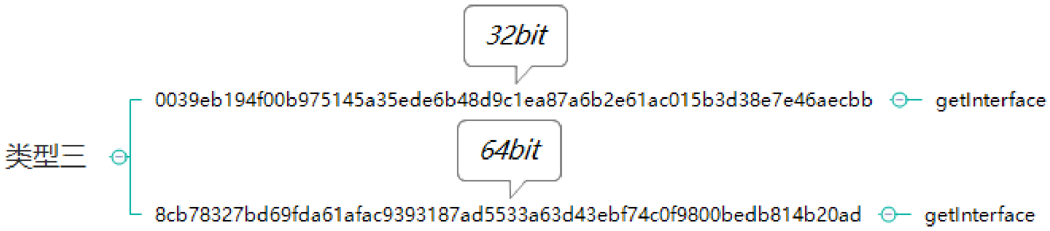
样本 1 文件名称: af8ccd0294530c659580f522fcc8492d92c2296dc068f9a42474d52b2b2f16e4 原始文件名: sensrsvr.dll 文件大小: 786 KB (804,864 字节) 编译时间: 2011-06-01 09:09:29 MD5: 98303a3a424c407a3e27ab818066811c SHA1: 5ab8b1ac11789606333ff94066cae6048a335ac5 SHA256: af8ccd0294530c659580f522fcc8492d92c2296dc068f9a42474d52b2b2f16e4
样本 2 文件名称: 426aa55d2afb9eb08b601d373671594f39a1d9d9a73639c4a64f17d674ca9a82 原始文件名: sensrsvcs.dll 文件大小: 783 KB (801,792 字节) 编译时间: 2006-06-08 07:23:59 MD5: 7ab1c0c5e7d1ed834bccdfcafb5b07f2 SHA1: 21d3d7c33f63def5aed98d54dac5de218c49a35f SHA256: 426aa55d2afb9eb08b601d373671594f39a1d9d9a73639c4a64f17d674ca9a82
样本 3 文件名称: 15a9b1d233c02d1fdf80071797ff9077f6ac374958f7d0f2b6e84b8d487c9cd1 原始文件名: sensrsvcs.dll 文件大小: 791 KB (809,984 字节) 编译时间: 2006-06-17 10:25:13 MD5: 2a2614756387176845187a7de247a98a SHA1: ef2f8fca2a010f49ab4080a6439651320b95e44f SHA256: 15a9b1d233c02d1fdf80071797ff9077f6ac374958f7d0f2b6e84b8d487c9cd1
样本 4 文件名称: 69227d046ad108e5729e6bfaecc4e05a0da30d8e7e87769d9d3bbf17b4366e64 原始文件名: sensrsvr.dll 文件大小: 798 KB (817,152 字节) 编译时间: 2010-05-26 05:50:49

MD5：2529ecdd21ad9854d52ab737306bee59 SHA1：b144c68108d9a9208accb562b141d8b8a15550d7 SHA256：69227d046ad108e5729e6bfaecc4e05a0da30d8e7e87769d9d3bbf17b4366e64
样本 5 文件名称：134849f697ab5f31ffb043b06e9ca1c9b98ffebba8af8ccdedd036a6263bf3a4 原始文件名：wmihost.dll 文件大小：849 KB（869,376 字节） 编译时间：2011-04-24 16:40:33 MD5：294be9caf93116430f7a8007a202e9fd SHA1：45f348b46a745c1f45e4eac0185d73cc4e65edc3 SHA256：134849f697ab5f31ffb043b06e9ca1c9b98ffebba8af8ccdedd036a6263bf3a4
样本 6 文件名称：b61c62724421d38a13c58877f31298bd663c1c8f8c3fe7d108eb9c8fe5ad0362 原始文件名：wmihost64.dll 文件大小：0.97 MB（1,025,024 字节） 编译时间：2011-02-07 14:36:09 MD5：6ce0a12d7461f3267af7fa835a0b5677 SHA1：941195b52f5ea4eb60027c3aeb67cd72e95f4c8e SHA256：b61c62724421d38a13c58877f31298bd663c1c8f8c3fe7d108eb9c8fe5ad0362
样本 7 文件名称：0039eb194f00b975145a35ede6b48d9c1ea87a6b2e61ac015b3d38e7e46aecbb 原始文件名：wmisvcs64.dll 文件大小：940 KB（962,560 字节） 编译时间：2006-08-02 14:36:56 MD5：15a0b9948d60e6bc6f60d7226caa923f SHA1：16a02af1746adbc173a5dc5a16012468133777c5 SHA256：0039eb194f00b975145a35ede6b48d9c1ea87a6b2e61ac015b3d38e7e46aecbb
样本 8 文件名称：8cb78327bd69fda61afac9393187ad5533a63d43ebf74c0f9800bedb814b20ad 原始文件名：wmisvcs64.dll 文件大小：1.18 MB（1,239,040 字节） 编译时间：2006-05-11 14:22:00 MD5：883034ba4657ba4765a20f680721d0ea SHA1：eafb4e041587f4204c2dda9bbb91622ce34421f0 SHA256：8cb78327bd69fda61afac9393187ad5533a63d43ebf74c0f9800bedb814b20ad

根据获取到的样本的导出函数，可以将样本归为以下三类：


类型一
32bit
af8ccd0294530c659580f522fcc8492d92c2296dc068f9a42474d52b2b2f16e4
CheckValidConnection
GenerateRsopPolicy
IsInsideContext
IsOutsideContext
RemoveRsopPolicy
RsopFileAccessCheck
32bit
426aa55d2afb9eb08b601d373671594f39a1d9d9a73639c4a64f17d674ca9a82
CheckValidConnection
CheckValidLengthConnection
RsopFileAccessCheck
64bit
15a9b1d233c02d1fdf80071797ff9077f6ac374958f7d0f2b6e84b8d487c9cd1
CheckValidConnection
CheckValidLengthConnection
RsopFileAccessCheck
64bit
69227d046ad108e5729e6bfaecc4e05a0da30d8e7e87769d9d3bbf17b4366e64
CheckValidConnection
GenerateRsopPolicy
IsInsideContext
RemoveRsopPolicy


类型二
32bit
134849f697ab5f31ffb043b06e9ca1c9b98ffebba8af8ccdedd036a6263bf3a4
ConnectHost
DisconnectHost
CreateHost
CloseHost
QueryHost
Terminate
Abort
Reset
QueryHostEx
Initialize
Unintialize
64bit
b61c62724421d38a13c58877f31298bd663c1c8f8c3fe7d108eb9c8fe5ad0362
ConnectHost
DisconnectHost
CreateHost
CloseHost
QueryHost
Terminate
Abort
Reset
QueryHostEx
Initialize
Unintialize

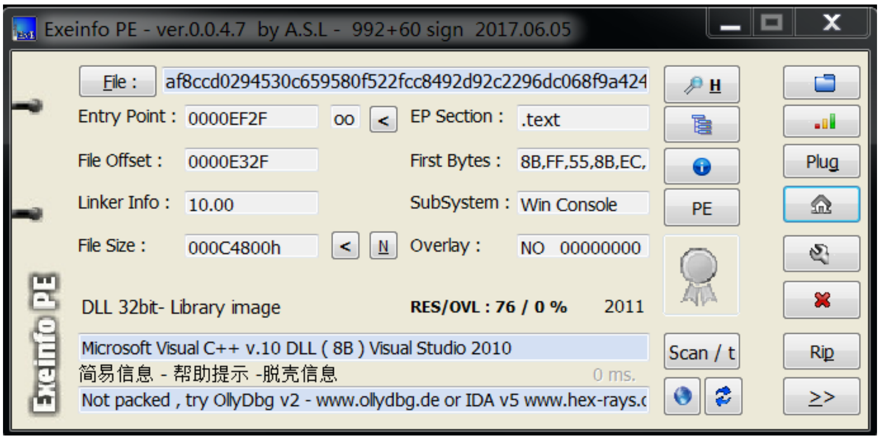

类型三
32bit
0039eb194f00b975145a35ede6b48d9c1ea87a6b2e61ac015b3d38e7e46aecbb
getInterface
64bit
8cb78327bd69fda61afac9393187ad5533a63d43ebf74c0f9800bedb814b20ad
getInterface

三、技术分析

af8ccd0294530c659580f522fcc8492d92c2296dc068f9a42474d52b2b2f16e4

静态信息

DLL x86



Exeinfo PE - ver.0.0.4.7 by A.S.L - 992+60 sign 2017.06.05

File : af8ccd0294530c659580f522fcc8492d92c2296dc068f9a424

Entry Point : 0000EF2F oo

EP Section : .text

File Offset : 0000E32F

First Bytes : 8B,FF,55,8B,EC,

Linker Info : 10.00

SubSystem : Win Console

File Size : 000C4800h

Overlay : NO 00000000

DLL 32bit- Library image

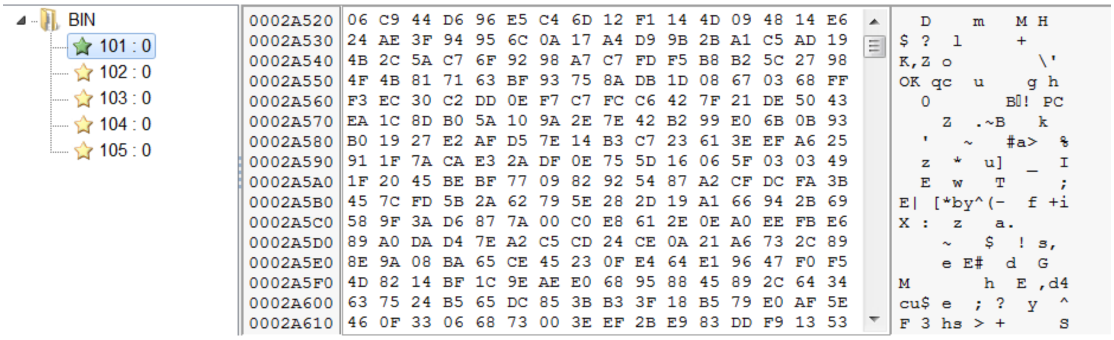
RES/OVL : 76 / 0 % 2011

Microsoft Visual C++ v.10 DLL (8B) Visual Studio 2010

简易信息 - 帮助提示 -脱壳信息 0 ms.

Not packed , try OllyDbg v2 - www.ollydbg.de or IDA v5 www.hex-rays.c

资源部分

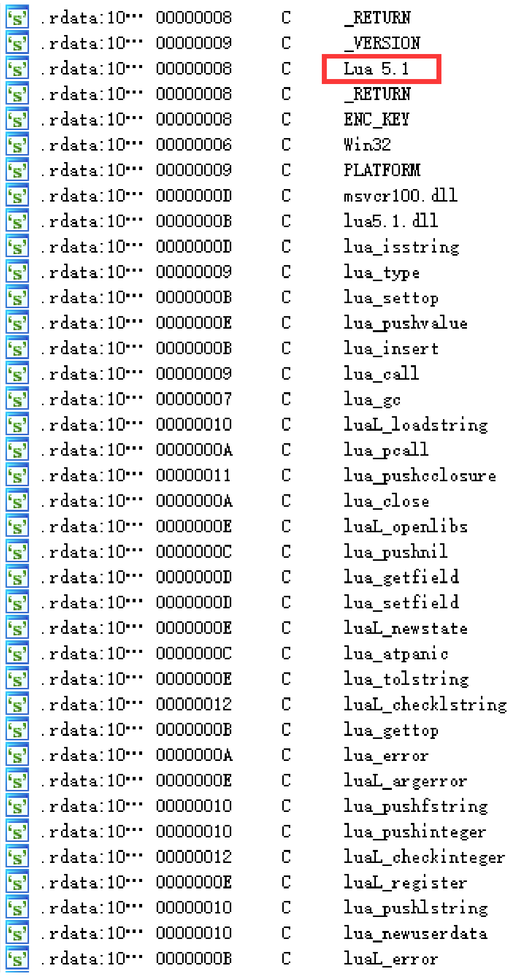


BIN
- 101 : 0
- 102 : 0
- 103 : 0
- 104 : 0
- 105 : 0

```
0002A520  06 C9 44 D6 96 E5 C4 6D 12 F1 14 4D 09 48 14 E6    D    m   M H
0002A530  24 AE 3F 94 95 6C 0A 17 A4 D9 9B 2B A1 C5 AD 19   $ ?  l      +
0002A540  4B 2C 5A C7 6F 92 98 A7 C7 FD F5 B8 B2 5C 27 98   K,Z o        \'
0002A550  4F 4B 81 71 63 BF 93 75 8A DB 1D 08 67 03 68 FF   OK qc  u    g h
0002A560  F3 EC 30 C2 DD 0E F7 C7 FC C6 42 7F 21 DE 50 43     0        B! PC
0002A570  EA 1C 8D B0 5A 10 9A 2E 7E 42 B2 99 E0 6B 0B 93       Z  .~B   k
0002A580  B0 19 27 E2 AF D5 7E 14 B3 C7 23 61 3E EF A6 25     '   ~    #a>  %
0002A590  91 1F 7A CA E3 2A DF 0E 75 5D 16 06 5F 03 03 49     z  *  u]   _  I
0002A5A0  1F 20 45 BE BF 77 09 82 92 54 87 A2 CF DC FA 3B     E  w   T      ;
0002A5B0  45 7C FD 5B 2A 62 79 5E 28 2D 19 A1 66 94 2B 69   E| [*by^(-  f +i
0002A5C0  58 9F 3A D6 87 7A 00 C0 E8 61 2E 0E A0 EE FB E6   X :  z   a.
0002A5D0  89 A0 DA D4 7E A2 C5 CD 24 CE 0A 21 A6 73 2C 89       ~    $  ! s,
0002A5E0  8E 9A 08 BA 65 CE 45 23 0F E4 64 E1 96 47 F0 F5       e E#  d  G
0002A5F0  4D 82 14 BF 1C 9E AE E0 68 95 88 45 89 2C 64 34   M        h  E ,d4
0002A600  63 75 24 B5 65 DC 85 3B B3 3F 18 B5 79 E0 AF 5E   cu$ e  ; ?  y   ^
0002A610  46 0F 33 06 68 73 00 3E EF 2B E9 83 DD F9 13 53   F 3 hs > +      S
```

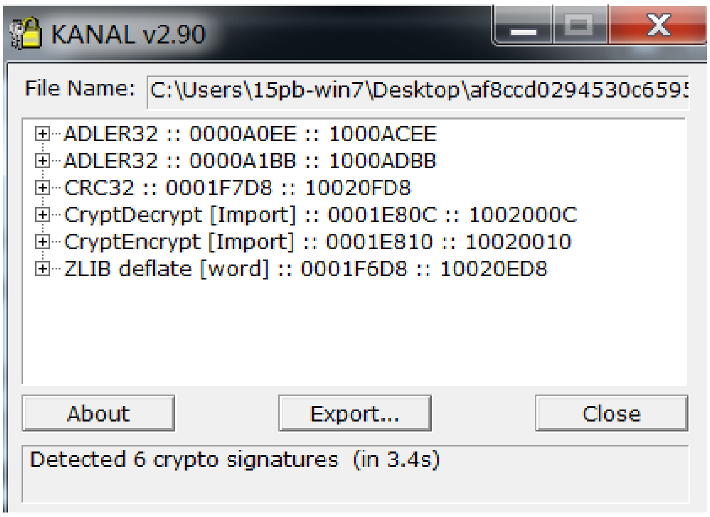
包含 Lua 5.1 相关内容

代码分析



```
.rdata:10···  00000008    C    _RETURN
.rdata:10···  00000009    C    _VERSION
.rdata:10···  00000008    C    Lua 5.1
.rdata:10···  00000008    C    _RETURN
.rdata:10···  00000008    C    ENC_KEY
.rdata:10···  00000006    C    Win32
.rdata:10···  00000009    C    PLATFORM
.rdata:10···  0000000D    C    msvcr100.dll
.rdata:10···  0000000B    C    lua5.1.dll
.rdata:10···  0000000D    C    lua_isstring
.rdata:10···  00000009    C    lua_type
.rdata:10···  0000000B    C    lua_settop
.rdata:10···  0000000E    C    lua_pushvalue
.rdata:10···  0000000B    C    lua_insert
.rdata:10···  00000009    C    lua_call
.rdata:10···  00000007    C    lua_gc
.rdata:10···  00000010    C    luaL_loadstring
.rdata:10···  0000000A    C    lua_pcall
.rdata:10···  00000011    C    lua_pushcclosure
.rdata:10···  0000000A    C    lua_close
.rdata:10···  0000000E    C    luaL_openlibs
.rdata:10···  0000000C    C    lua_pushnil
.rdata:10···  0000000D    C    lua_getfield
.rdata:10···  0000000D    C    lua_setfield
.rdata:10···  0000000E    C    luaL_newstate
.rdata:10···  0000000C    C    lua_atpanic
.rdata:10···  0000000E    C    lua_tolstring
.rdata:10···  00000012    C    luaL_checklstring
.rdata:10···  0000000B    C    lua_gettop
.rdata:10···  0000000A    C    lua_error
.rdata:10···  0000000E    C    luaL_argerror
.rdata:10···  00000010    C    lua_pushfstring
.rdata:10···  00000010    C    lua_pushinteger
.rdata:10···  00000012    C    luaL_checkinteger
.rdata:10···  0000000E    C    luaL_register
.rdata:10···  00000010    C    lua_pushlstring
.rdata:10···  00000010    C    lua_newuserdata
.rdata:10···  0000000B    C    luaL_error
```

可能存在的加密算法



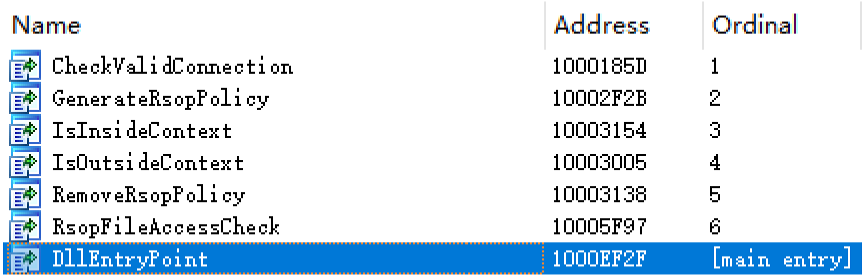
File Name: C:\Users\15pb-win7\Desktop\af8ccd0294530c659

```
ADLER32 :: 0000A0EE :: 1000ACEE
ADLER32 :: 0000A1BB :: 1000ADBB
CRC32 :: 0001F7D8 :: 10020FD8
CryptDecrypt [Import] :: 0001E80C :: 1002000C
CryptEncrypt [Import] :: 0001E810 :: 10020010
ZLIB deflate [word] :: 0001F6D8 :: 10020ED8
```

Detected 6 crypto signatures (in 3.4s)

代码分析

导出函数



Name	Address	Ordinal
CheckValidConnection	1000185D	1
GenerateRsopPolicy	10002F2B	2
IsInsideContext	10003154	3
IsOutsideContext	10003005	4
RemoveRsopPolicy	10003138	5
RsopFileAccessCheck	10005F97	6
DllEntryPoint	1000EF2F	[main entry]

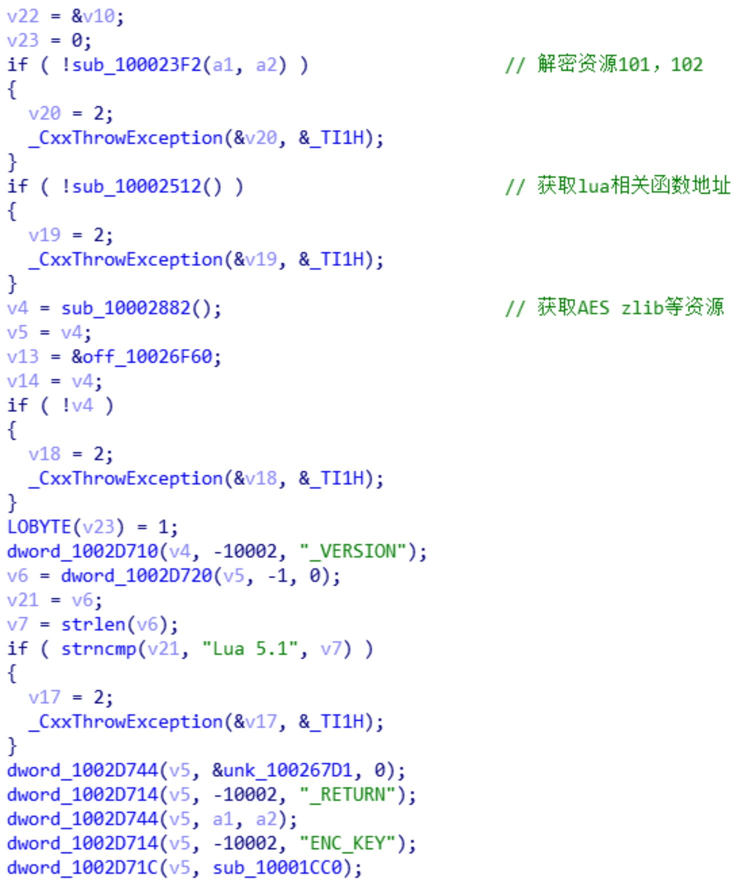
根据导出函数名称猜测，此文件包含 6 个功能

- 检测有效链接
- 生成 Rsop 策略
- 是否内部上下文
- 是否外部上下文
- 移除 Rsop 策略
- Rsop 文件访问检查

DLL 运行后，导出函数需经过外部调用才能运行，其他几个函数最终会直接或间接调用 CheckValidConnection，主要功能在此函数中。

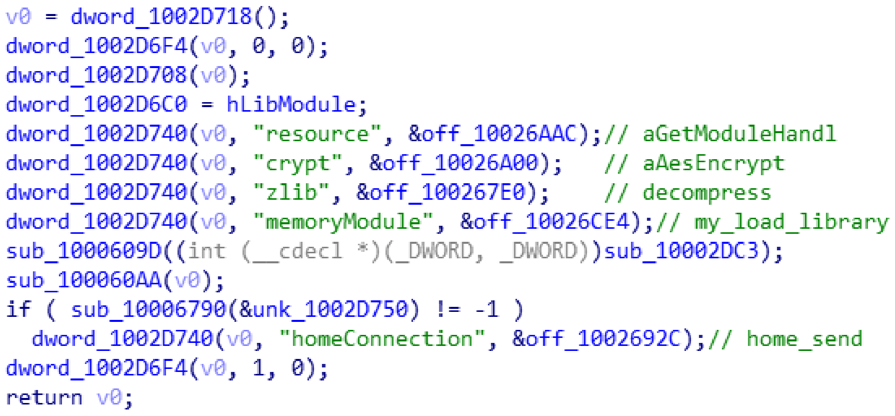
CheckValidConnection

整体功能，解密资源文件，获取 Lua 相关函数等信息。



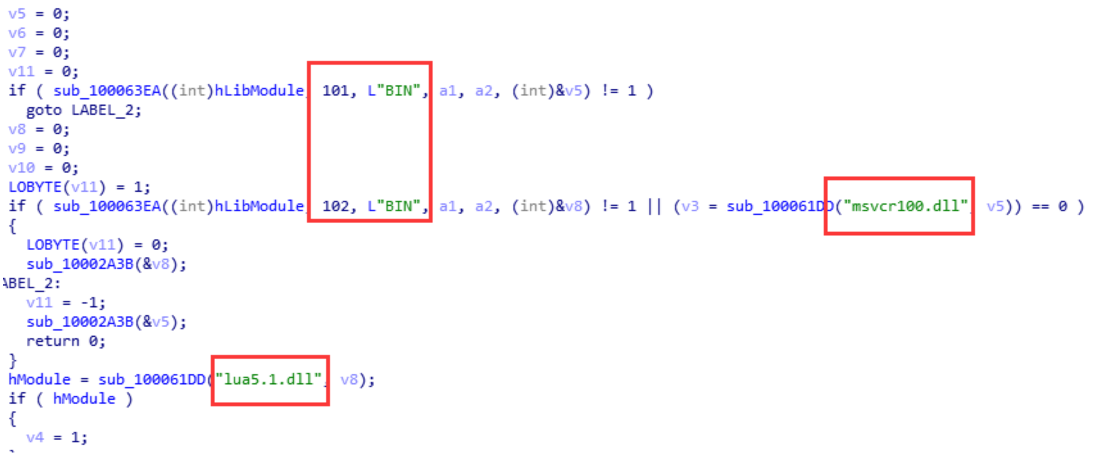
```
v22 = &v10;
v23 = 0;
if ( !sub_100023F2(a1, a2) )                  // 解密资源101，102
{
  v20 = 2;
  _CxxThrowException(&v20, &_TI1H);
}
if ( !sub_10002512() )                        // 获取lua相关函数地址
{
  v19 = 2;
  _CxxThrowException(&v19, &_TI1H);
}
v4 = sub_10002882();                          // 获取AES zlib等资源
v5 = v4;
v13 = &off_10026F60;
v14 = v4;
if ( !v4 )
{
  v18 = 2;
  _CxxThrowException(&v18, &_TI1H);
}
LOBYTE(v23) = 1;
dword_1002D710(v4, -10002, "_VERSION");
v6 = dword_1002D720(v5, -1, 0);
v21 = v6;
v7 = strlen(v6);
if ( strncmp(v21, "Lua 5.1", v7) )
{
  v17 = 2;
  _CxxThrowException(&v17, &_TI1H);
}
dword_1002D744(v5, &unk_100267D1, 0);
dword_1002D714(v5, -10002, "_RETURN");
dword_1002D744(v5, a1, a2);
dword_1002D714(v5, -10002, "ENC_KEY");
dword_1002D71C(v5, sub_10001CC0);
```

