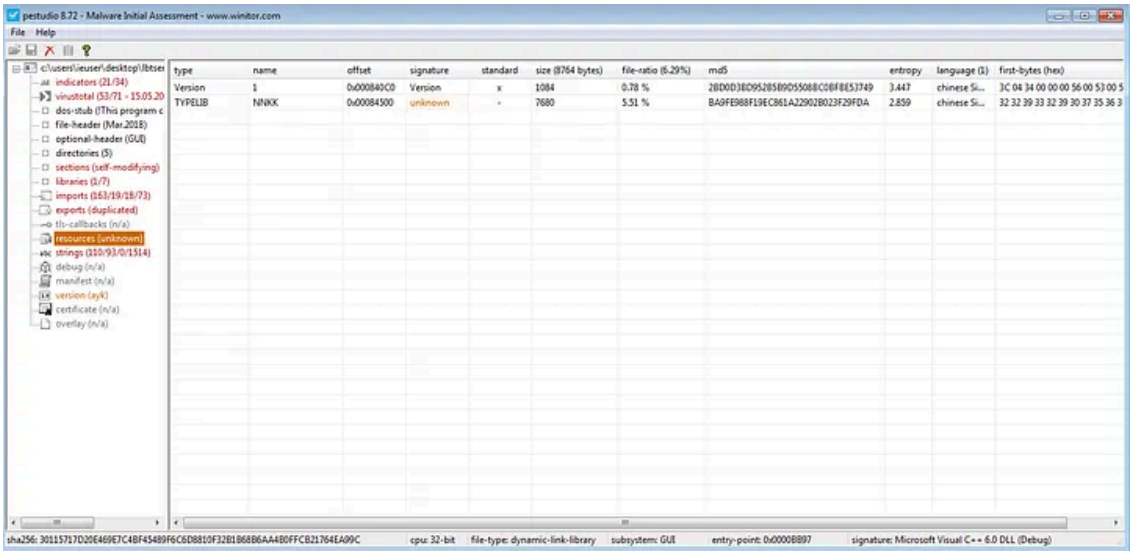
How to unpack Chinoxy backdoor and decipher the configuration of the backdoor

By Sebdraven

Published: 2020-07-08 · Archived: 2026-04-05 19:34:58 UTC

In my last article on Chinoxy backdoor, this version has its configuration in a resource called NNKK and it is deciphered. The purpose of this article is to explain the unpacking and deciphering of the configuration of this backdoor.

Press enter or click to view image in full size



The backdoor is loading with the program confax.exe, a utility of Logitech for the Bluetooth.

The function called by confax.exe is LGBT_Launch.

In checking this function,

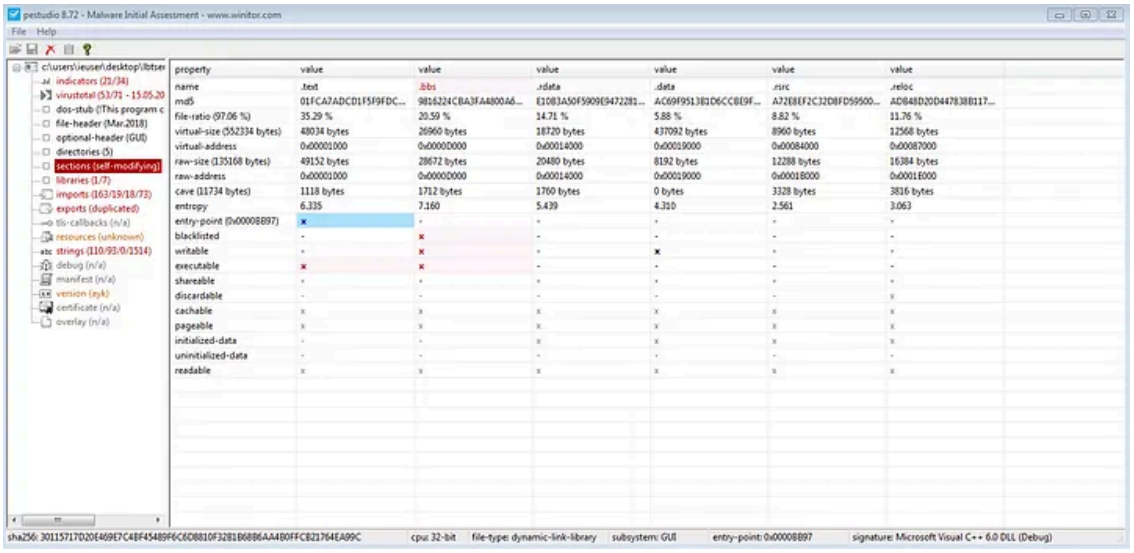
```
void LGBT_Launch(void)

{
  HANDLE hHandle;
  DWORD local_4;

  do {
    hHandle = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,(LPTHREAD_START_ROUTINE)&DAT_10011fe0,
                           (LPVOID)0x0,0,&local_4);
    WaitForSingleObject(hHandle,0xffffffff);
    CloseHandle(hHandle);
    Sleep(10000);
  } while( true );
}
```

The entry of the thread is pointed by the address DAT_10011fe0. this address is in the section .bss. This section has rwx and the entropy is very high.