获取 AES、zlib 等资源



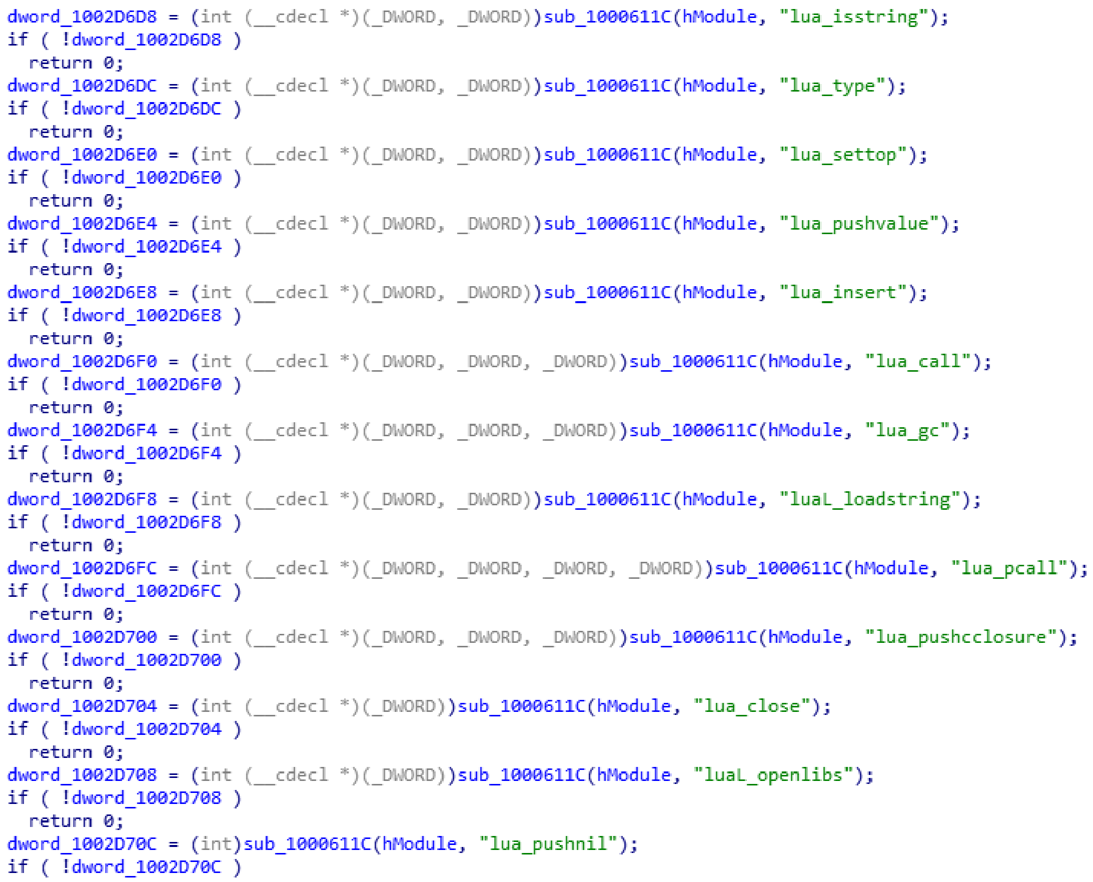
```
v0 = dword_1002D718();
dword_1002D6F4(v0, 0, 0);
dword_1002D708(v0);
dword_1002D6C0 = hLibModule;
dword_1002D740(v0, "resource", &off_10026AAC);// aGetModuleHandl
dword_1002D740(v0, "crypt", &off_10026A00);   // aAesEncrypt
dword_1002D740(v0, "zlib", &off_100267E0);    // decompress
dword_1002D740(v0, "memoryModule", &off_10026CE4);// my_load_library
sub_1000609D((int (__cdecl *)(_DWORD, _DWORD))sub_10002DC3);
sub_100060AA(v0);
if ( sub_10006790(&unk_1002D750) != -1 )
  dword_1002D740(v0, "homeConnection", &off_1002692C);// home_send
dword_1002D6F4(v0, 1, 0);
return v0;
```

解密资源文件 101，102，并载入 dll。



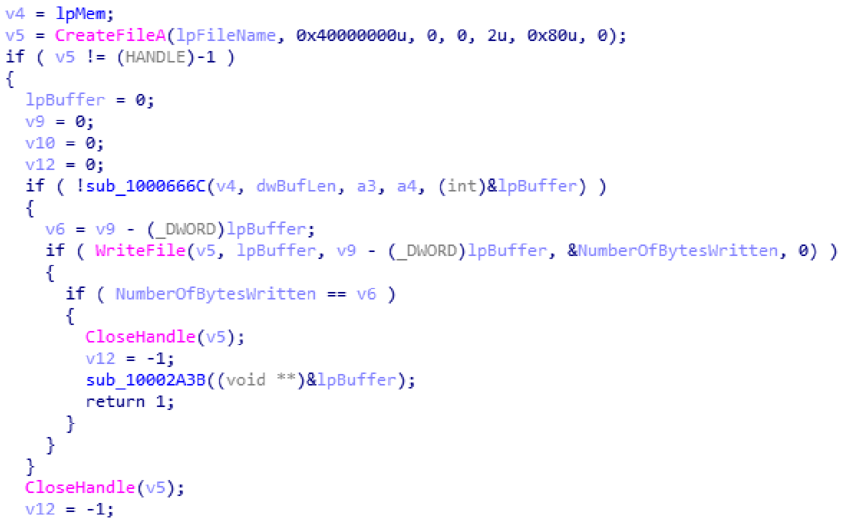
```
v5 = 0;
v6 = 0;
v7 = 0;
v11 = 0;
if ( sub_100063EA((int)hLibModule, 101, L"BIN", a1, a2, (int)&v5) != 1 )
  goto LABEL_2;
v8 = 0;
v9 = 0;
v10 = 0;
LOBYTE(v11) = 1;
if ( sub_100063EA((int)hLibModule, 102, L"BIN", a1, a2, (int)&v8) != 1 || (v3 = sub_100061DD("msvcr100.dll", v5)) == 0 )
{
  LOBYTE(v11) = 0;
  sub_10002A3B(&v8);
ABEL_2:
  v11 = -1;
  sub_10002A3B(&v5);
  return 0;
}
hModule = sub_100061DD("lua5.1.dll", v8);
if ( hModule )
{
  v4 = 1;
```

获取 lua 相关函数



```
dword_1002D6D8 = (int (__cdecl *)(_DWORD, _DWORD))sub_1000611C(hModule, "lua_isstring");
if ( !dword_1002D6D8 )
  return 0;
dword_1002D6DC = (int (__cdecl *)(_DWORD, _DWORD))sub_1000611C(hModule, "lua_type");
if ( !dword_1002D6DC )
  return 0;
dword_1002D6E0 = (int (__cdecl *)(_DWORD, _DWORD))sub_1000611C(hModule, "lua_settop");
if ( !dword_1002D6E0 )
  return 0;
dword_1002D6E4 = (int (__cdecl *)(_DWORD, _DWORD))sub_1000611C(hModule, "lua_pushvalue");
if ( !dword_1002D6E4 )
  return 0;
dword_1002D6E8 = (int (__cdecl *)(_DWORD, _DWORD))sub_1000611C(hModule, "lua_insert");
if ( !dword_1002D6E8 )
  return 0;
dword_1002D6F0 = (int (__cdecl *)(_DWORD, _DWORD, _DWORD))sub_1000611C(hModule, "lua_call");
if ( !dword_1002D6F0 )
  return 0;
dword_1002D6F4 = (int (__cdecl *)(_DWORD, _DWORD, _DWORD))sub_1000611C(hModule, "lua_gc");
if ( !dword_1002D6F4 )
  return 0;
dword_1002D6F8 = (int (__cdecl *)(_DWORD, _DWORD))sub_1000611C(hModule, "luaL_loadstring");
if ( !dword_1002D6F8 )
  return 0;
dword_1002D6FC = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD))sub_1000611C(hModule, "lua_pcall");
if ( !dword_1002D6FC )
  return 0;
dword_1002D700 = (int (__cdecl *)(_DWORD, _DWORD, _DWORD))sub_1000611C(hModule, "lua_pushcclosure");
if ( !dword_1002D700 )
  return 0;
dword_1002D704 = (int (__cdecl *)(_DWORD))sub_1000611C(hModule, "lua_close");
if ( !dword_1002D704 )
  return 0;
dword_1002D708 = (int (__cdecl *)(_DWORD))sub_1000611C(hModule, "luaL_openlibs");
if ( !dword_1002D708 )
  return 0;
dword_1002D70C = (int)sub_1000611C(hModule, "lua_pushnil");
if ( !dword_1002D70C )
```

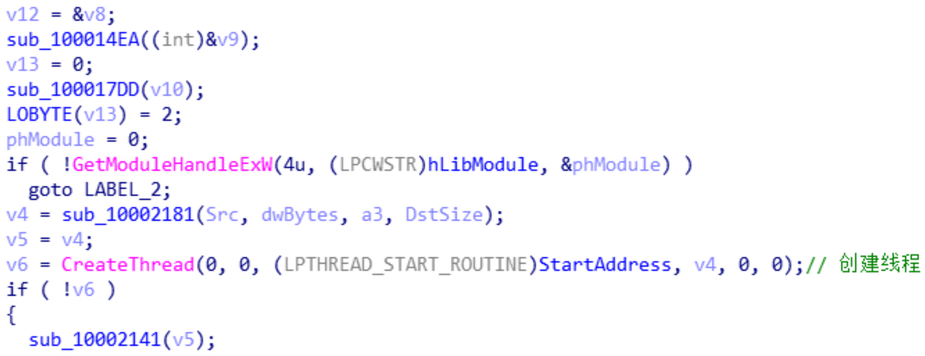
写入文件



```
v4 = lpMem;
v5 = CreateFileA(lpFileName, 0x40000000u, 0, 0, 2u, 0x80u, 0);
if ( v5 != (HANDLE)-1 )
{
  lpBuffer = 0;
  v9 = 0;
  v10 = 0;
  v12 = 0;
  if ( !sub_1000666C(v4, dwBufLen, a3, a4, (int)&lpBuffer) )
  {
    v6 = v9 - (_DWORD)lpBuffer;
    if ( WriteFile(v5, lpBuffer, v9 - (_DWORD)lpBuffer, &NumberOfBytesWritten, 0) )
    {
      if ( NumberOfBytesWritten == v6 )
      {
        CloseHandle(v5);
        v12 = -1;
        sub_10002A3B((void **)&lpBuffer);
        return 1;
      }
    }
  }
  CloseHandle(v5);
  v12 = -1;
```

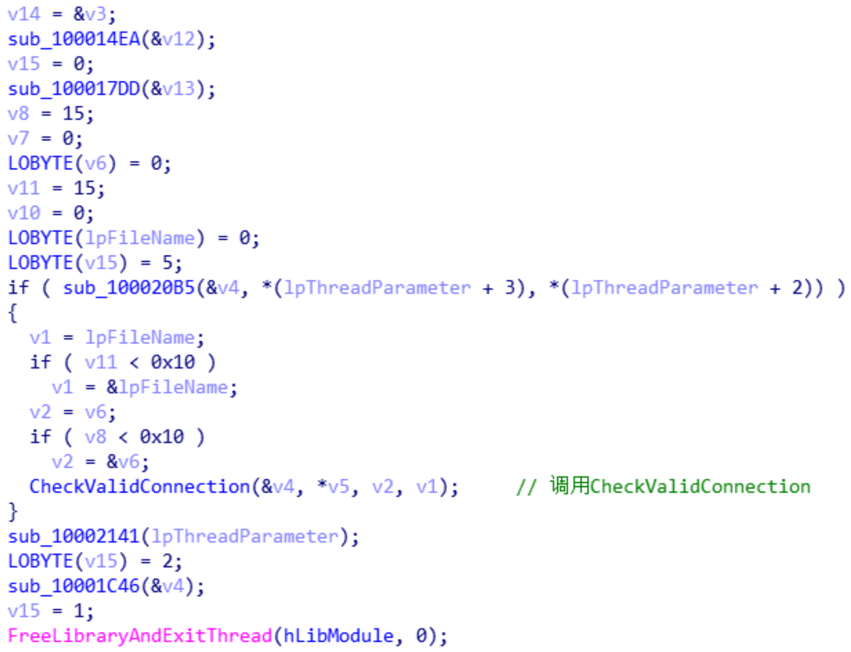
GenerateRsopPolicy

创建线程



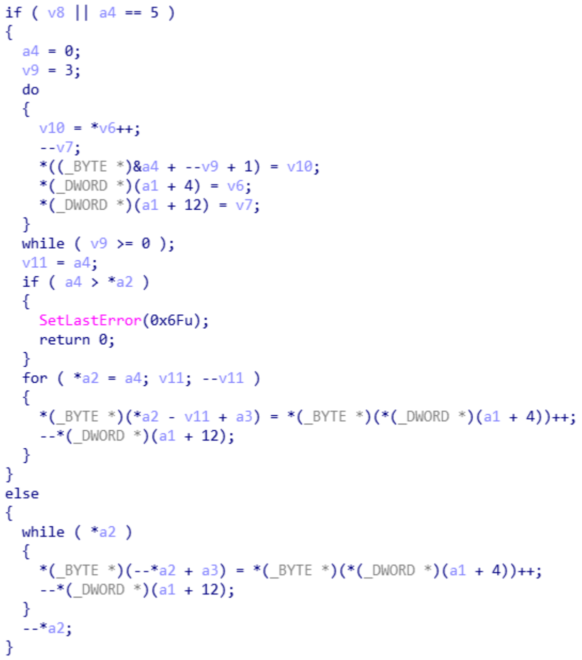
```
v12 = &v8;
sub_100014EA((int)&v9);
v13 = 0;
sub_100017DD(v10);
LOBYTE(v13) = 2;
phModule = 0;
if ( !GetModuleHandleExW(4u, (LPCWSTR)hLibModule, &phModule) )
  goto LABEL_2;
v4 = sub_10002181(Src, dwBytes, a3, DstSize);
v5 = v4;
v6 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, v4, 0, 0);// 创建线程
if ( !v6 )
{
  sub_10002141(v5);
```

线程中调用了 CheckValidConnection



```
v14 = &v3;
sub_100014EA(&v12);
v15 = 0;
sub_100017DD(&v13);
v8 = 15;
v7 = 0;
LOBYTE(v6) = 0;
v11 = 15;
v10 = 0;
LOBYTE(lpFileName) = 0;
LOBYTE(v15) = 5;
if ( sub_100020B5(&v4, *(lpThreadParameter + 3), *(lpThreadParameter + 2)) )
{
  v1 = lpFileName;
  if ( v11 < 0x10 )
    v1 = &lpFileName;
  v2 = v6;
  if ( v8 < 0x10 )
    v2 = &v6;
  CheckValidConnection(&v4, *v5, v2, v1);     // 调用CheckValidConnection
}
sub_10002141(lpThreadParameter);
LOBYTE(v15) = 2;
sub_10001C46(&v4);
v15 = 1;
FreeLibraryAndExitThread(hLibModule, 0);
```

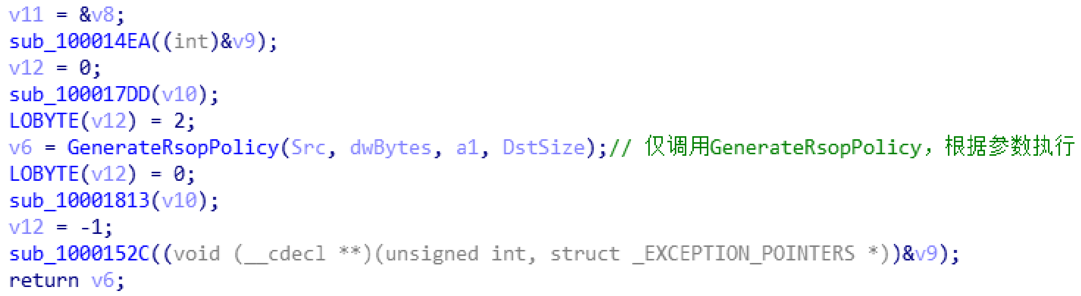
疑似解密操作



```
if ( v8 || a4 == 5 )
{
  a4 = 0;
  v9 = 3;
  do
  {
    v10 = *v6++;
    --v7;
    *((_BYTE *)&a4 + --v9 + 1) = v10;
    *(_DWORD *)(a1 + 4) = v6;
    *(_DWORD *)(a1 + 12) = v7;
  }
  while ( v9 >= 0 );
  v11 = a4;
  if ( a4 > *a2 )
  {
    SetLastError(0x6Fu);
    return 0;
  }
  for ( *a2 = a4; v11; --v11 )
  {
    *(_BYTE *)(*a2 - v11 + a3) = *(_BYTE *)(*(_DWORD *)(a1 + 4))++;
    --*(_DWORD *)(a1 + 12);
  }
}
else
{
  while ( *a2 )
  {
    *(_BYTE *)(--*a2 + a3) = *(_BYTE *)(*(_DWORD *)(a1 + 4))++;
    --*(_DWORD *)(a1 + 12);
  }
  --*a2;
}
```

IsInsideContext

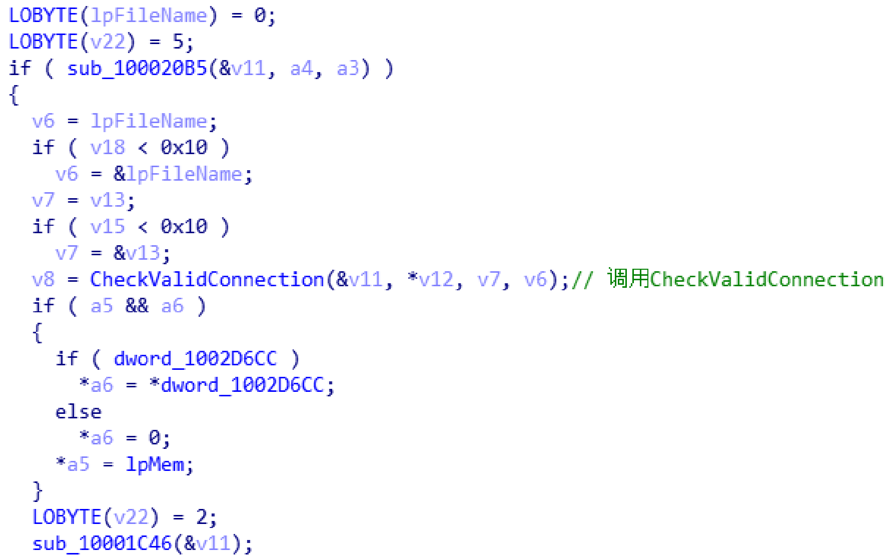
主要调用了 GenerateRsopPolicy 函数



```
v11 = &v8;
sub_100014EA((int)&v9);
v12 = 0;
sub_100017DD(v10);
LOBYTE(v12) = 2;
v6 = GenerateRsopPolicy(Src, dwBytes, a1, DstSize);// 仅调用GenerateRsopPolicy，根据参数执行
LOBYTE(v12) = 0;
sub_10001813(v10);
v12 = -1;
sub_1000152C((void (__cdecl **)(unsigned int, struct _EXCEPTION_POINTERS *))&v9);
return v6;
```

IsOutsideContext

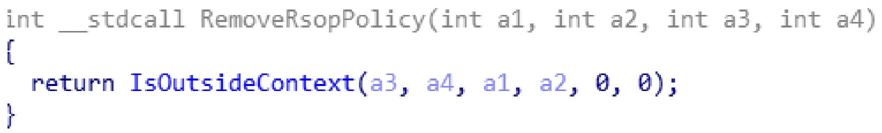
疑似进行解密操作后，调用 CheckValidConnection



```
LOBYTE(lpFileName) = 0;
LOBYTE(v22) = 5;
if ( sub_100020B5(&v11, a4, a3) )
{
  v6 = lpFileName;
  if ( v18 < 0x10 )
    v6 = &lpFileName;
  v7 = v13;
  if ( v15 < 0x10 )
    v7 = &v13;
  v8 = CheckValidConnection(&v11, *v12, v7, v6);// 调用CheckValidConnection
  if ( a5 && a6 )
  {
    if ( dword_1002D6CC )
      *a6 = *dword_1002D6CC;
    else
      *a6 = 0;
    *a5 = lpMem;
  }
  LOBYTE(v22) = 2;
  sub_10001C46(&v11);
```

RemoveRsopPolicy

调用 IsOutsideContext 函数

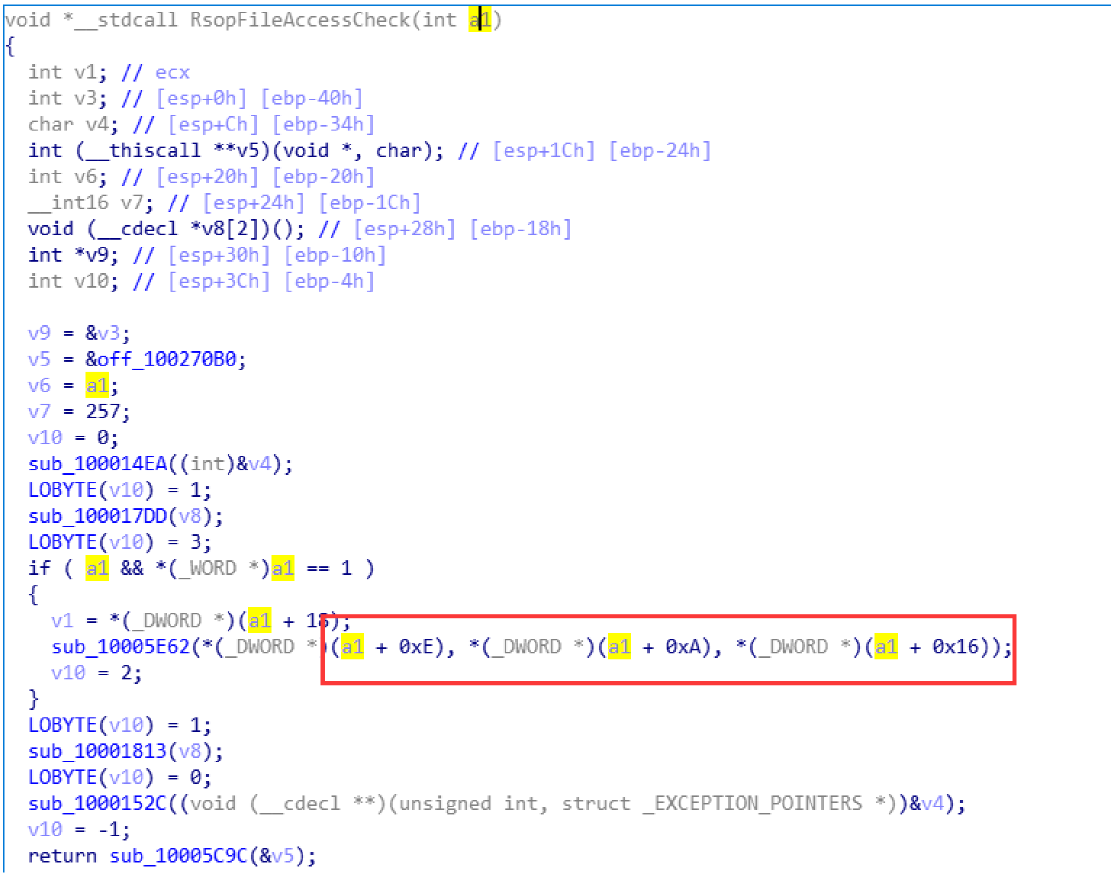


```
int __stdcall RemoveRsopPolicy(int a1, int a2, int a3, int a4)
{
  return IsOutsideContext(a3, a4, a1, a2, 0, 0);
}
```

RsopFileAccessCheck

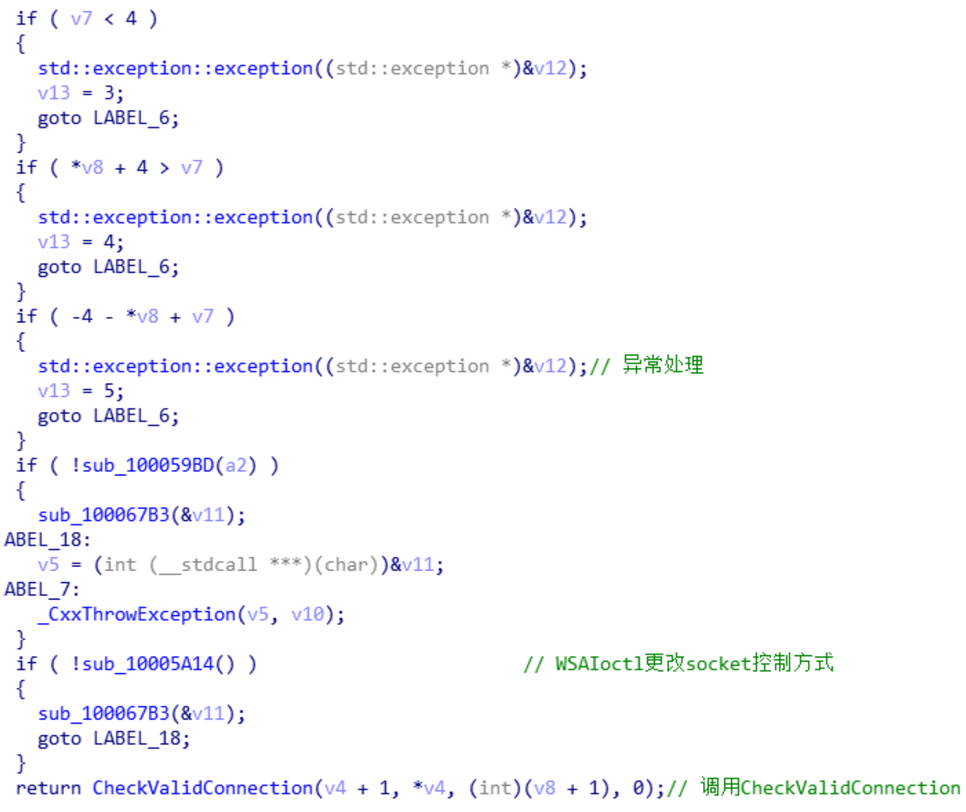
该函数的参数可能为一个结构体，其中的值会根据不同的字段来使用。最后

一个参数是 a1+0x16，说明此结构体至少有 0x16 个字节



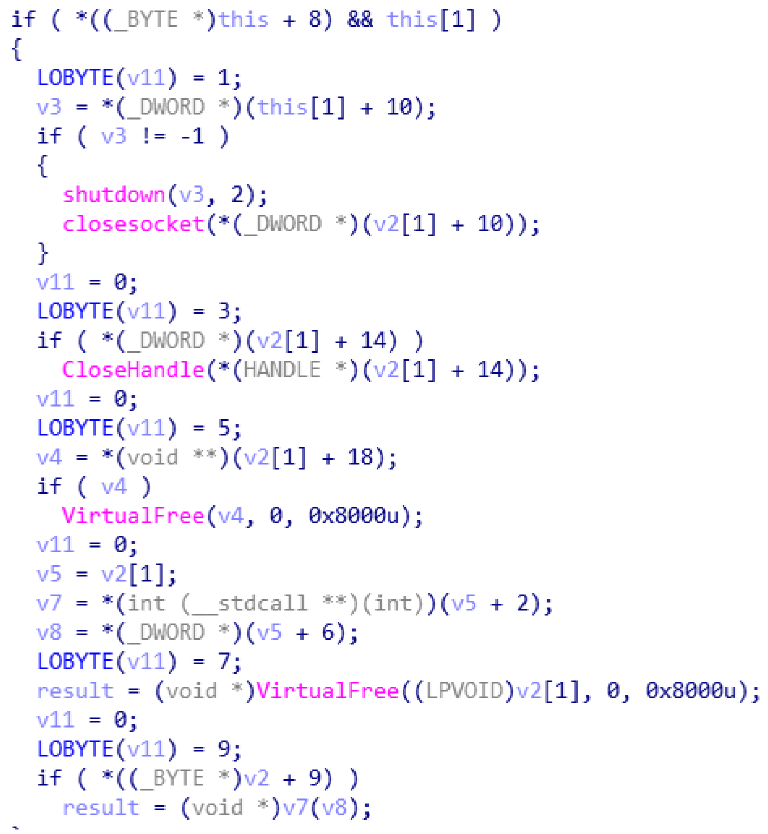
```
void *__stdcall RsopFileAccessCheck(int a1)
{
  int v1; // ecx
  int v3; // [esp+0h] [ebp-40h]
  char v4; // [esp+Ch] [ebp-34h]
  int (__thiscall **v5)(void *, char); // [esp+1Ch] [ebp-24h]
  int v6; // [esp+20h] [ebp-20h]
  __int16 v7; // [esp+24h] [ebp-1Ch]
  void (__cdecl *v8[2])(); // [esp+28h] [ebp-18h]
  int *v9; // [esp+30h] [ebp-10h]
  int v10; // [esp+3Ch] [ebp-4h]

  v9 = &v3;
  v5 = &off_100270B0;
  v6 = a1;
  v7 = 257;
  v10 = 0;
  sub_100014EA((int)&v4);
  LOBYTE(v10) = 1;
  sub_100017DD(v8);
  LOBYTE(v10) = 3;
  if ( a1 && *(_WORD *)a1 == 1 )
  {
    v1 = *(_DWORD *)(a1 + 18);
    sub_10005E62(*(_DWORD *)(a1 + 0xE), *(_DWORD *)(a1 + 0xA), *(_DWORD *)(a1 + 0x16));
    v10 = 2;
  }
  LOBYTE(v10) = 1;
  sub_10001813(v8);
  LOBYTE(v10) = 0;
  sub_1000152C((void (__cdecl **)(unsigned int, struct _EXCEPTION_POINTERS *))&v4);
  v10 = -1;
  return sub_10005C9C(&v5);
```

异常处理，socket 相关，调用 CheckValidConnection



```
 if ( v7 < 4 )
 {
   std::exception::exception((std::exception *)&v12);
   v13 = 3;
   goto LABEL_6;
 }
 if ( *v8 + 4 > v7 )
 {
   std::exception::exception((std::exception *)&v12);
   v13 = 4;
   goto LABEL_6;
 }
 if ( -4 - *v8 + v7 )
 {
   std::exception::exception((std::exception *)&v12);// 异常处理
   v13 = 5;
   goto LABEL_6;
 }
 if ( !sub_100059BD(a2) )
 {
   sub_100067B3(&v11);
ABEL_18:
   v5 = (int (__stdcall ***)(char))&v11;
ABEL_7:
   _CxxThrowException(v5, v10);
 }
 if ( !sub_10005A14() )                        // WSAIoctl更改socket控制方式
 {
   sub_100067B3(&v11);
   goto LABEL_18;
 }
 return CheckValidConnection(v4 + 1, *v4, (int)(v8 + 1), 0);// 调用CheckValidConnection
```

清理环境

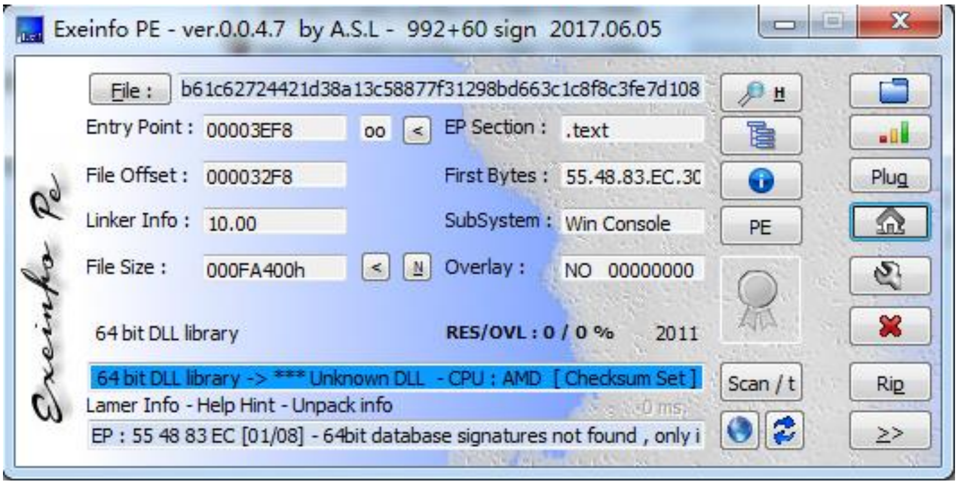


```
if ( *((_BYTE *)this + 8) && this[1] )
{
  LOBYTE(v11) = 1;
  v3 = *(_DWORD *)(this[1] + 10);
  if ( v3 != -1 )
  {
    shutdown(v3, 2);
    closesocket(*(_DWORD *)(v2[1] + 10));
  }
  v11 = 0;
  LOBYTE(v11) = 3;
  if ( *(_DWORD *)(v2[1] + 14) )
    CloseHandle(*(HANDLE *)(v2[1] + 14));
  v11 = 0;
  LOBYTE(v11) = 5;
  v4 = *(void **)(v2[1] + 18);
  if ( v4 )
    VirtualFree(v4, 0, 0x8000u);
  v11 = 0;
  v5 = v2[1];
  v7 = *(int (__stdcall **)(int))(v5 + 2);
  v8 = *(_DWORD *)(v5 + 6);
  LOBYTE(v11) = 7;
  result = (void *)VirtualFree((LPVOID)v2[1], 0, 0x8000u);
  v11 = 0;
  LOBYTE(v11) = 9;
  if ( *((_BYTE *)v2 + 9) )
    result = (void *)v7(v8);
```

b61c62724421d38a13c58877f31298bd663c1c8f8c3fe7d108eb9c8fe5ad0362

静态信息

DLL x64



Exeinfo PE - ver.0.0.4.7 by A.S.L - 992+60 sign 2017.06.05

File : b61c62724421d38a13c58877f31298bd663c1c8f8c3fe7d108

Entry Point : 00003EF8 oo < EP Section : .text

File Offset : 000032F8 First Bytes : 55.48.83.EC.30

Linker Info : 10.00 SubSystem : Win Console

File Size : 000FA400h < N Overlay : NO 00000000

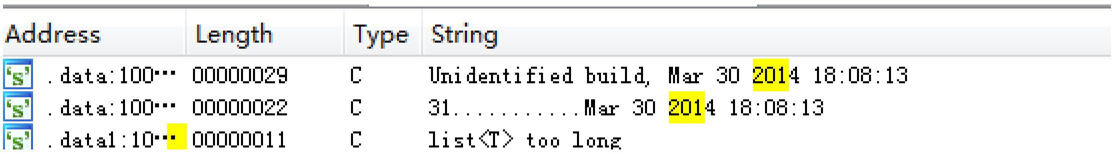
64 bit DLL library RES/OVL : 0 / 0 % 2011

64 bit DLL library -> *** Unknown DLL - CPU : AMD [Checksum Set]

Lamer Info - Help Hint - Unpack info 0 ms.

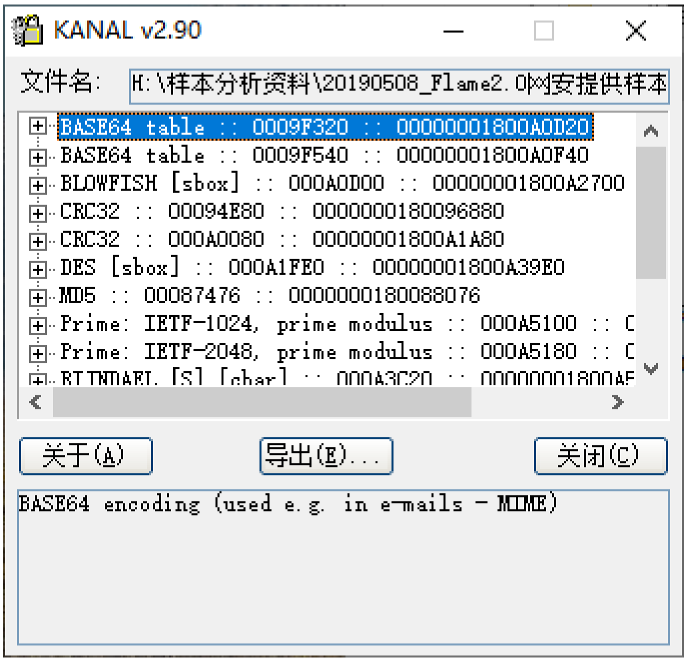
EP : 55 48 83 EC [01/08] - 64bit database signatures not found , only i

泄露的静态库编译时间信息



Address	Length	Type	String
.data:100⋯	00000029	C	Unidentified build, Mar 30 2014 18:08:13
.data:100⋯	00000022	C	31..........Mar 30 2014 18:08:13
.data1:10⋯	00000011	C	list<T> too long

可能包含的加密算法



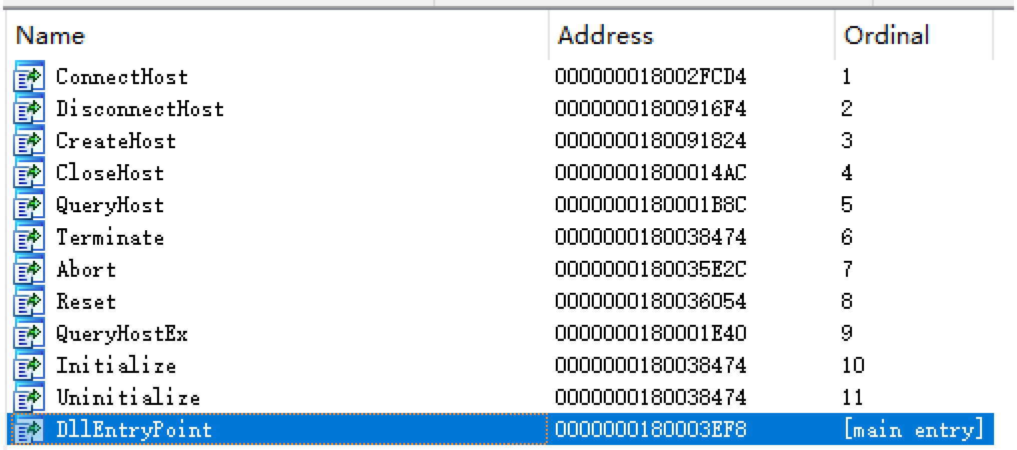
文件名: H:\样本分析资料\20190508_Flame2.0\网安提供样本

```
BASE64 table :: 0009F320 :: 00000001800A0D20
BASE64 table :: 0009F540 :: 00000001800A0F40
BLOWFISH [sbox] :: 000A0D00 :: 00000001800A2700
CRC32 :: 00094E80 :: 0000000180096880
CRC32 :: 000A0080 :: 00000001800A1A80
DES [sbox] :: 000A1FE0 :: 00000001800A39E0
MD5 :: 00087476 :: 0000000180088076
Prime: IETF-1024, prime modulus :: 000A5100 :: C
Prime: IETF-2048, prime modulus :: 000A5180 :: C
RIJNDAEL [S] [char] :: 000A3C20 :: 00000001800AF
```

BASE64 encoding (used e.g. in e-mails - MIME)

代码分析

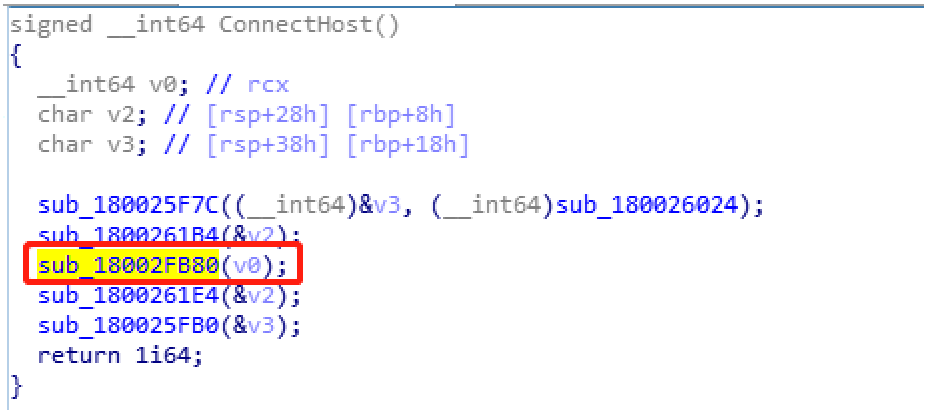
此样本导出函数:



Name	Address	Ordinal
ConnectHost	000000018002FCD4	1
DisconnectHost	00000001800916F4	2
CreateHost	0000000180091824	3
CloseHost	00000001800014AC	4
QueryHost	0000000180001B8C	5
Terminate	0000000180038474	6
Abort	000000018003SE2C	7
Reset	0000000180036054	8
QueryHostEx	0000000180001E40	9
Initialize	0000000180038474	10
Uninitialize	0000000180038474	11
DllEntryPoint	0000000180003EF8	[main entry]

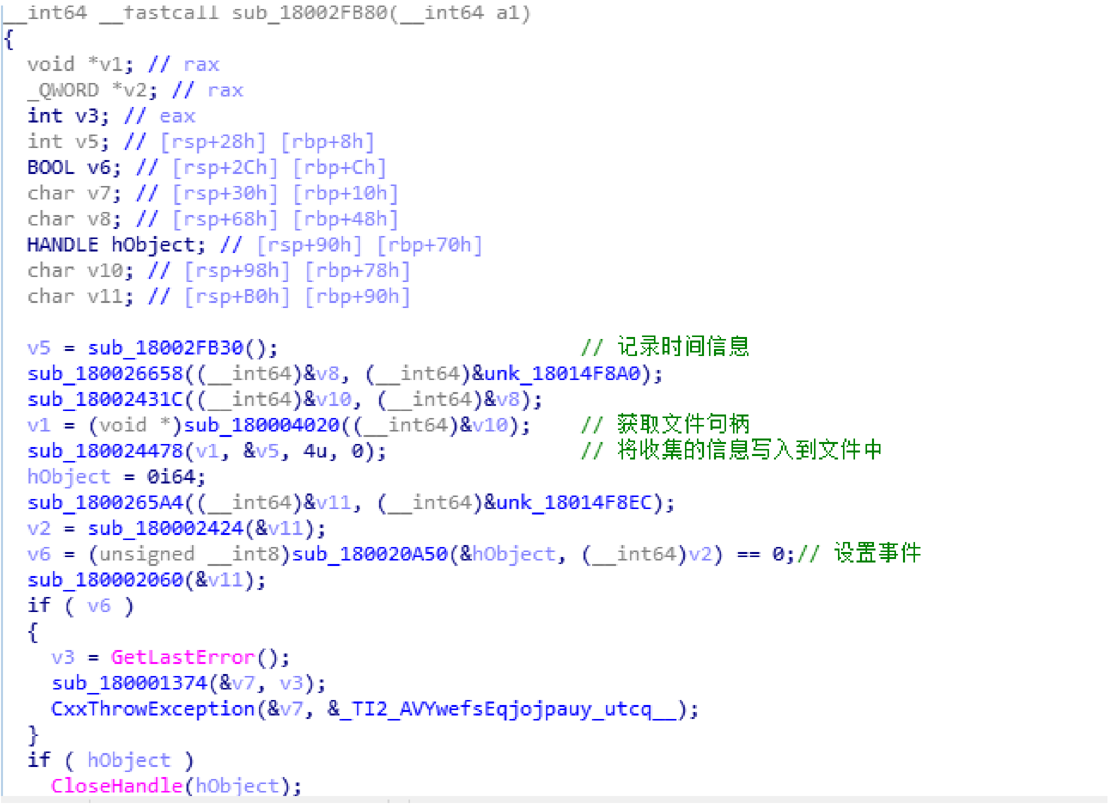
ConnectHost

经过分析，sub_18002FB80()函数为主要的功能函数。



```
signed __int64 ConnectHost()
{
  __int64 v0; // rcx
  char v2; // [rsp+28h] [rbp+8h]
  char v3; // [rsp+38h] [rbp+18h]

  sub_180025F7C((__int64)&v3, (__int64)sub_180026024);
  sub_1800261B4(&v2);
  sub_18002FB80(v0);
  sub_1800261E4(&v2);
  sub_180025FB0(&v3);
  return 1i64;
}
```

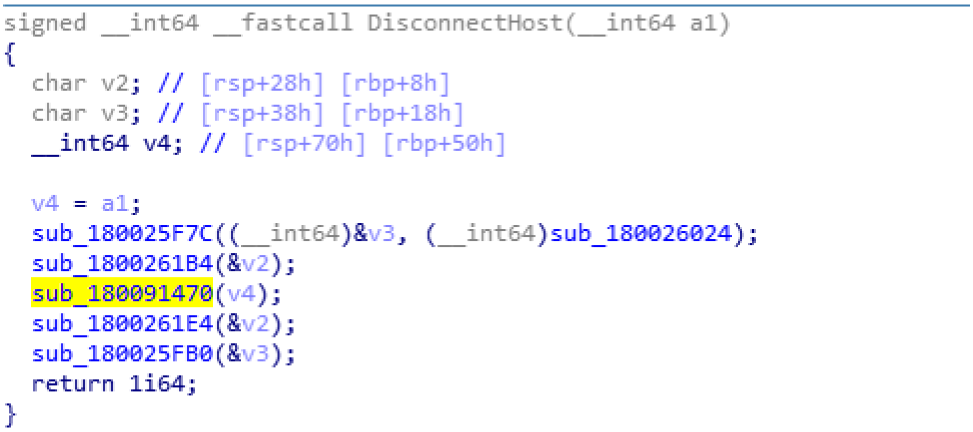
在 sub_18002FB80()函数中，会将收集到的时间信息写入到文件；然后通过字符串的拼接，得到事件名称，打开事件并对其进行设置。函数如下：



```
__int64 __fastcall sub_18002FB80(__int64 a1)
{
  void *v1; // rax
  _QWORD *v2; // rax
  int v3; // eax
  int v5; // [rsp+28h] [rbp+8h]
  BOOL v6; // [rsp+2Ch] [rbp+Ch]
  char v7; // [rsp+30h] [rbp+10h]
  char v8; // [rsp+68h] [rbp+48h]
  HANDLE hObject; // [rsp+90h] [rbp+70h]
  char v10; // [rsp+98h] [rbp+78h]
  char v11; // [rsp+B0h] [rbp+90h]

  v5 = sub_18002FB30();                                  // 记录时间信息
  sub_180026658((__int64)&v8, (__int64)&unk_18014F8A0);
  sub_18002431C((__int64)&v10, (__int64)&v8);
  v1 = (void *)sub_180004020((__int64)&v10);             // 获取文件句柄
  sub_180024478(v1, &v5, 4u, 0);                         // 将收集的信息写入到文件中
  hObject = 0i64;
  sub_1800265A4((__int64)&v11, (__int64)&unk_18014F8EC);
  v2 = sub_180002424(&v11);
  v6 = (unsigned __int8)sub_180020A50(&hObject, (__int64)v2) == 0;// 设置事件
  sub_180002060(&v11);
  if ( v6 )
  {
    v3 = GetLastError();
    sub_180001374(&v7, v3);
    CxxThrowException(&v7, &_TI2_AVYwefsEqjojpauy_utcq__);
  }
  if ( hObject )
    CloseHandle(hObject);
```

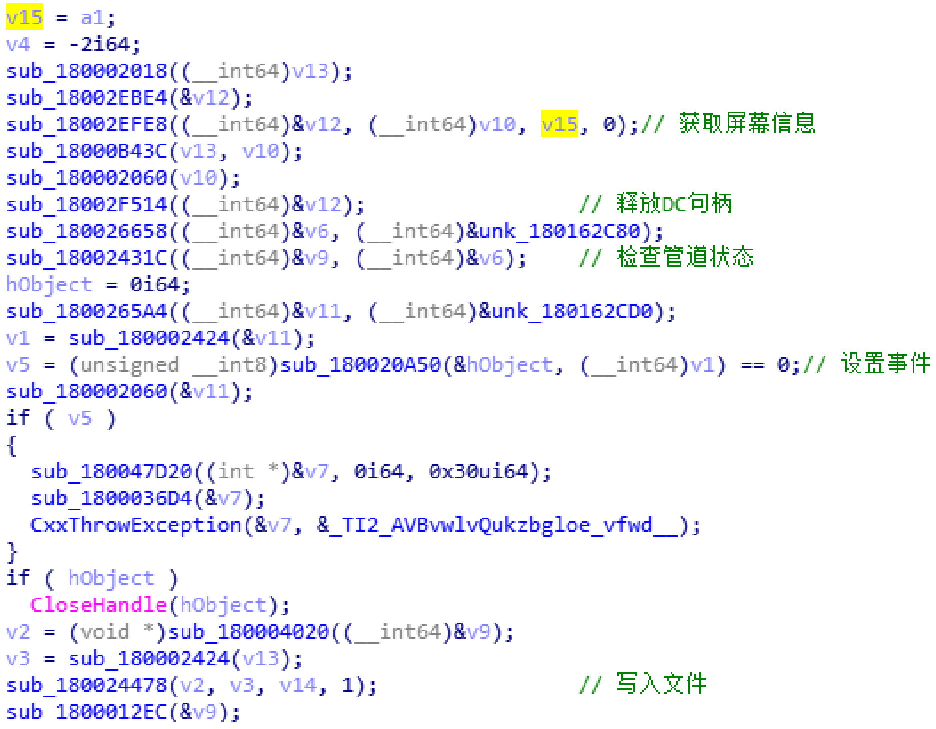
DisconnectHost

经过分析，为 sub_180091470()函数主要的功能函数：



```
signed __int64 __fastcall DisconnectHost(__int64 a1)
{
  char v2; // [rsp+28h] [rbp+8h]
  char v3; // [rsp+38h] [rbp+18h]
  __int64 v4; // [rsp+70h] [rbp+50h]

  v4 = a1;
  sub_180025F7C((__int64)&v3, (__int64)sub_180026024);
  sub_1800261B4(&v2);
  sub_180091470(v4);
  sub_1800261E4(&v2);
  sub_180025FB0(&v3);
  return 1i64;
}
```

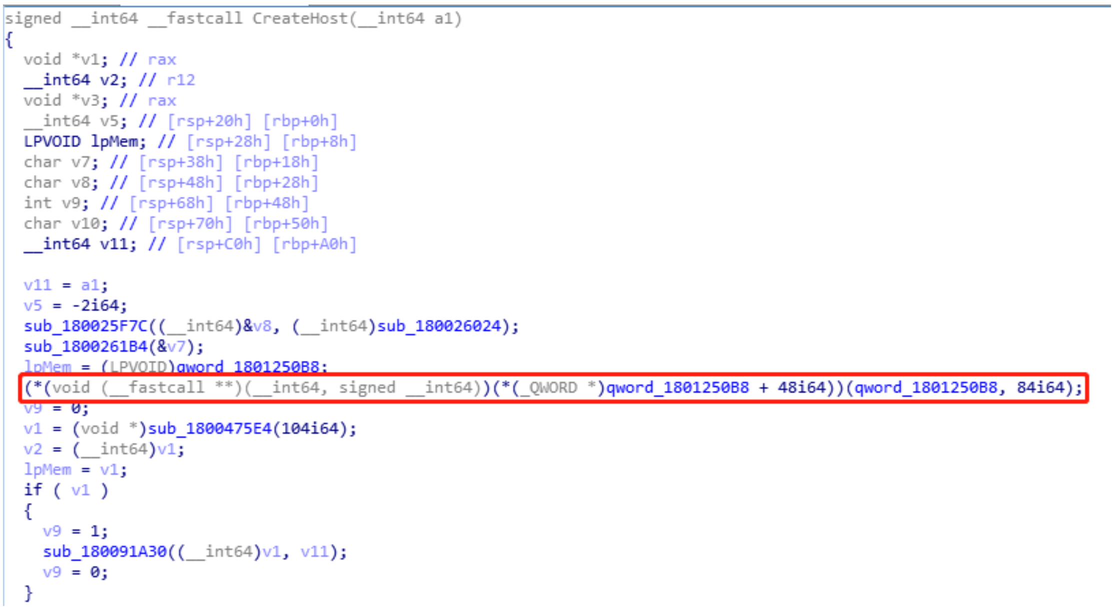
对 sub_180091470()函数进行分析，发现其具有获取屏幕信息、查看管道状态、设置事件和写入文件的操作：



```
v15 = a1;
v4 = -2i64;
sub_180002018((__int64)v13);
sub_18002EBE4(&v12);
sub_18002EFE8((__int64)&v12, (__int64)v10, v15, 0);// 获取屏幕信息
sub_18000B43C(v13, v10);
sub_180002060(v10);
sub_18002F514((__int64)&v12);                // 释放DC句柄
sub_180026658((__int64)&v6, (__int64)&unk_180162C80);
sub_18002431C((__int64)&v9, (__int64)&v6);    // 检查管道状态
hObject = 0i64;
sub_1800265A4((__int64)&v11, (__int64)&unk_180162CD0);
v1 = sub_180002424(&v11);
v5 = (unsigned __int8)sub_180020A50(&hObject, (__int64)v1) == 0;// 设置事件
sub_180002060(&v11);
if ( v5 )
{
  sub_180047D20((int *)&v7, 0i64, 0x30ui64);
  sub_1800036D4(&v7);
  CxxThrowException(&v7, &_TI2_AVBvwlvQukzbgloe_vfwd__);
}
if ( hObject )
  CloseHandle(hObject);
v2 = (void *)sub_180004020((__int64)&v9);
v3 = sub_180002424(v13);
sub_180024478(v2, v3, v14, 1);               // 写入文件
sub_1800012EC(&v9);
```

CreateHost

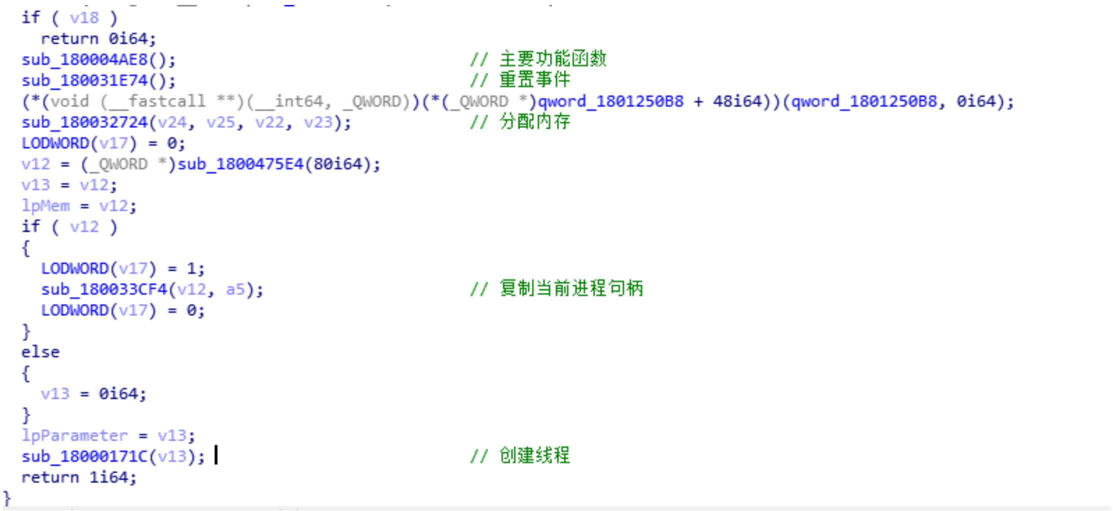
经过分析，判断本函数主要用于初始化操作，对一些字段进行赋值，在函数中，存在一个函数指针的调用，由于无法对样本进行调试，所以无法判断该函数执行的功能：



```
signed __int64 __fastcall CreateHost(__int64 a1)
{
  void *v1; // rax
  __int64 v2; // r12
  void *v3; // rax
  __int64 v5; // [rsp+20h] [rbp+0h]
  LPVOID lpMem; // [rsp+28h] [rbp+8h]
  char v7; // [rsp+38h] [rbp+18h]
  char v8; // [rsp+48h] [rbp+28h]
  int v9; // [rsp+68h] [rbp+48h]
  char v10; // [rsp+70h] [rbp+50h]
  __int64 v11; // [rsp+C0h] [rbp+A0h]

  v11 = a1;
  v5 = -2i64;
  sub_180025F7C((__int64)&v8, (__int64)sub_180026024);
  sub_1800261B4(&v7);
  lpMem = (LPVOID)qword_1801250B8;
  (*(void (__fastcall **)(__int64, signed __int64))(*(_QWORD *)qword_1801250B8 + 48i64))(qword_1801250B8, 84i64);
  v9 = 0;
  v1 = (void *)sub_1800475E4(104i64);
  v2 = (__int64)v1;
  lpMem = v1;
  if ( v1 )
  {
    v9 = 1;
    sub_180091A30((__int64)v1, v11);
    v9 = 0;
  }
```

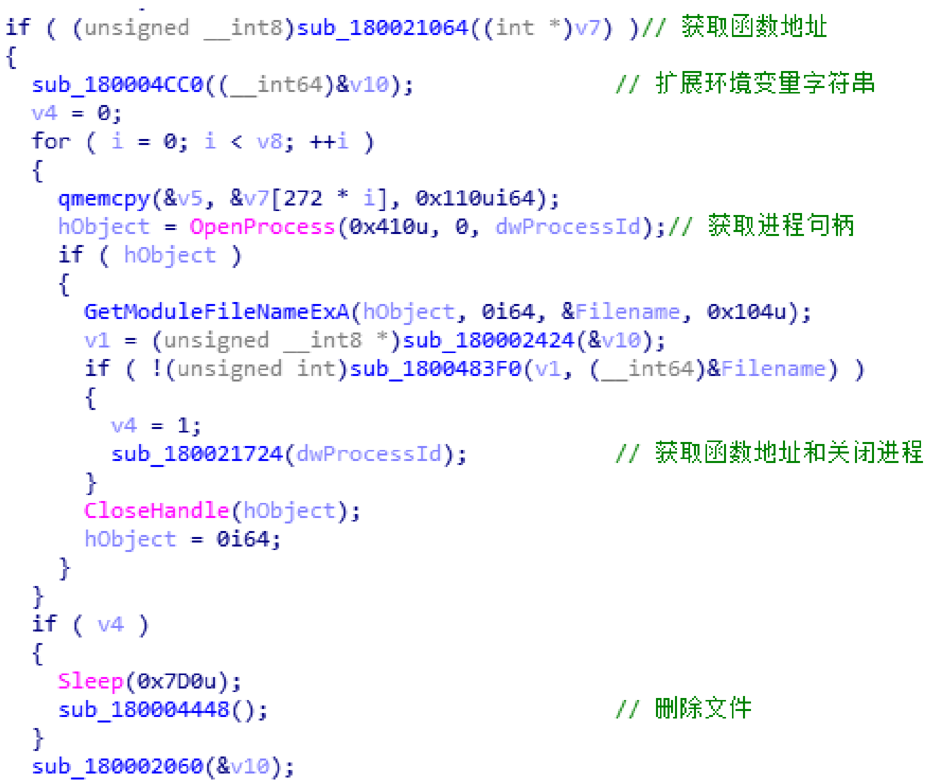
CloseHost

经过分析，sub_180004AE8()函数为主要的功能函数，除此之外，本函数还具有重置事件、分配内存、复制当前进程句柄和创建线程的功能：



```
  if ( v18 )
    return 0i64;
  sub_180004AE8();                             // 主要功能函数
  sub_180031E74();                             // 重置事件
  (*(void (__fastcall **)(__int64, _QWORD))(*(_QWORD *)qword_1801250B8 + 48i64))(qword_1801250B8, 0i64);
  sub_180032724(v24, v25, v22, v23);           // 分配内存
  LODWORD(v17) = 0;
  v12 = (_QWORD *)sub_1800475E4(80i64);
  v13 = v12;
  lpMem = v12;
  if ( v12 )
  {
    LODWORD(v17) = 1;
    sub_180033CF4(v12, a5);                    // 复制当前进程句柄
    LODWORD(v17) = 0;
  }
  else
  {
    v13 = 0i64;
  }
  lpParameter = v13;
  sub_18000171C(v13);                          // 创建线程
  return 1i64;
}
```

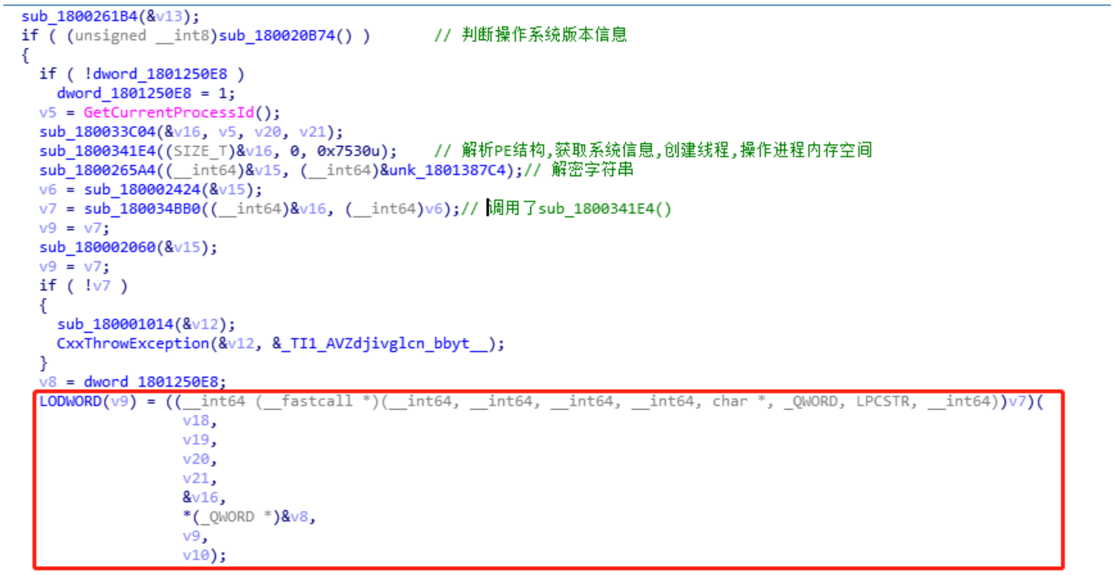
对 sub_180004AE8()函数进行分析，发现在本函数中，通过对GetProcAddress()函数进行调用，获取一些函数的地址，但由于函数名是动态获取的，在无法调试的情况下，不能得知具体函数信息。除此之外，本函数还具有拓展环境变量字符串、关闭进程和删除文件等功能：



```
if ( (unsigned __int8)sub_180021064((int *)v7) )// 获取函数地址
{
  sub_180004CC0((__int64)&v10);                 // 扩展环境变量字符串
  v4 = 0;
  for ( i = 0; i < v8; ++i )
  {
    qmemcpy(&v5, &v7[272 * i], 0x110ui64);
    hObject = OpenProcess(0x410u, 0, dwProcessId);// 获取进程句柄
    if ( hObject )
    {
      GetModuleFileNameExA(hObject, 0i64, &Filename, 0x104u);
      v1 = (unsigned __int8 *)sub_180002424(&v10);
      if ( !(unsigned int)sub_1800483F0(v1, (__int64)&Filename) )
      {
        v4 = 1;
        sub_180021724(dwProcessId);             // 获取函数地址和关闭进程
      }
      CloseHandle(hObject);
      hObject = 0i64;
    }
  }
  if ( v4 )
  {
    Sleep(0x7D0u);
    sub_180004448();                            // 删除文件
  }
  sub_180002060(&v10);
```

QueryHost

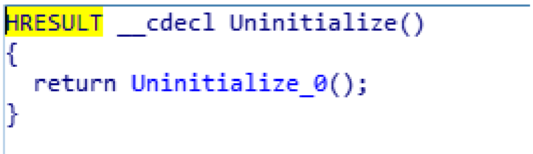
经过分析，本函数通过使用 GetVersionExA()函数，判断系统的版本信息，根据不同的版本信息，返回不同的全局变量。其次，本函数还有对 PE 结构的操作，并使用 WriteProcessMemory()函数向进程内存中写入数据。而后，本函数还通过函数指针执行了一个函数，这个函数的功能尚不可知：



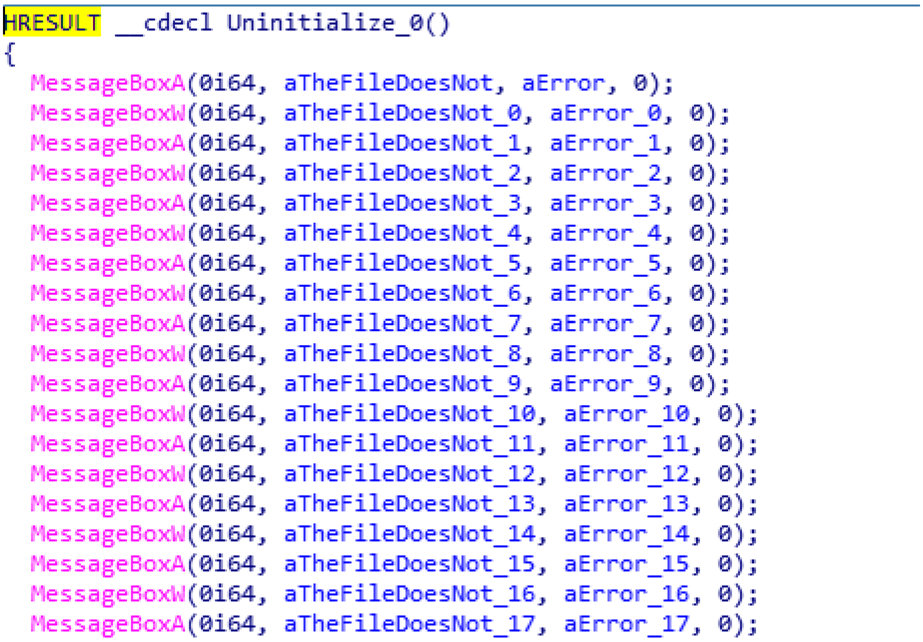
```
  sub_1800261B4(&v13);
  if ( (unsigned __int8)sub_180020B74() )       // 判断操作系统版本信息
  {
    if ( !dword_1801250E8 )
      dword_1801250E8 = 1;
    v5 = GetCurrentProcessId();
    sub_180033C04(&v16, v5, v20, v21);
    sub_1800341E4((SIZE_T)&v16, 0, 0x7530u);    // 解析PE结构,获取系统信息,创建线程,操作进程内存空间
    sub_1800265A4((__int64)&v15, (__int64)&unk_1801387C4);// 解密字符串
    v6 = sub_180002424(&v15);
    v7 = sub_180034BB0((__int64)&v16, (__int64)v6);// 调用了sub_1800341E4()
    v9 = v7;
    sub_180002060(&v15);
    v9 = v7;
    if ( !v7 )
    {
      sub_180001014(&v12);
      CxxThrowException(&v12, &_TI1_AVZdjivglcn_bbyt__);
    }
    v8 = dword_1801250E8;
    LODWORD(v9) = ((__int64 (__fastcall *)(__int64, __int64, __int64, __int64, char *, _QWORD, LPCSTR, __int64))v7)(
                    v18,
                    v19,
                    v20,
                    v21,
                    &v16,
                    *(_QWORD *)&v8,
                    v9,
                    v10);
```

Uninitialize

经过分析发现，本函数调用了大量的 MessageBoxA/W()函数，输出错误提示信息。进一步的分析中，还发现导出函数 Terminate()函数和 Initialize()函数，都被定位到 Uninitialize()函数，因此下面不再对这两个导出函数进行描述：



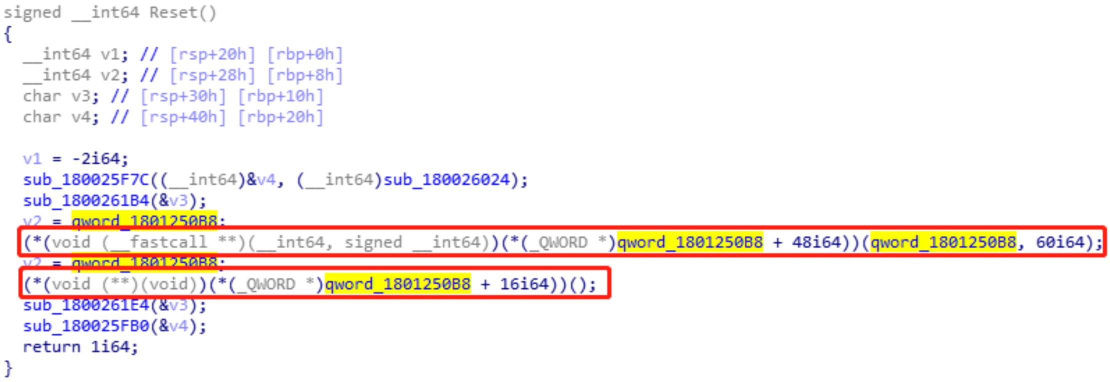
```
HRESULT __cdecl Uninitialize()
{
  return Uninitialize_0();
}
```



```
HRESULT __cdecl Uninitialize_0()
{
  MessageBoxA(0i64, aTheFileDoesNot, aError, 0);
  MessageBoxW(0i64, aTheFileDoesNot_0, aError_0, 0);
  MessageBoxA(0i64, aTheFileDoesNot_1, aError_1, 0);
  MessageBoxW(0i64, aTheFileDoesNot_2, aError_2, 0);
  MessageBoxA(0i64, aTheFileDoesNot_3, aError_3, 0);
  MessageBoxW(0i64, aTheFileDoesNot_4, aError_4, 0);
  MessageBoxA(0i64, aTheFileDoesNot_5, aError_5, 0);
  MessageBoxW(0i64, aTheFileDoesNot_6, aError_6, 0);
  MessageBoxA(0i64, aTheFileDoesNot_7, aError_7, 0);
  MessageBoxW(0i64, aTheFileDoesNot_8, aError_8, 0);
  MessageBoxA(0i64, aTheFileDoesNot_9, aError_9, 0);
  MessageBoxW(0i64, aTheFileDoesNot_10, aError_10, 0);
  MessageBoxA(0i64, aTheFileDoesNot_11, aError_11, 0);
  MessageBoxW(0i64, aTheFileDoesNot_12, aError_12, 0);
  MessageBoxA(0i64, aTheFileDoesNot_13, aError_13, 0);
  MessageBoxW(0i64, aTheFileDoesNot_14, aError_14, 0);
  MessageBoxA(0i64, aTheFileDoesNot_15, aError_15, 0);
  MessageBoxW(0i64, aTheFileDoesNot_16, aError_16, 0);
  MessageBoxA(0i64, aTheFileDoesNot_17, aError_17, 0);
```

Reset

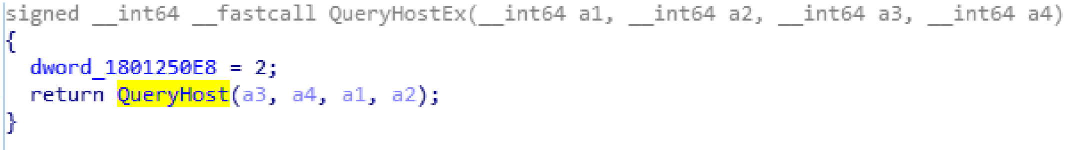
经过分析，发现本函数通过两个函数指针进行函数的调用，由于所知信息有限，不能根据现有信息，得出函数指针所指向的函数，也就无法分析其功能：



```
signed __int64 Reset()
{
  __int64 v1; // [rsp+20h] [rbp+0h]
  __int64 v2; // [rsp+28h] [rbp+8h]
  char v3; // [rsp+30h] [rbp+10h]
  char v4; // [rsp+40h] [rbp+20h]

  v1 = -2i64;
  sub_180025F7C((__int64)&v4, (__int64)sub_180026024);
  sub_1800261B4(&v3);
  v2 = qword_1801250B8;
  (*(void (__fastcall **)(__int64, signed __int64))(*(_QWORD *)qword_1801250B8 + 48i64))(qword_1801250B8, 60i64);
  v2 = qword_1801250B8;
  (*(void (**)(void))(*(_QWORD *)qword_1801250B8 + 16i64))();
  sub_1800261E4(&v3);
  sub_180025FB0(&v4);
  return 1i64;
}
```

QueryHostEx

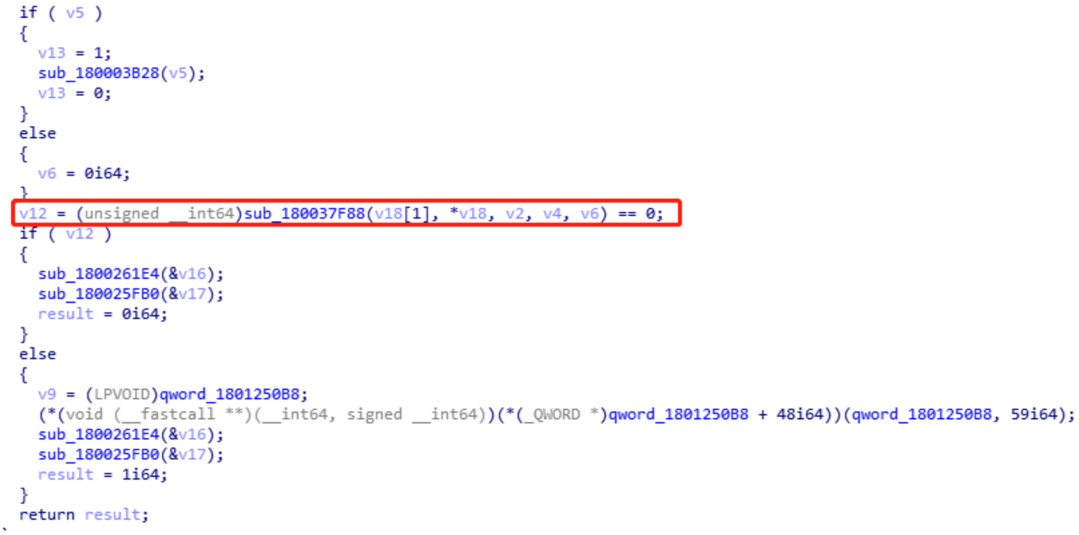
经过分析，发现本函数仅修改了全局变量 dword_1801250E8 的值，然后调用 QuerHost()函数实现其功能，具体参考 QuerHost()函数的分析：



```
signed __int64 __fastcall QueryHostEx(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
{
  dword_1801250E8 = 2;
  return QueryHost(a3, a4, a1, a2);
}
```

Abort

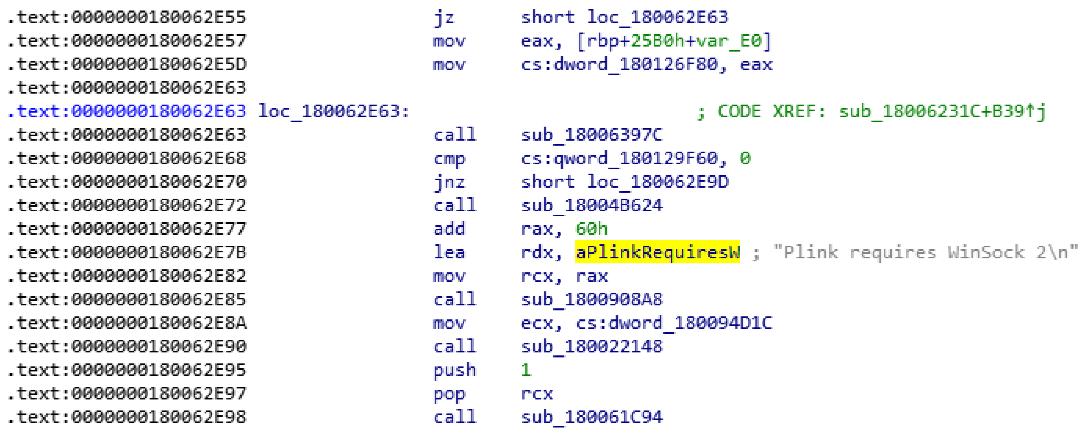
经过分析，本函数主要的功能函数为 sub_180037F88()函数。对 sub_180037F88()函数进一步分析发现，该函数存在大量的自定义函数调用，因此，该函数的具体功能尚不明确：



```
  if ( v5 )
  {
    v13 = 1;
    sub_180003B28(v5);
    v13 = 0;
  }
  else
  {
    v6 = 0i64;
  }
  v12 = (unsigned __int64)sub_180037F88(v18[1], *v18, v2, v4, v6) == 0;
  if ( v12 )
  {
    sub_1800261E4(&v16);
    sub_180025FB0(&v17);
    result = 0i64;
  }
  else
  {
    v9 = (LPVOID)qword_1801250B8;
    (*(void (__fastcall **)(__int64, signed __int64))(*(_QWORD *)qword_1801250B8 + 48i64))(qword_1801250B8, 59i64);
    sub_1800261E4(&v16);
    sub_180025FB0(&v17);
    result = 1i64;
  }
  return result;
```

使用 Plink

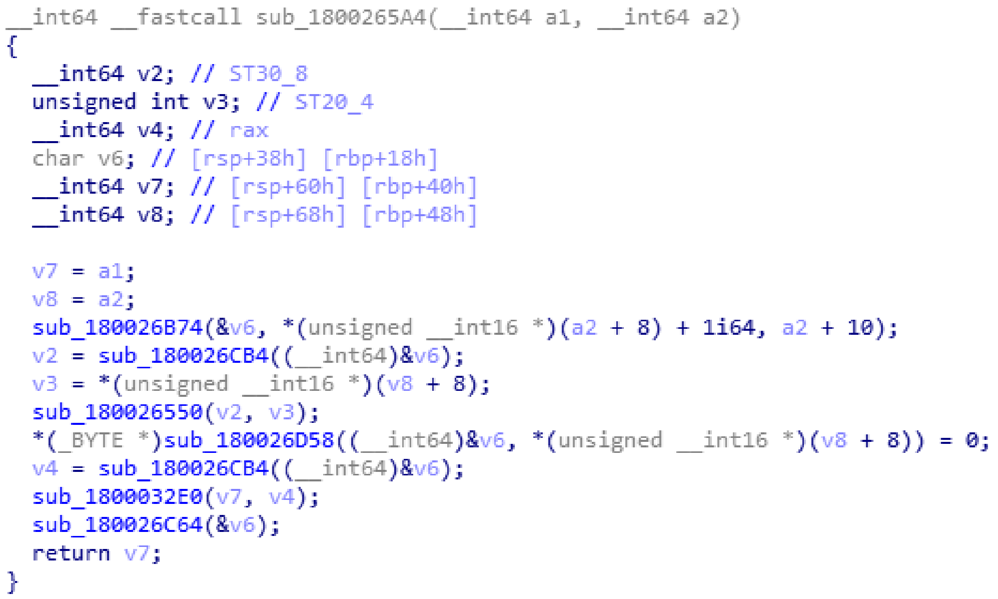
在分析过程中，发现本样本使用了 Plink 部分源码：



```
.text:0000000180062E55                 jz      short loc_180062E63
.text:0000000180062E57                 mov     eax, [rbp+25B0h+var_E0]
.text:0000000180062E5D                 mov     cs:dword_180126F80, eax
.text:0000000180062E63
.text:0000000180062E63 loc_180062E63:                          ; CODE XREF: sub_18006231C+B39↑j
.text:0000000180062E63                 call    sub_18006397C
.text:0000000180062E68                 cmp     cs:qword_180129F60, 0
.text:0000000180062E70                 jnz     short loc_180062E9D
.text:0000000180062E72                 call    sub_18004B624
.text:0000000180062E77                 add     rax, 60h
.text:0000000180062E7B                 lea     rdx, aPlinkRequiresW ; "Plink requires WinSock 2\n"
.text:0000000180062E82                 mov     rcx, rax
.text:0000000180062E85                 call    sub_1800908A8
.text:0000000180062E8A                 mov     ecx, cs:dword_180094D1C
.text:0000000180062E90                 call    sub_180022148
.text:0000000180062E95                 push    1
.text:0000000180062E97                 pop     rcx
.text:0000000180062E98                 call    sub_180061C94
```

解密函数

分析过程中，发现存在以下的解密函数：

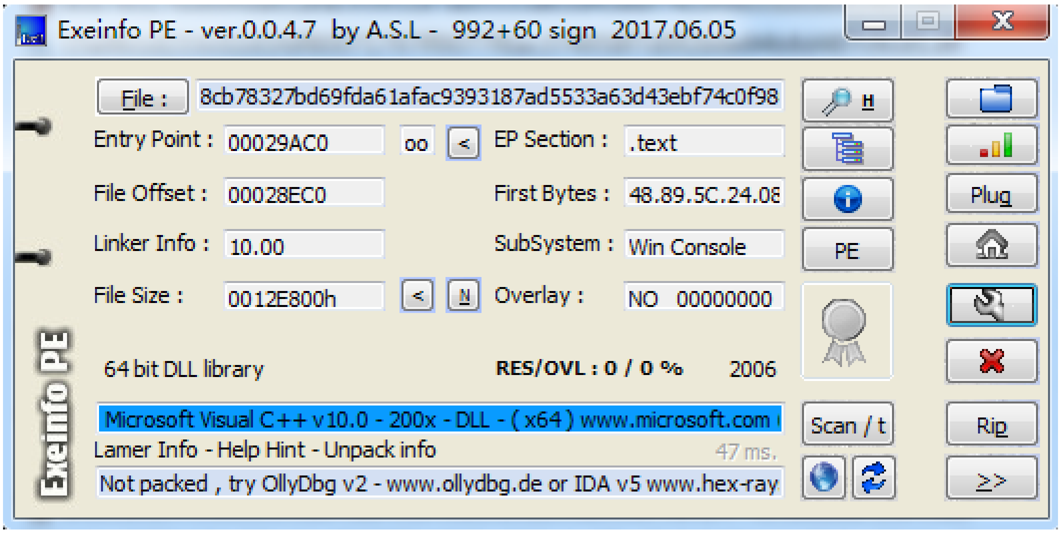


```
__int64 __fastcall sub_1800265A4(__int64 a1, __int64 a2)
{
  __int64 v2; // ST30_8
  unsigned int v3; // ST20_4
  __int64 v4; // rax
  char v6; // [rsp+38h] [rbp+18h]
  __int64 v7; // [rsp+60h] [rbp+40h]
  __int64 v8; // [rsp+68h] [rbp+48h]

  v7 = a1;
  v8 = a2;
  sub_180026B74(&v6, *(unsigned __int16 *)(a2 + 8) + 1i64, a2 + 10);
  v2 = sub_180026CB4((__int64)&v6);
  v3 = *(unsigned __int16 *)(v8 + 8);
  sub_180026550(v2, v3);
  *(_BYTE *)sub_180026D58((__int64)&v6, *(unsigned __int16 *)(v8 + 8)) = 0;
  v4 = sub_180026CB4((__int64)&v6);
  sub_1800032E0(v7, v4);
  sub_180026C64(&v6);
  return v7;
}
```

8cb78327bd69fda61afac9393187ad5533a63d43ebf74c0f9800bedb814b20ad

静态信息

DLL x64



Exeinfo PE - ver.0.0.4.7 by A.S.L - 992+60 sign 2017.06.05

File : 8cb78327bd69fda61afac9393187ad5533a63d43ebf74c0f98

Entry Point : 00029AC0 oo < EP Section : .text

File Offset : 00028EC0 First Bytes : 48.89.5C.24.08

Linker Info : 10.00 SubSystem : Win Console

File Size : 0012E800h < N Overlay : NO 00000000

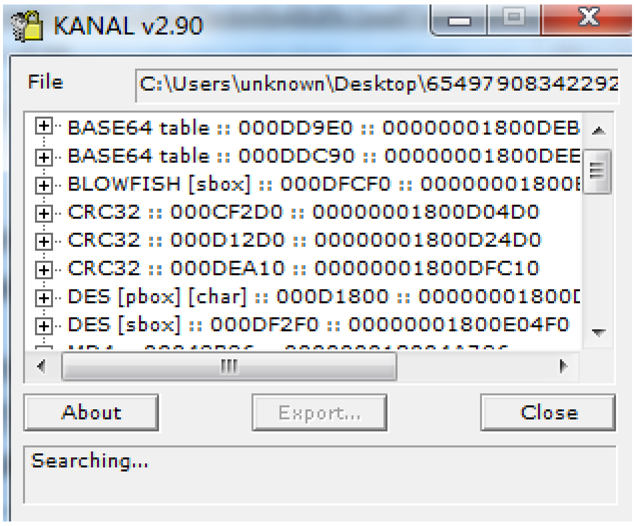
64 bit DLL library RES/OVL : 0 / 0 % 2006

Microsoft Visual C++ v10.0 - 200x - DLL - (x64) www.microsoft.com

Lamer Info - Help Hint - Unpack info 47 ms.

Not packed , try OllyDbg v2 - www.ollydbg.de or IDA v5 www.hex-ray

可能存在的加密算法：



KANAL v2.90

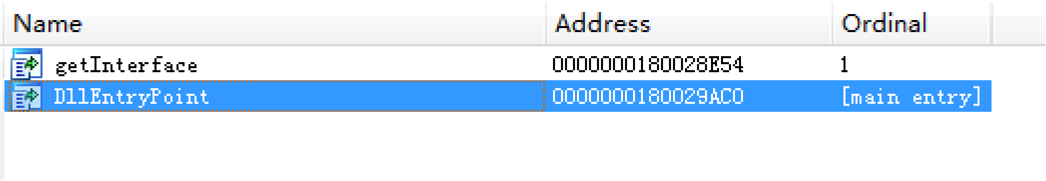
File C:\Users\unknown\Desktop\65497908342292

```
BASE64 table :: 000DD9E0 :: 00000001800DEB
BASE64 table :: 000DDC90 :: 00000001800DEE
BLOWFISH [sbox] :: 000DFCF0 :: 000000018001
CRC32 :: 000CF2D0 :: 00000001800D04D0
CRC32 :: 000D12D0 :: 00000001800D24D0
CRC32 :: 000DEA10 :: 00000001800DFC10
DES [pbox] [char] :: 000D1800 :: 00000001800D
DES [sbox] :: 000DF2F0 :: 00000001800E04F0
```

Searching...

代码分析

本样本导出函数如下：



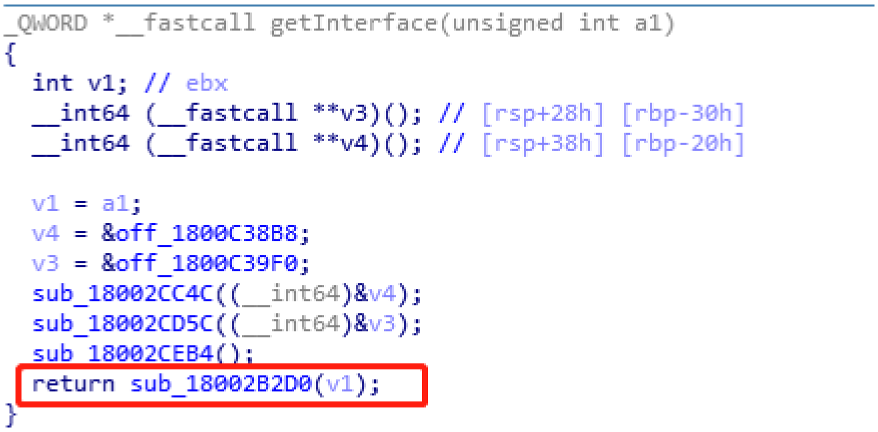
Name	Address	Ordinal
getInterface	0000000180028E54	1
DllEntryPoint	0000000180029AC0	[main entry]

getInterface

根据分析，函数的参数 a1 可以为下列表格中的值。在本函数中，

sub_18002B2D0()是主要的功能函数，在 sub_18002B2D0()函数内，会通过 if 结构对参数选择相应的操作。由于存在多处函数指针的动态调用，所以本函数的功能尚不明确：

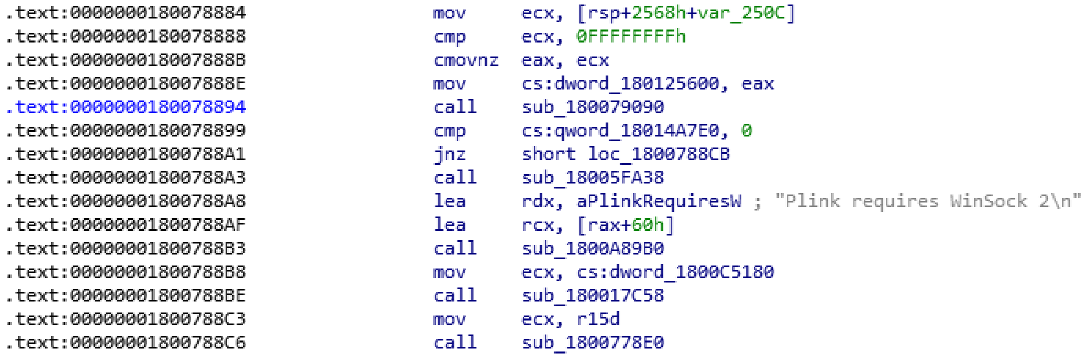
序号	参数值
1	0x2710
2	0x2774
3	0x27D8



```
_QWORD *__fastcall getInterface(unsigned int a1)
{
  int v1; // ebx
  __int64 (__fastcall **v3)(); // [rsp+28h] [rbp-30h]
  __int64 (__fastcall **v4)(); // [rsp+38h] [rbp-20h]

  v1 = a1;
  v4 = &off_1800C38B8;
  v3 = &off_1800C39F0;
  sub_18002CC4C((__int64)&v4);
  sub_18002CD5C((__int64)&v3);
  sub_18002CEB4();
  return sub_18002B2D0(v1);
}
```

·使用 Plink

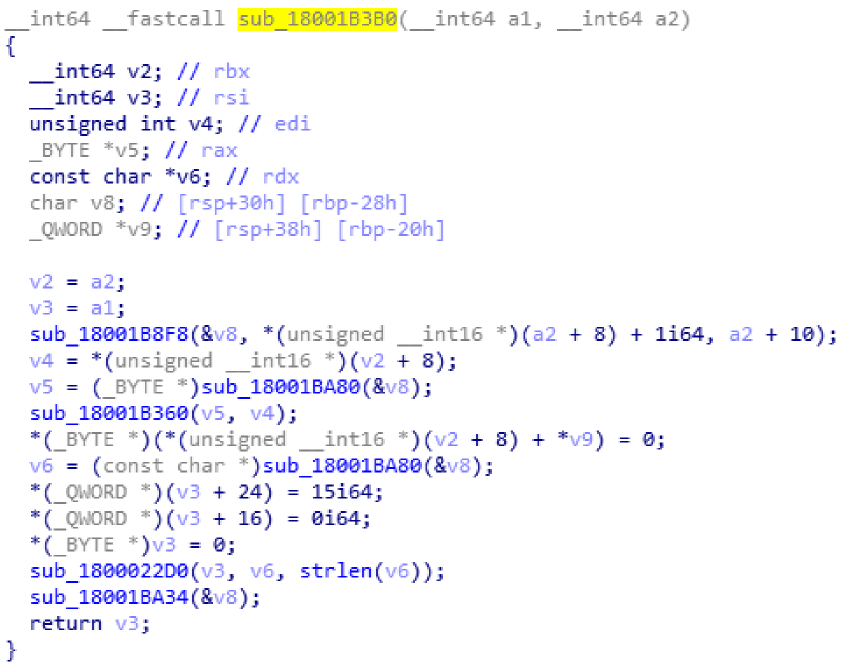
在分析过程中，发现本样本使用了 Plink 部分源码：



```
.text:0000000180078884                 mov     ecx, [rsp+2568h+var_250C]
.text:0000000180078888                 cmp     ecx, 0FFFFFFFFh
.text:000000018007888B                 cmovnz  eax, ecx
.text:000000018007888E                 mov     cs:dword_180125600, eax
.text:0000000180078894                 call    sub_180079090
.text:0000000180078899                 cmp     cs:qword_18014A7E0, 0
.text:00000001800788A1                 jnz     short loc_1800788CB
.text:00000001800788A3                 call    sub_18005FA38
.text:00000001800788A8                 lea     rdx, aPlinkRequiresW ; "Plink requires WinSock 2\n"
.text:00000001800788AF                 lea     rcx, [rax+60h]
.text:00000001800788B3                 call    sub_1800A89B0
.text:00000001800788B8                 mov     ecx, cs:dword_1800C5180
.text:00000001800788BE                 call    sub_180017C58
.text:00000001800788C3                 mov     ecx, r15d
.text:00000001800788C6                 call    sub_1800778E0
```

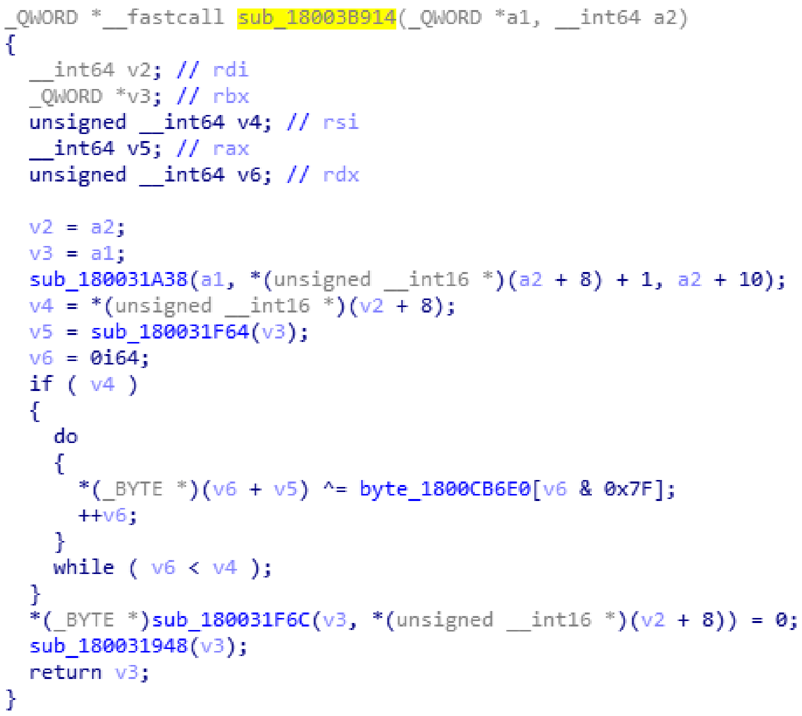
解密函数

在本样本分析过程中，共发现以下两个解密函数，这些解密函数在代码执行过程中，被用来解密被加密的字符串：



```
__int64 __fastcall sub_18001B3B0(__int64 a1, __int64 a2)
{
  __int64 v2; // rbx
  __int64 v3; // rsi
  unsigned int v4; // edi
  _BYTE *v5; // rax
  const char *v6; // rdx
  char v8; // [rsp+30h] [rbp-28h]
  _QWORD *v9; // [rsp+38h] [rbp-20h]

  v2 = a2;
  v3 = a1;
  sub_18001B8F8(&v8, *(unsigned __int16 *)(a2 + 8) + 1i64, a2 + 10);
  v4 = *(unsigned __int16 *)(v2 + 8);
  v5 = (_BYTE *)sub_18001BA80(&v8);
  sub_18001B360(v5, v4);
  *(_BYTE *)(*(unsigned __int16 *)(v2 + 8) + *v9) = 0;
  v6 = (const char *)sub_18001BA80(&v8);
  *(_QWORD *)(v3 + 24) = 15i64;
  *(_QWORD *)(v3 + 16) = 0i64;
  *(_BYTE *)v3 = 0;
  sub_1800022D0(v3, v6, strlen(v6));
  sub_18001BA34(&v8);
  return v3;
}
```

解密函数 1



```
_QWORD *__fastcall sub_18003B914(_QWORD *a1, __int64 a2)
{
  __int64 v2; // rdi
  _QWORD *v3; // rbx
  unsigned __int64 v4; // rsi
  __int64 v5; // rax
  unsigned __int64 v6; // rdx

  v2 = a2;
  v3 = a1;
  sub_180031A38(a1, *(unsigned __int16 *)(a2 + 8) + 1, a2 + 10);
  v4 = *(unsigned __int16 *)(v2 + 8);
  v5 = sub_180031F64(v3);
  v6 = 0i64;
  if ( v4 )
  {
    do
    {
      *(_BYTE *)(v6 + v5) ^= byte_1800CB6E0[v6 & 0x7F];
      ++v6;
    }
    while ( v6 < v4 );
  }
  *(_BYTE *)sub_180031F6C(v3, *(unsigned __int16 *)(v2 + 8)) = 0;
  sub_180031948(v3);
  return v3;
}
```

解密函数 2

友商公开的 Python 解密代码如下

解密函数 1

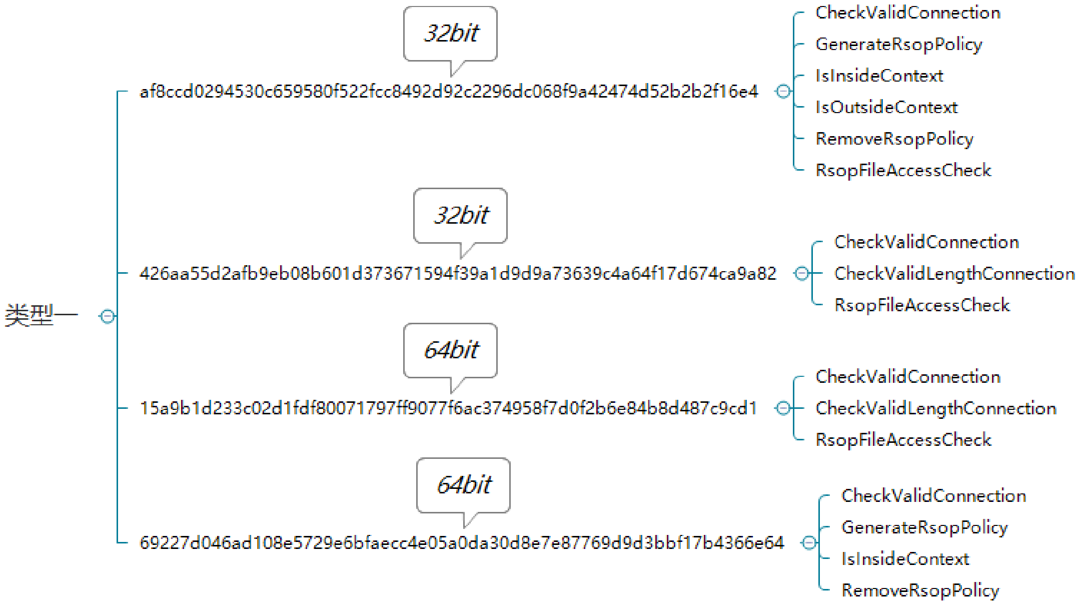
```
def DecodeMethod1(indata, r_start, r_length):
    dec_data = ""
    enc_data = indata[r_start:]
    dec_len = ord(indata[r_length])
    for index, byte in enumerate(enc_data[:dec_len]):
        eax = (
            (((dec_len - index) -1) ^ 0x1D) * ((dec_len - index) +
0x10)
            ) & 0xFFFFFFFF
        eax += 0x1000193
        cl = ( ((eax >> 0x18) & 0xFF) ^ ((eax >> 0x10) & 0xFF) )
        cl = ( cl ^ ((eax >> 0x8) & 0xFF) )
        cl = ( cl ^ ord(byte)) & 0xFF
        cl = ( cl ^ (eax & 0xFF) )
        dec_data += chr(cl)
    return dec_data
```

解密函数 2

```
def DecodeMethod2(indata, key, r_start, r_length):
    enc_data = indata[r_start:]
    dec_length = ord(indata[r_length])
    dec_data = ""
    for index, byte in enumerate(enc_data[:dec_length]):
        if ord(enc_data[index]) == 0 and ord(enc_data[index+1]) ==
0: break
    dec_data += chr( ord(byte) ^ ord(key[index % len(key)]) )
    return dec_data
```

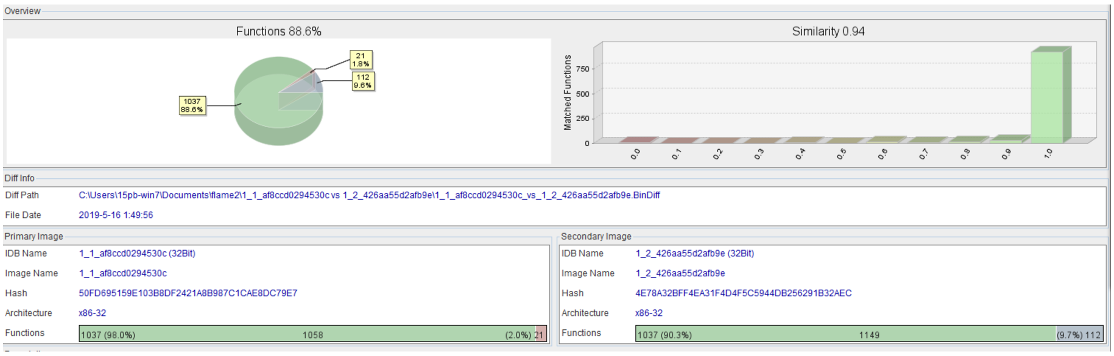
四、相似度分析

类型一


类型一
32bit
af8ccd0294530c659580f522fcc8492d92c2296dc068f9a42474d52b2b2f16e4
CheckValidConnection
GenerateRsopPolicy
IsInsideContext
IsOutsideContext
RemoveRsopPolicy
RsopFileAccessCheck
32bit
426aa55d2afb9eb08b601d373671594f39a1d9d9a73639c4a64f17d674ca9a82
CheckValidConnection
CheckValidLengthConnection
RsopFileAccessCheck
64bit
15a9b1d233c02d1fdf80071797ff9077f6ac374958f7d0f2b6e84b8d487c9cd1
CheckValidConnection
CheckValidLengthConnection
RsopFileAccessCheck
64bit
69227d046ad108e5729e6bfaecc4e05a0da30d8e7e87769d9d3bbf17b4366e64
CheckValidConnection
GenerateRsopPolicy
IsInsideContext
RemoveRsopPolicy

由于样本类型一的导出函数中，均对 CheckValidConnection()函数进行了调用，这里主要分析 CheckValidConnection()函数的相似度情况。

样本 1 VS 样本 2



Diff Info

Diff Path C:\Users\15pb-win7\Documents\flame2\1_1_af8ccd0294530c vs 1_2_426aa55d2afb9e\1_1_af8ccd0294530c_vs_1_2_426aa55d2afb9e.BinDiff

File Date 2019-5-16 1:49:56

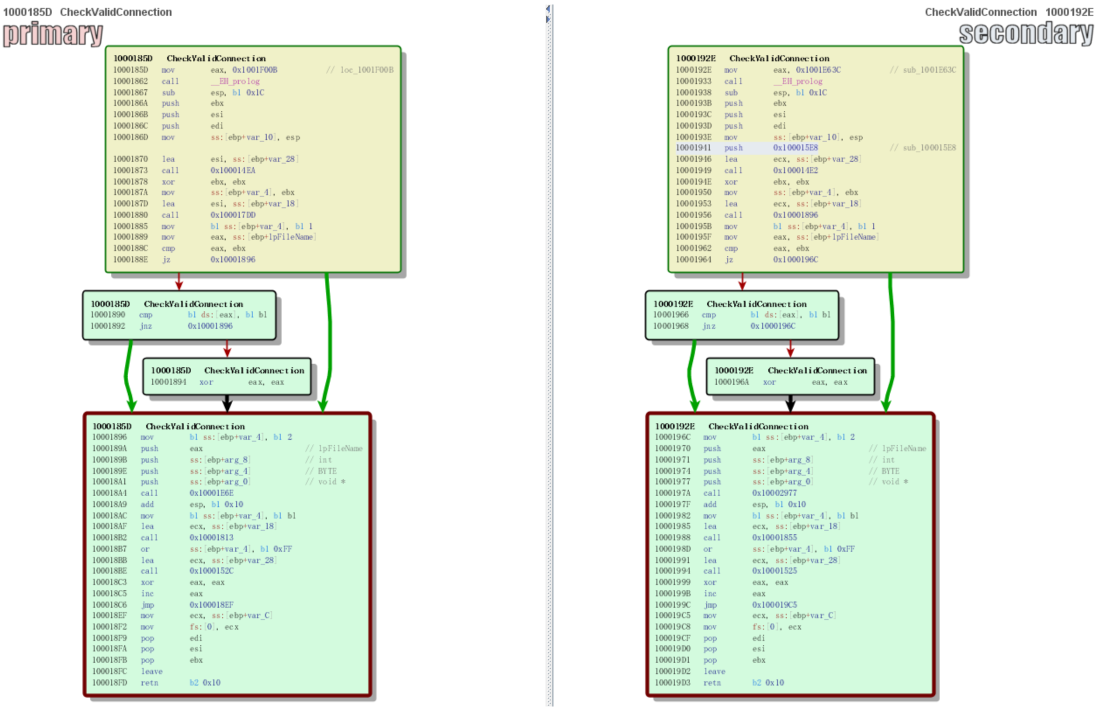
Primary Image

IDB Name	1_1_af8ccd0294530c (32Bit)
Image Name	1_1_af8ccd0294530c
Hash	50FD695159E103B8DF2421A8B987C1CAE8DC79E7
Architecture	x86-32
Functions	1037 (98.0%) 1058 (2.0%) 21

Secondary Image

IDB Name	1_2_426aa55d2afb9e (32Bit)
Image Name	1_2_426aa55d2afb9e
Hash	4E78A32BFF4EA31F4D4F5C5944DB256291B32AEC
Architecture	x86-32
Functions	1037 (90.3%) 1149 (9.7%) 112

代码相似度达到 0.95

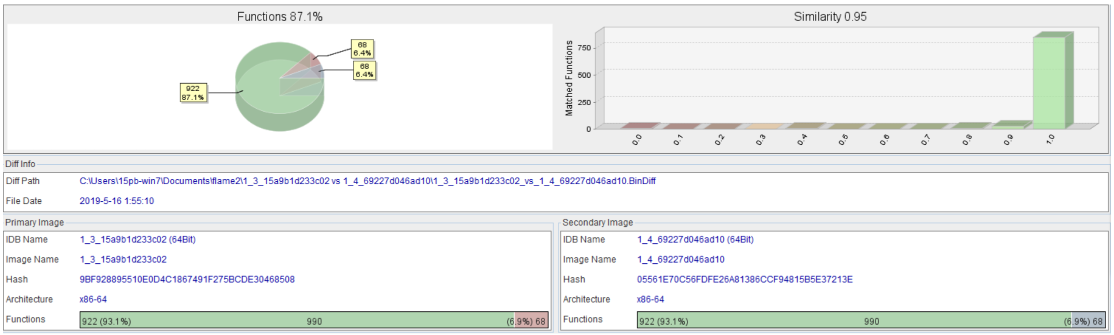


```
1000185D   CheckValidConnection
1000185D   mov     eax, 0x1001F00B                 // loc_1001F00B
10001862   call    __EH_prolog
10001867   sub     esp, bl 0x1C
1000186A   push    ebx
1000186B   push    esi
1000186C   push    edi
1000186D   mov     ss:[ebp+var_10], esp

10001870   lea     esi, ss:[ebp+var_28]
10001873   call    0x100014EA
10001878   xor     ebx, ebx
1000187A   mov     ss:[ebp+var_4], ebx
1000187D   lea     esi, ss:[ebp+var_18]
10001880   call    0x100017DD
10001885   mov     bl ss:[ebp+var_4], bl 1
10001889   mov     eax, ss:[ebp+lpFileName]
1000188C   cmp     eax, ebx
1000188E   jz      0x10001896

1000185D   CheckValidConnection
10001890   cmp     bl ds:[eax], bl bl
10001892   jnz     0x10001896

1000185D   CheckValidConnection
10001894   xor     eax, eax

1000185D   CheckValidConnection
10001896   mov     bl ss:[ebp+var_4], bl 2
1000189A   push    eax                             // lpFileName
1000189B   push    ss:[ebp+arg_8]                  // int
1000189E   push    ss:[ebp+arg_4]                  // BYTE
100018A1   push    ss:[ebp+arg_0]                  // void *
100018A4   call    0x10001E6E
100018A9   add     esp, bl 0x10
100018AC   mov     bl ss:[ebp+var_4], bl bl
100018AF   lea     ecx, ss:[ebp+var_18]
100018B2   call    0x10001813
100018B7   or      ss:[ebp+var_4], bl 0xFF
100018BB   lea     ecx, ss:[ebp+var_28]
100018BE   call    0x1000152C
100018C3   xor     eax, eax
100018C5   inc     eax
100018C6   jmp     0x100018EF
100018EF   mov     ecx, ss:[ebp+var_C]
100018F2   mov     fs:[0], ecx
100018F9   pop     edi
100018FA   pop     esi
100018FB   pop     ebx
100018FC   leave
100018FD   retn    b2 0x10
```

```
1000192E   CheckValidConnection
1000192E   mov     eax, 0x1001E63C                 // sub_1001E63C
10001933   call    __EH_prolog
10001938   sub     esp, bl 0x1C
1000193B   push    ebx
1000193C   push    esi
1000193D   push    edi
1000193E   mov     ss:[ebp+var_10], esp
10001941   push    0x100015E8                      // sub_100015E8
10001946   lea     ecx, ss:[ebp+var_28]
10001949   call    0x100014E2
1000194E   xor     ebx, ebx
10001950   mov     ss:[ebp+var_4], ebx
10001953   lea     ecx, ss:[ebp+var_18]
10001956   call    0x10001896
1000195B   mov     bl ss:[ebp+var_4], bl 1
1000195F   mov     eax, ss:[ebp+lpFileName]
10001962   cmp     eax, ebx
10001964   jz      0x1000196C

1000192E   CheckValidConnection
10001966   cmp     bl ds:[eax], bl bl
10001968   jnz     0x1000196C

1000192E   CheckValidConnection
1000196A   xor     eax, eax

1000192E   CheckValidConnection
1000196C   mov     bl ss:[ebp+var_4], bl 2
10001970   push    eax                             // lpFileName
10001971   push    ss:[ebp+arg_8]                  // int
10001974   push    ss:[ebp+arg_4]                  // BYTE
10001977   push    ss:[ebp+arg_0]                  // void *
1000197A   call    0x10002977
1000197F   add     esp, bl 0x10
10001982   mov     bl ss:[ebp+var_4], bl bl
10001985   lea     ecx, ss:[ebp+var_18]
10001988   call    0x10001855
1000198D   or      ss:[ebp+var_4], bl 0xFF
10001991   lea     ecx, ss:[ebp+var_28]
10001994   call    0x10001525
10001999   xor     eax, eax
1000199B   inc     eax
1000199C   jmp     0x100019C5
100019C5   mov     ecx, ss:[ebp+var_C]
100019C8   mov     fs:[0], ecx
100019CF   pop     edi
100019D0   pop     esi
100019D1   pop     ebx
100019D2   leave
100019D3   retn    b2 0x10
```

CheckValidConnection 函数代码一致

样本 3 VS 样本 4



Diff Info

Diff Path C:\Users\15pb-win7\Documents\flame2\1_3_15a9b1d233c02 vs 1_4_69227d046ad10\1_3_15a9b1d233c02_vs_1_4_69227d046ad10.BinDiff

File Date 2019-5-16 1:55:10

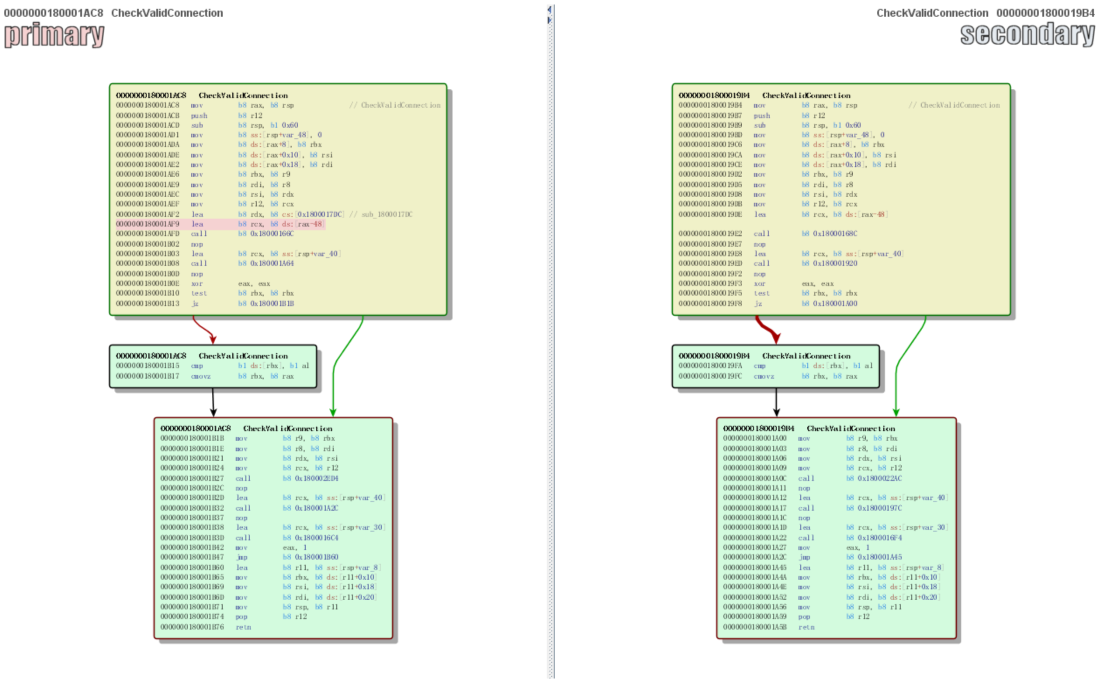
Primary Image

IDB Name	1_3_15a9b1d233c02 (64Bit)
Image Name	1_3_15a9b1d233c02
Hash	9BF928895510E0D4C1867491F275BCDE30468508
Architecture	x86-64
Functions	922 (93.1%) 990 (6.9%) 68

Secondary Image

IDB Name	1_4_69227d046ad10 (64Bit)
Image Name	1_4_69227d046ad10
Hash	05561E70C56FDFE26A81386CCF94815B5E37213E
Architecture	x86-64
Functions	922 (93.1%) 990 (6.9%) 68

代码相似度达到 0.95


0000000180001AC8 CheckValidConnection
primary
CheckValidConnection 00000001800019B4
secondary

```
0000000180001AC8   CheckValidConnection
0000000180001AC8   mov     b8 rax, b8 rsp                   // CheckValidConnection
0000000180001ACB   push    b8 r12
0000000180001ACD   sub     b8 rsp, b1 0x60
0000000180001AD1   mov     b8 ss:[rsp+var_48], 0
0000000180001ADA   mov     b8 ds:[rax+8], b8 rbx
0000000180001ADE   mov     b8 ds:[rax+0x10], b8 rsi
0000000180001AE2   mov     b8 ds:[rax+0x18], b8 rdi
0000000180001AE6   mov     b8 rbx, b8 r9
0000000180001AE9   mov     b8 rdi, b8 r8
0000000180001AEC   mov     b8 rsi, b8 rdx
0000000180001AEF   mov     b8 r12, b8 rcx
0000000180001AF2   lea     b8 rdx, b8 cs:[0x1800017DC] // sub_1800017DC
0000000180001AF9   lea     b8 rcx, b8 ds:[rax-48]
0000000180001AFD   call    b8 0x18000166C
0000000180001B02   nop
0000000180001B03   lea     b8 rcx, b8 ss:[rsp+var_40]
0000000180001B08   call    b8 0x180001A64
0000000180001B0D   nop
0000000180001B0E   xor     eax, eax
0000000180001B10   test    b8 rbx, b8 rbx
0000000180001B13   jz      b8 0x180001B1B
```

```
0000000180001AC8   CheckValidConnection
0000000180001B15   cmp     b1 ds:[rbx], b1 al
0000000180001B17   cmovz   b8 rbx, b8 rax
```

```
0000000180001AC8   CheckValidConnection
0000000180001B1B   mov     b8 r9, b8 rbx
0000000180001B1E   mov     b8 r8, b8 rdi
0000000180001B21   mov     b8 rdx, b8 rsi
0000000180001B24   mov     b8 rcx, b8 r12
0000000180001B27   call    b8 0x180002ED4
0000000180001B2C   nop
0000000180001B2D   lea     b8 rcx, b8 ss:[rsp+var_40]
0000000180001B32   call    b8 0x180001A2C
0000000180001B37   nop
0000000180001B38   lea     b8 rcx, b8 ss:[rsp+var_30]
0000000180001B3D   call    b8 0x1800016C4
0000000180001B42   mov     eax, 1
0000000180001B47   jmp     b8 0x180001B60
0000000180001B60   lea     b8 r11, b8 ss:[rsp+var_8]
0000000180001B65   mov     b8 rbx, b8 ds:[r11+0x10]
0000000180001B69   mov     b8 rsi, b8 ds:[r11+0x18]
0000000180001B6D   mov     b8 rdi, b8 ds:[r11+0x20]
0000000180001B71   mov     b8 rsp, b8 r11
0000000180001B74   pop     b8 r12
0000000180001B76   retn
```

```
00000001800019B4   CheckValidConnection
00000001800019B4   mov     b8 rax, b8 rsp                   // CheckValidConnection
00000001800019B7   push    b8 r12
00000001800019B9   sub     b8 rsp, b1 0x60
00000001800019BD   mov     b8 ss:[rsp+var_48], 0
00000001800019C6   mov     b8 ds:[rax+8], b8 rbx
00000001800019CA   mov     b8 ds:[rax+0x10], b8 rsi
00000001800019CE   mov     b8 ds:[rax+0x18], b8 rdi
00000001800019D2   mov     b8 rbx, b8 r9
00000001800019D5   mov     b8 rdi, b8 r8
00000001800019D8   mov     b8 rsi, b8 rdx
00000001800019DB   mov     b8 r12, b8 rcx
00000001800019DE   lea     b8 rcx, b8 ds:[rax-48]

00000001800019E2   call    b8 0x18000168C
00000001800019E7   nop
00000001800019E8   lea     b8 rcx, b8 ss:[rsp+var_40]
00000001800019ED   call    b8 0x180001920
00000001800019F2   nop
00000001800019F3   xor     eax, eax
00000001800019F5   test    b8 rbx, b8 rbx
00000001800019F8   jz      b8 0x180001A00
```

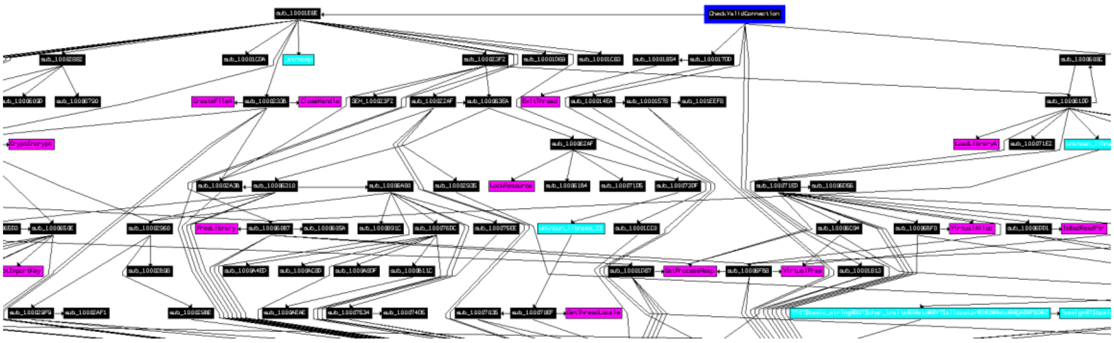
```
00000001800019B4   CheckValidConnection
00000001800019FA   cmp     b1 ds:[rbx], b1 al
00000001800019FC   cmovz   b8 rbx, b8 rax
```

```
00000001800019B4   CheckValidConnection
0000000180001A00   mov     b8 r9, b8 rbx
0000000180001A03   mov     b8 r8, b8 rdi
0000000180001A06   mov     b8 rdx, b8 rsi
0000000180001A09   mov     b8 rcx, b8 r12
0000000180001A0C   call    b8 0x1800022AC
0000000180001A11   nop
0000000180001A12   lea     b8 rcx, b8 ss:[rsp+var_40]
0000000180001A17   call    b8 0x18000197C
0000000180001A1C   nop
0000000180001A1D   lea     b8 rcx, b8 ss:[rsp+var_30]
0000000180001A22   call    b8 0x1800016F4
0000000180001A27   mov     eax, 1
0000000180001A2C   jmp     b8 0x180001A45
0000000180001A45   lea     b8 r11, b8 ss:[rsp+var_8]
0000000180001A4A   mov     b8 rbx, b8 ds:[r11+0x10]
0000000180001A4E   mov     b8 rsi, b8 ds:[r11+0x18]
0000000180001A52   mov     b8 rdi, b8 ds:[r11+0x20]
0000000180001A56   mov     b8 rsp, b8 r11
0000000180001A59   pop     b8 r12
0000000180001A5B   retn
```

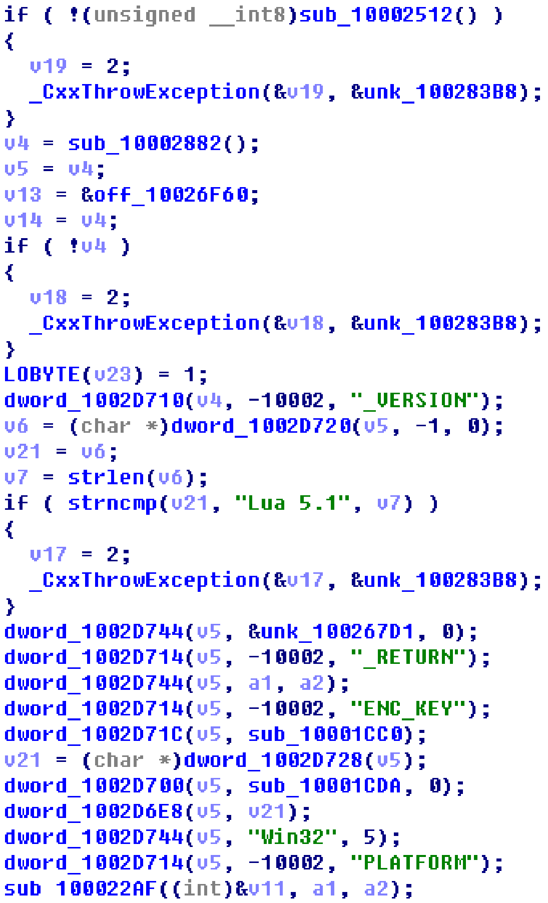
CheckValidConnection 函数代码一致

样本 1 VS 样本 3

样本 1 为 32 位，样本 3 为 64 位，通过函数执行流程和关键 API 可以确定为功能基本一致

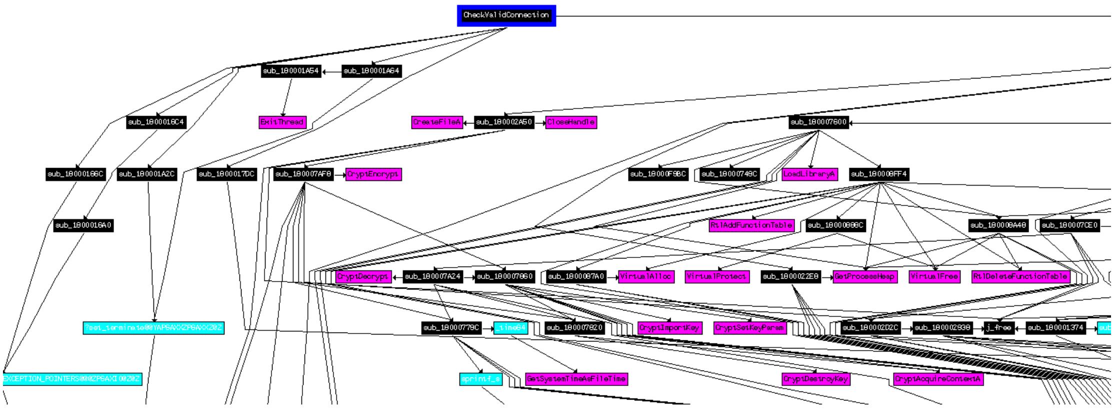


样本 1 部分 API 调用

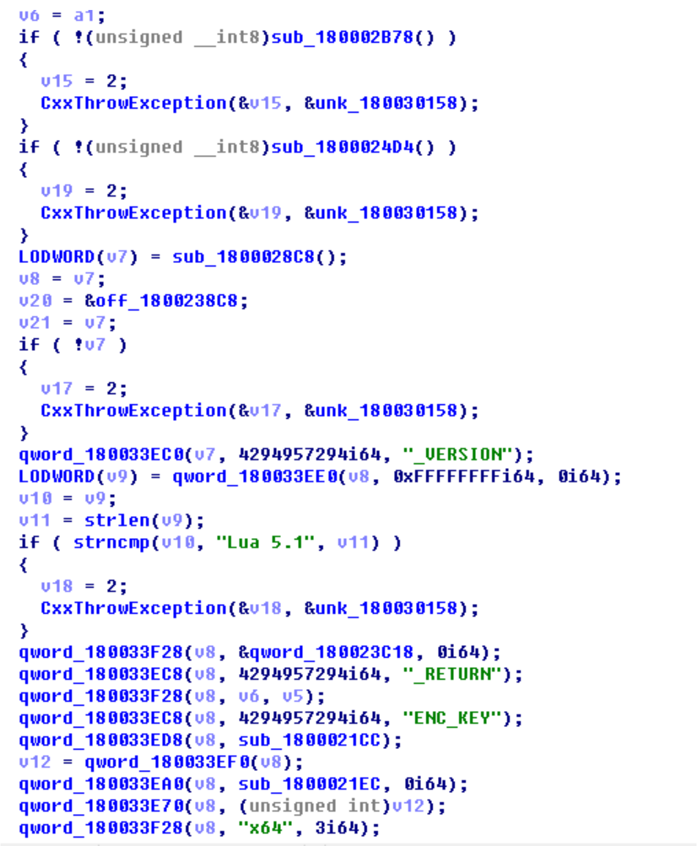


```
if ( !(unsigned __int8)sub_10002512() )
{
  v19 = 2;
  _CxxThrowException(&v19, &unk_100283B8);
}
v4 = sub_10002882();
v5 = v4;
v13 = &off_10026F60;
v14 = v4;
if ( !v4 )
{
  v18 = 2;
  _CxxThrowException(&v18, &unk_100283B8);
}
LOBYTE(v23) = 1;
dword_1002D710(v4, -10002, "_VERSION");
v6 = (char *)dword_1002D720(v5, -1, 0);
v21 = v6;
v7 = strlen(v6);
if ( strncmp(v21, "Lua 5.1", v7) )
{
  v17 = 2;
  _CxxThrowException(&v17, &unk_100283B8);
}
dword_1002D744(v5, &unk_100267D1, 0);
dword_1002D714(v5, -10002, "_RETURN");
dword_1002D744(v5, a1, a2);
dword_1002D714(v5, -10002, "ENC_KEY");
dword_1002D71C(v5, sub_10001CC0);
v21 = (char *)dword_1002D728(v5);
dword_1002D700(v5, sub_10001CDA, 0);
dword_1002D6E8(v5, v21);
dword_1002D744(v5, "Win32", 5);
dword_1002D714(v5, -10002, "PLATFORM");
sub_100022AF((int)&v11, a1, a2);
```

样本 1 部分代码



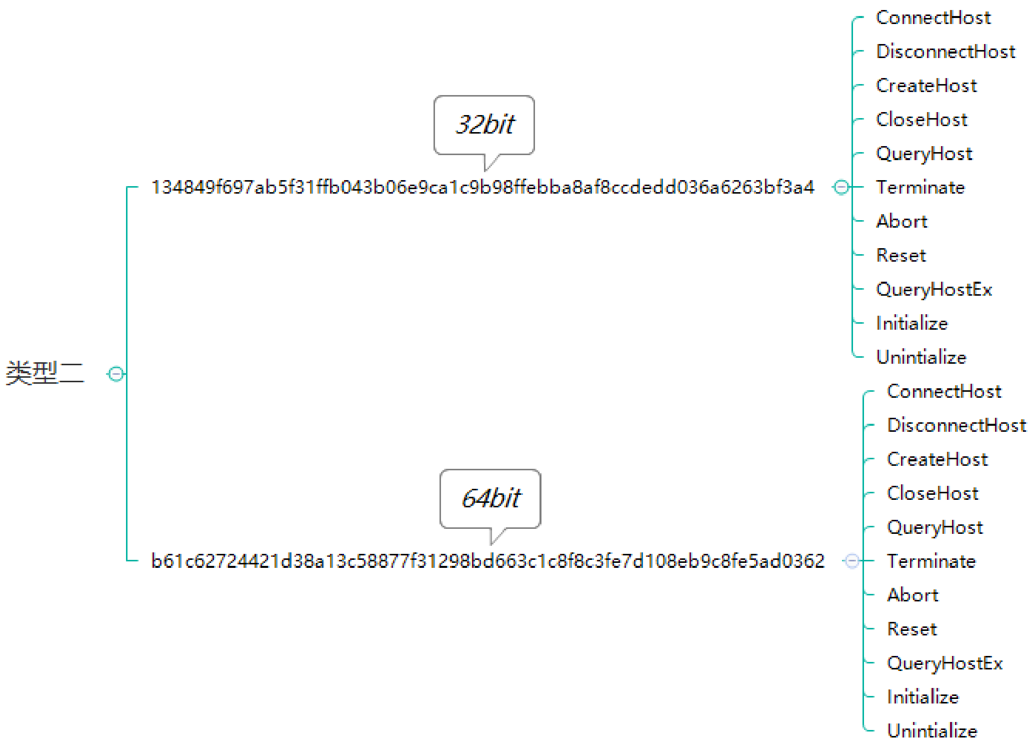
样本 3 部分 API 调用



```
v6 = a1;
if ( !(unsigned __int8)sub_180002B78() )
{
  v15 = 2;
  CxxThrowException(&v15, &unk_180030158);
}
if ( !(unsigned __int8)sub_1800024D4() )
{
  v19 = 2;
  CxxThrowException(&v19, &unk_180030158);
}
LODWORD(v7) = sub_1800028C8();
v8 = v7;
v20 = &off_1800238C8;
v21 = v7;
if ( !v7 )
{
  v17 = 2;
  CxxThrowException(&v17, &unk_180030158);
}
qword_180033EC0(v7, 4294957294i64, "_VERSION");
LODWORD(v9) = qword_180033EE0(v8, 0xFFFFFFFFi64, 0i64);
v10 = v9;
v11 = strlen(v9);
if ( strncmp(v10, "Lua 5.1", v11) )
{
  v18 = 2;
  CxxThrowException(&v18, &unk_180030158);
}
qword_180033F28(v8, &qword_180023C18, 0i64);
qword_180033EC8(v8, 4294957294i64, "_RETURN");
qword_180033F28(v8, v6, v5);
qword_180033EC8(v8, 4294957294i64, "ENC_KEY");
qword_180033ED8(v8, sub_1800021CC);
v12 = qword_180033EF0(v8);
qword_180033EA0(v8, sub_1800021EC, 0i64);
qword_180033E70(v8, (unsigned int)v12);
qword_180033F28(v8, "x64", 3i64);
```

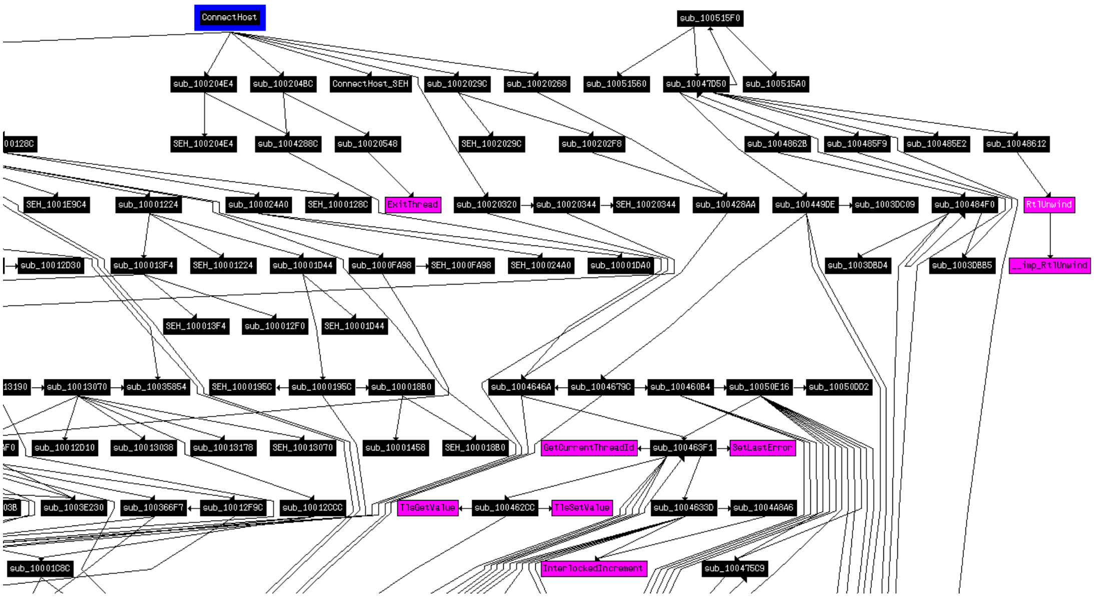
样本 3 部分代码

类型二

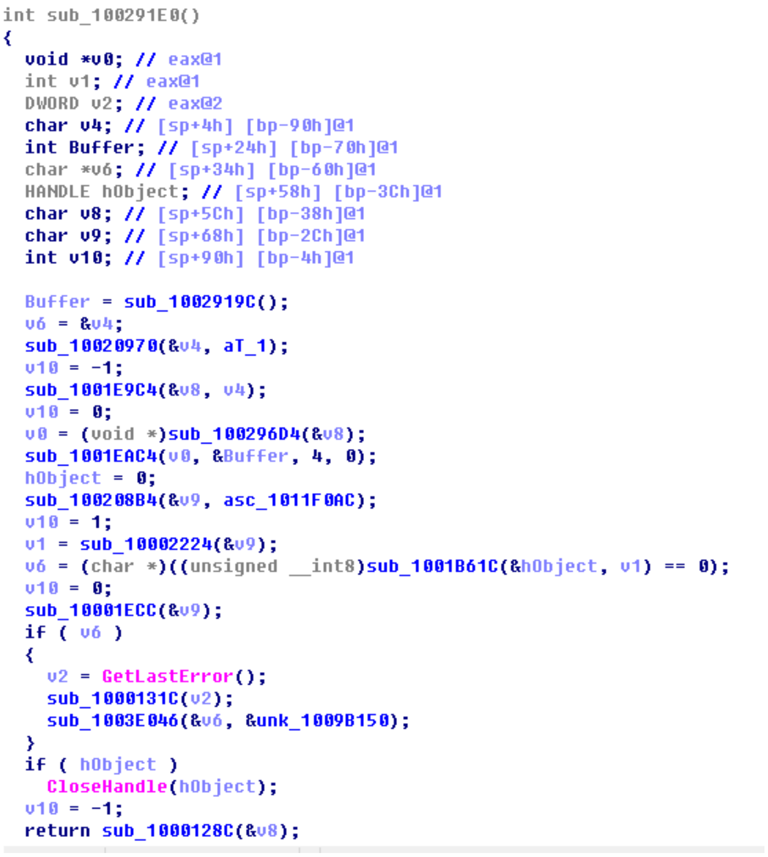


样本 5 为 32 位，样本 6 为 64 位，经对比函数流程与 API 调用，功能一致

样本 5 VS 样本 6

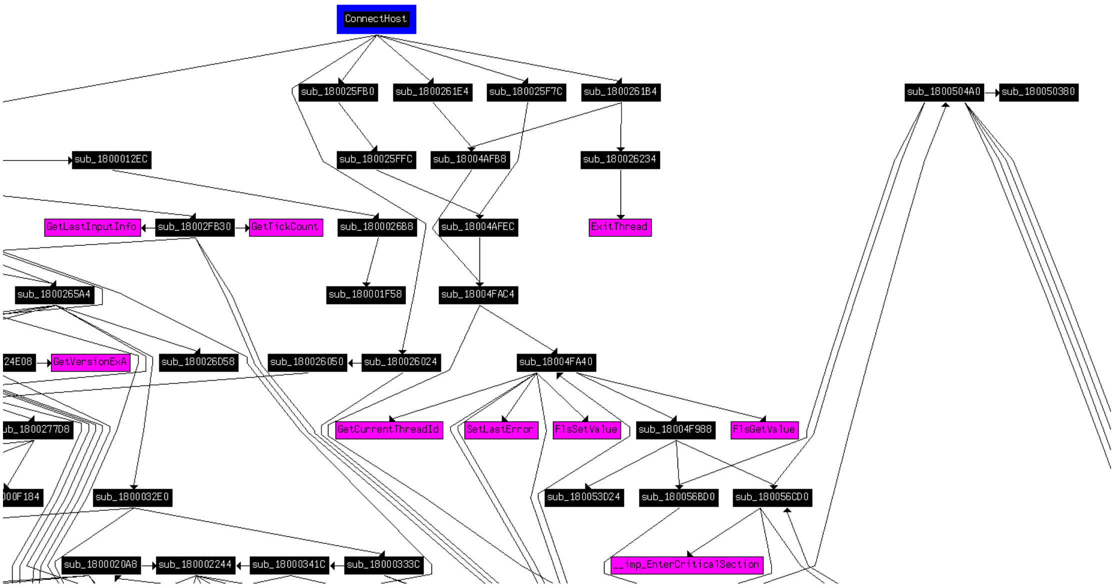


样本 5 部分 API 调用

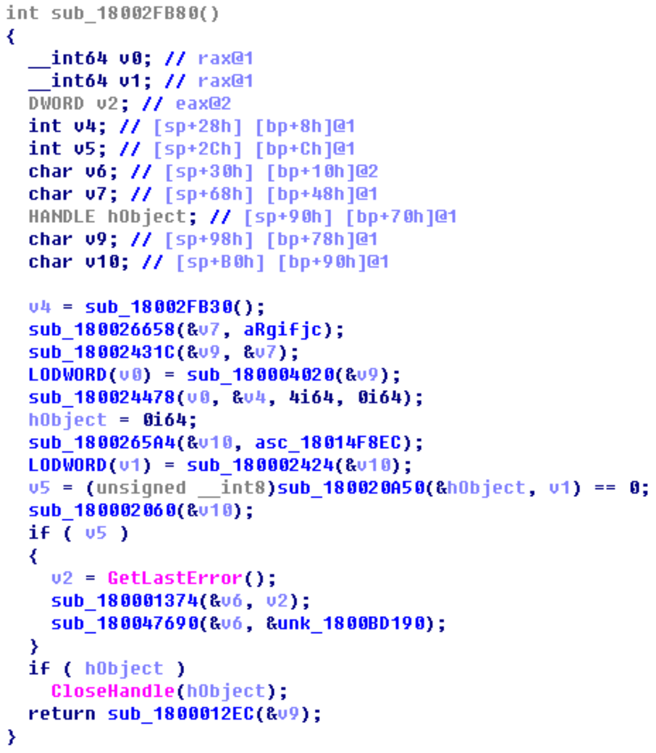


```
int sub_100291E0()
{
  void *v0; // eax@1
  int v1; // eax@1
  DWORD v2; // eax@2
  char v4; // [sp+4h] [bp-90h]@1
  int Buffer; // [sp+24h] [bp-70h]@1
  char *v6; // [sp+34h] [bp-60h]@1
  HANDLE hObject; // [sp+58h] [bp-3Ch]@1
  char v8; // [sp+5Ch] [bp-38h]@1
  char v9; // [sp+68h] [bp-2Ch]@1
  int v10; // [sp+90h] [bp-4h]@1

  Buffer = sub_1002919C();
  v6 = &v4;
  sub_10020970(&v4, aT_1);
  v10 = -1;
  sub_1001E9C4(&v8, v4);
  v10 = 0;
  v0 = (void *)sub_100296D4(&v8);
  sub_1001EAC4(v0, &Buffer, 4, 0);
  hObject = 0;
  sub_100208B4(&v9, asc_1011F0AC);
  v10 = 1;
  v1 = sub_10002224(&v9);
  v6 = (char *)((unsigned __int8)sub_1001B61C(&hObject, v1) == 0);
  v10 = 0;
  sub_10001ECC(&v9);
  if ( v6 )
  {
    v2 = GetLastError();
    sub_1000131C(v2);
    sub_1003E046(&v6, &unk_1009B150);
  }
  if ( hObject )
    CloseHandle(hObject);
  v10 = -1;
  return sub_1000128C(&v8);
```

样本 5 部分代码



样本 6 部分 API 调用

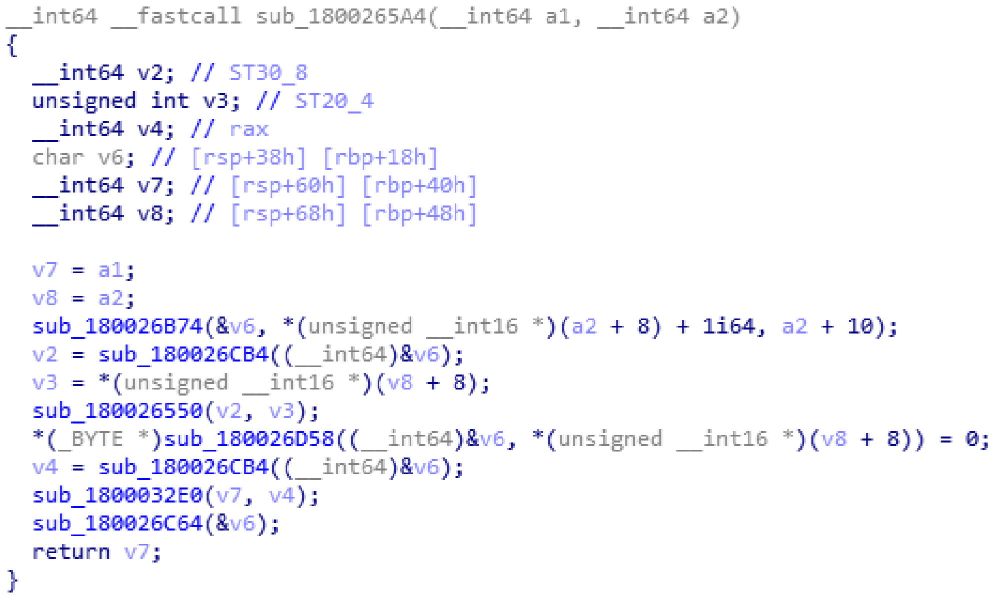


```
int sub_18002FB80()
{
  __int64 v0; // rax@1
  __int64 v1; // rax@1
  DWORD v2; // eax@2
  int v4; // [sp+28h] [bp+8h]@1
  int v5; // [sp+2Ch] [bp+Ch]@1
  char v6; // [sp+30h] [bp+10h]@2
  char v7; // [sp+68h] [bp+48h]@1
  HANDLE hObject; // [sp+90h] [bp+70h]@1
  char v9; // [sp+98h] [bp+78h]@1
  char v10; // [sp+B0h] [bp+90h]@1

  v4 = sub_18002FB30();
  sub_180026658(&v7, aRgifjc);
  sub_18002431C(&v9, &v7);
  LODWORD(v0) = sub_180004020(&v9);
  sub_180024478(v0, &v4, 4i64, 0i64);
  hObject = 0i64;
  sub_1800265A4(&v10, asc_18014F8EC);
  LODWORD(v1) = sub_180002424(&v10);
  v5 = (unsigned __int8)sub_180020A50(&hObject, v1) == 0;
  sub_180002060(&v10);
  if ( v5 )
  {
    v2 = GetLastError();
    sub_180001374(&v6, v2);
    sub_180047690(&v6, &unk_1800BD190);
  }
  if ( hObject )
    CloseHandle(hObject);
  return sub_1800012EC(&v9);
}
```

样本 6 部分代码

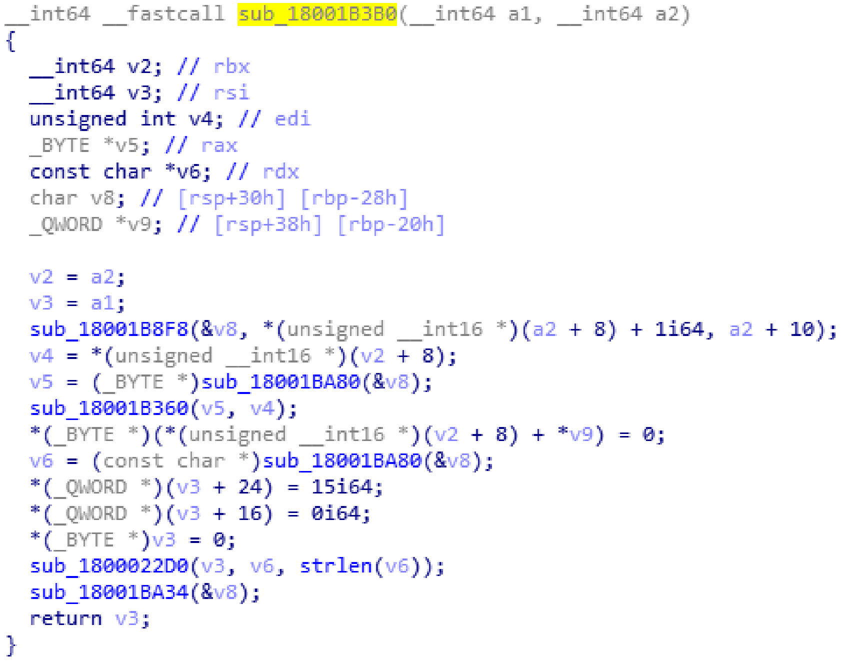
样本 6 VS 样本 8

在样本 6 和样本 8 中发现了同样的解密代码，因此样本类型二与类型三之间也存在关联



```
__int64 __fastcall sub_1800265A4(__int64 a1, __int64 a2)
{
  __int64 v2; // ST30_8
  unsigned int v3; // ST20_4
  __int64 v4; // rax
  char v6; // [rsp+38h] [rbp+18h]
  __int64 v7; // [rsp+60h] [rbp+40h]
  __int64 v8; // [rsp+68h] [rbp+48h]

  v7 = a1;
  v8 = a2;
  sub_180026B74(&v6, *(unsigned __int16 *)(a2 + 8) + 1i64, a2 + 10);
  v2 = sub_180026CB4((__int64)&v6);
  v3 = *(unsigned __int16 *)(v8 + 8);
  sub_180026550(v2, v3);
  *(_BYTE *)sub_180026D58((__int64)&v6, *(unsigned __int16 *)(v8 + 8)) = 0;
  v4 = sub_180026CB4((__int64)&v6);
  sub_1800032E0(v7, v4);
  sub_180026C64(&v6);
  return v7;
}
```

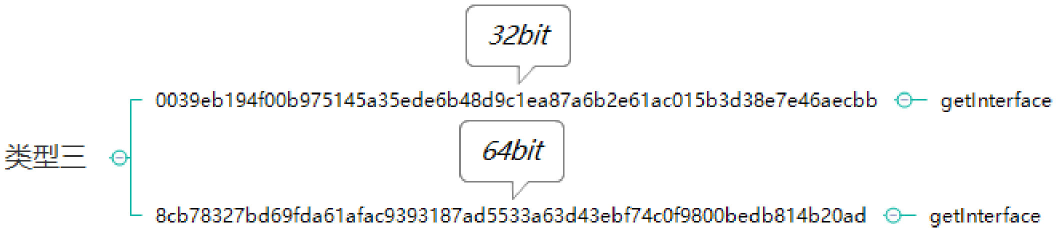
样本 6 解密代码



```
__int64 __fastcall sub_18001B3B0(__int64 a1, __int64 a2)
{
  __int64 v2; // rbx
  __int64 v3; // rsi
  unsigned int v4; // edi
  _BYTE *v5; // rax
  const char *v6; // rdx
  char v8; // [rsp+30h] [rbp-28h]
  _QWORD *v9; // [rsp+38h] [rbp-20h]

  v2 = a2;
  v3 = a1;
  sub_18001B8F8(&v8, *(unsigned __int16 *)(a2 + 8) + 1i64, a2 + 10);
  v4 = *(unsigned __int16 *)(v2 + 8);
  v5 = (_BYTE *)sub_18001BA80(&v8);
  sub_18001B360(v5, v4);
  *(_BYTE *)(*(unsigned __int16 *)(v2 + 8) + *v9) = 0;
  v6 = (const char *)sub_18001BA80(&v8);
  *(_QWORD *)(v3 + 24) = 15i64;
  *(_QWORD *)(v3 + 16) = 0i64;
  *(_BYTE *)v3 = 0;
  sub_1800022D0(v3, v6, strlen(v6));
  sub_18001BA34(&v8);
  return v3;
}
```

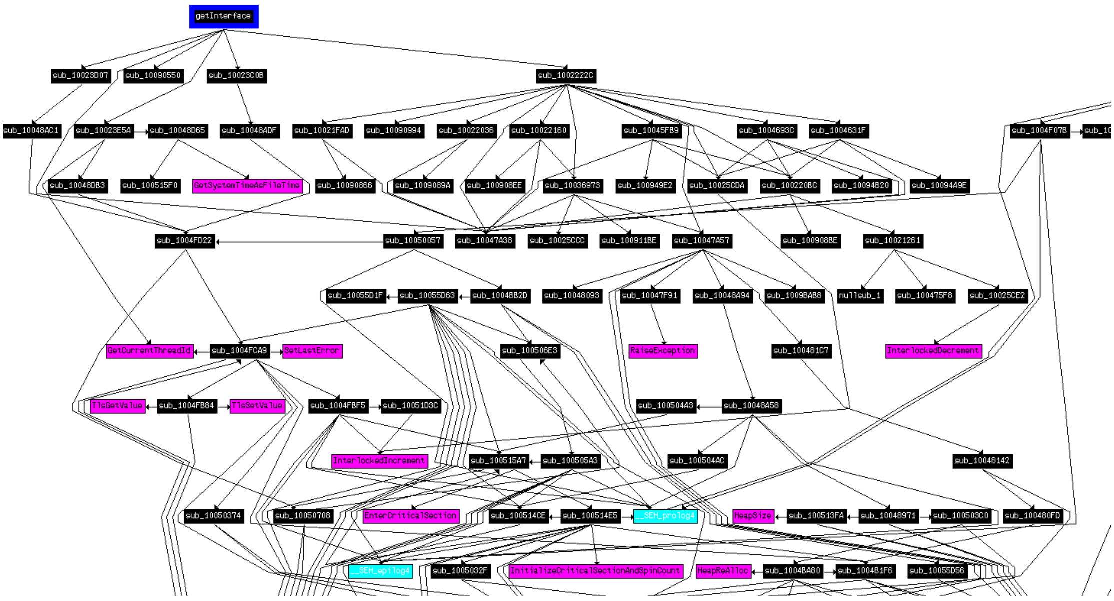
样本 8 解密代码

类型三

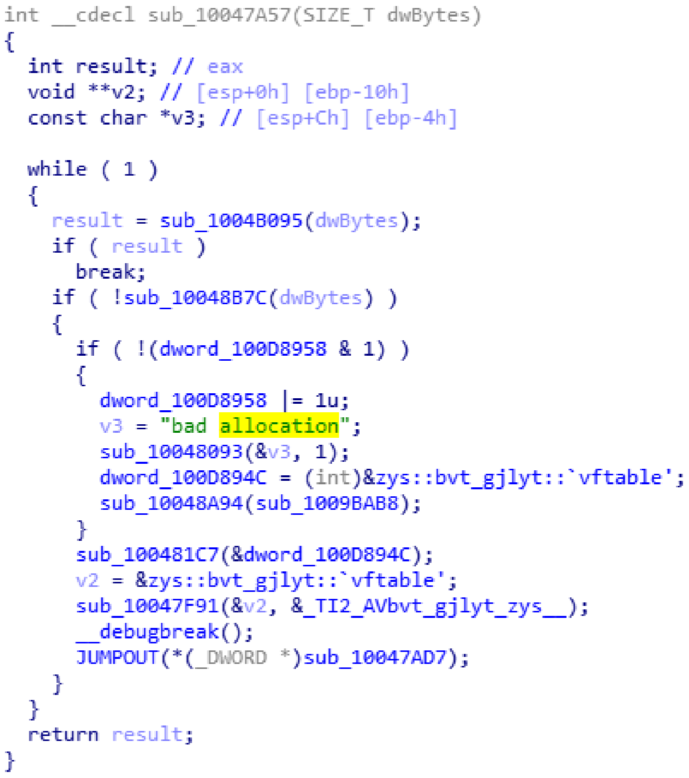

32bit
类型三
0039eb194f00b975145a35ede6b48d9c1ea87a6b2e61ac015b3d38e7e46aecbb
getInterface
64bit
8cb78327bd69fda61afac9393187ad5533a63d43ebf74c0f9800bedb814b20ad
getInterface

样本 7 为 32 位，样本 8 为 64 位，经对比函数流程与 API 调用，功能一致

样本 7 VS 样本 8

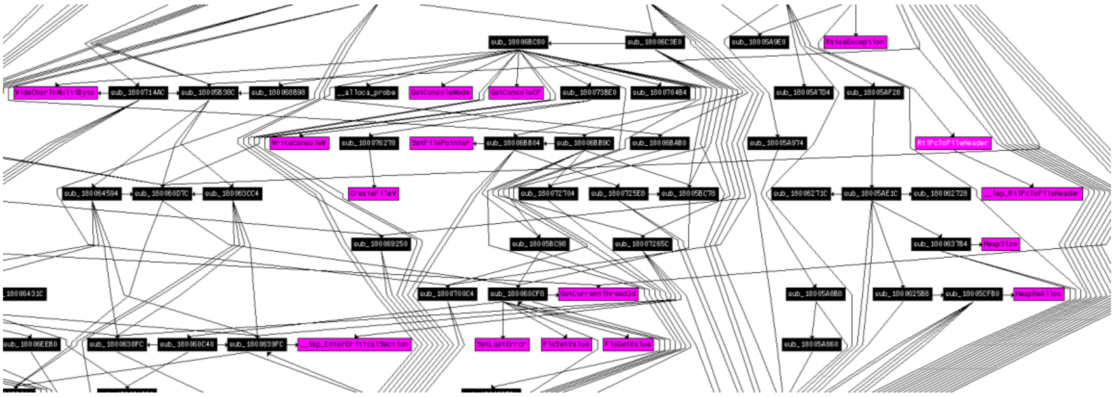


样本 7 部分 API 调用

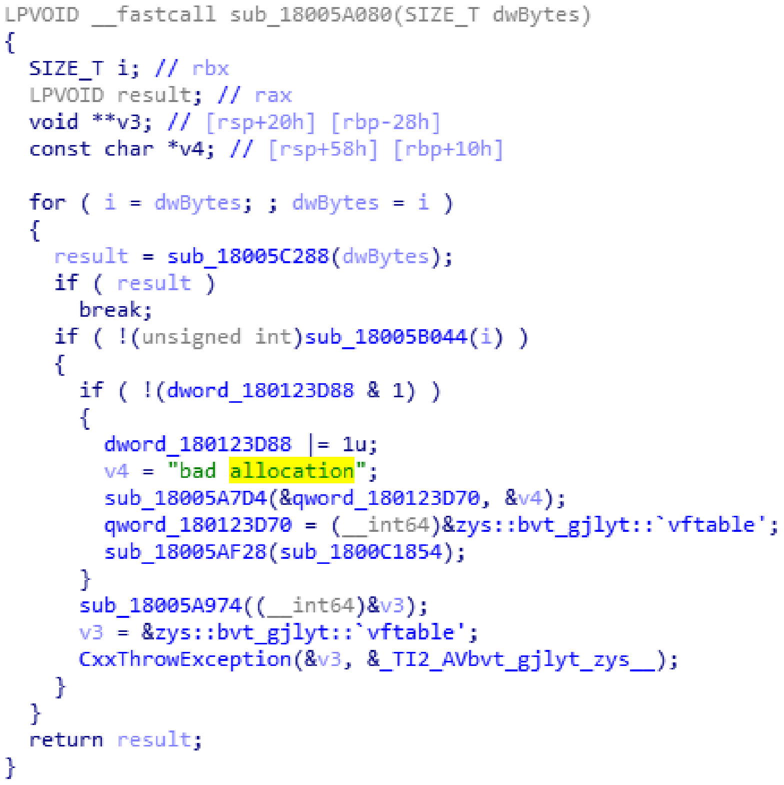


```
int __cdecl sub_10047A57(SIZE_T dwBytes)
{
  int result; // eax
  void **v2; // [esp+0h] [ebp-10h]
  const char *v3; // [esp+Ch] [ebp-4h]

  while ( 1 )
  {
    result = sub_1004B095(dwBytes);
    if ( result )
      break;
    if ( !sub_10048B7C(dwBytes) )
    {
      if ( !(dword_100D8958 & 1) )
      {
        dword_100D8958 |= 1u;
        v3 = "bad allocation";
        sub_10048093(&v3, 1);
        dword_100D894C = (int)&zys::bvt_gjlyt::`vftable';
        sub_10048A94(sub_1009BAB8);
      }
      sub_100481C7(&dword_100D894C);
      v2 = &zys::bvt_gjlyt::`vftable';
      sub_10047F91(&v2, &_TI2_AVbvt_gjlyt_zys__);
      __debugbreak();
      JUMPOUT(*(_DWORD *)sub_10047AD7);
    }
  }
  return result;
}
```

样本 7 部分代码



样本 8 部分 API 调用



```
LPVOID __fastcall sub_18005A080(SIZE_T dwBytes)
{
  SIZE_T i; // rbx
  LPVOID result; // rax
  void **v3; // [rsp+20h] [rbp-28h]
  const char *v4; // [rsp+58h] [rbp+10h]

  for ( i = dwBytes; ; dwBytes = i )
  {
    result = sub_18005C288(dwBytes);
    if ( result )
      break;
    if ( !(unsigned int)sub_18005B044(i) )
    {
      if ( !(dword_180123D88 & 1) )
      {
        dword_180123D88 |= 1u;
        v4 = "bad allocation";
        sub_18005A7D4(&qword_180123D70, &v4);
        qword_180123D70 = (__int64)&zys::bvt_gjlyt::`vftable';
        sub_18005AF28(sub_1800C1854);
      }
      sub_18005A974((__int64)&v3);
      v3 = &zys::bvt_gjlyt::`vftable';
      CxxThrowException(&v3, &_TI2_AVbvt_gjlyt_zys__);
    }
  }
  return result;
}
```

样本 8 部分代码

五、总结

通过对已有样本的分析，样本的编译时间为 2006-2011 年，样本中泄露的静态链接库编译时间为 2014 年，猜测编译时间已被修改。根据样本中包含的 Lua 相关内容，从资源中解密数据等功能，以及开源情报，确认该批样本为 Flame2.0。Flame1.0 模块通过基于 XOR 的密钥解密嵌入的资源，使用 ZLIB 解压缩。在此新

版本中，通过 AES256 加密嵌入式资源。部分模块包含 Plink 相关的字符串，猜测可能用于横向移动。由于样本导出函数需传参解密资源文件，无法正常执行调试，暂时只能根据反汇编代码分析部分片段，无法准确判断具体功能。