Press enter or click to view image in full size



Before the unpack, there are not a call with the function using this resource.

And at the address DAT_10011fe0, there is just data without code.

So the unpack procedure is using the entrypoint of the backdoor, and the code is executing when confax.exe load LBTServ.dll.

The entrypoint of the dll, the function interesting is FUN_10007800.

Get Sebdraven's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

the code call the function 10007770 with two parameters: an handle on the dll and the key hceqhqn of the xor.

Press enter or click to view image in full size

```
_DAT_1004a460 = param_1;
if (param_2 == 1) {
  hModule = GetModuleHandleW(u_LBTServ.dll_1001a87c);
  xor_fun_10000001((int)hModule,key_xor_hceqhqn);
  SetErrorMode(1);
```

In this function, the xor is at the end of the function after manipulating the the &DAT_1001a7dc for a copy.

Press enter or click to view image in full size

```
iVar3 = hModule + *(int *)(hModule + 0x3c);
iVar9 = 0;
pbVar7 = (byte *)(iVar3 + 0xf8);
iVar3 = (int)*(short *)(iVar3 + 6);
if (iVar3 < 1) {
  return;
}
do {
  pbVar8 = &DAT_1001a7dc;
  pbVar4 = pbVar7;
  do {
    bVar2 = *pbVar4;
    bVar11 = bVar2 < *pbVar8;
    if (bVar2 != *pbVar8) {
LAB_100077bb:
      iVar5 = (1 - (uint)bVar11) - (uint)(bVar11 != false);
      goto LAB_100077c0;
    }
    if (bVar2 == 0) break;
    bVar2 = pbVar4[1];
    bVar11 = bVar2 < pbVar8[1];
    if (bVar2 != pbVar8[1]) goto LAB_100077bb;
    pbVar4 = pbVar4 + 2;
    pbVar8 = pbVar8 + 2;
  } while (bVar2 != 0);
  iVar5 = 0;
LAB_100077c0:
  if (iVar5 == 0) {
    uVar6 = 0xffffffff;
    pcVar10 = key;
    break;
  }
  pbVar7 = pbVar7 + 0x28;
  iVar9 = iVar9 + 1;
  if (iVar3 <= iVar9) {
    return;
  }
} while( true );
while( true ) {
  uVar6 = uVar6 - 1;
  cVar1 = *pcVar10;
  pcVar10 = pcVar10 + 1;
  if (cVar1 == '\0') break;
  if (uVar6 == 0) break;
}
xor(*(int *)(pbVar7 + 0xc) + hModule,*(int *)(pbVar7 + 0x10),(int)key,~uVar6 - 1);
return;
}
```

the xor function is located at 10007730.

Press enter or click to view image in full size

```
void __cdecl xor(int offset_hmodule,int param_2,int key,int param_4)

{
  byte *pbVar1;
  int iVar2;
  int iVar3;
  int iVar4;

  iVar2 = 0;
  if (0 < param_2) {
    do {
      iVar4 = 0;
      iVar3 = iVar2;
      if (0 < param_4) {
        do {
          if (param_2 <= iVar3) {
            return;
          }
          pbVar1 = (byte *)(iVar4 + key);
          iVar4 = iVar4 + 1;
          iVar2 = iVar3 + 1;
          *(byte *)(iVar3 + offset_hmodule) = ~(*pbVar1 ^ *(byte *)(iVar3 + offset_hmodule));
          iVar3 = iVar2;
        } while (iVar4 < param_4);
      }
    } while (iVar2 < param_2);
  }
  return;
}
```

And after the function, if the dll is dumped. We found the good function of the thread and the function manipulating the resource.

Press enter or click to view image in full size

```
void LGBT_Launch(void)

{
  HANDLE hHandle;
  DWORD local_4;

  do {
    hHandle = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,(LPTHREAD_START_ROUTINE)&LAB_10011fe0,
                           (LPVOID)0x0,0,&local_4);
    WaitForSingleObject(hHandle,0xffffffff);
    CloseHandle(hHandle);
    Sleep(10000);
  } while( true );
}
```

Press enter or click to view image in full size

```
                    u_NNKK_100159fc                                 XREF[1]:     FUN_10005c50:10005c60(*)
100159fc 4e 00 4e       unicode     u"NNKK"
         00 4b 00
         4b 00 00 00
```

So we check the function 10005c50 using this resource called by the thread.

```
         04 10
100120b4 89 5c 24 20     MOV         dword ptr [ESP + 0x20],EBX
100120b8 e8 93 3b        CALL        FUN_10005c50
         ff ff
```

In this function, the ressource is locked and two keys are catching:

Press enter or click to view image in full size

```
pHVar1 = FindResourceW(param_2,u_NNKK_100159fc,u_TYPELIB_1001a290);
if (pHVar1 != (HRSRC)0x0) {
  hModule = LoadLibraryW(u_kernel32.dll_10019720);
  local_b = 0x65;
  local_5 = 0x65;
  local_10 = 'L';
  local_f = 0x6f;
  local_e = 0x61;
  local_d = 100;
  local_c = 0x52;
  local_a = 0x73;
  local_9 = 0x6f;
  local_8 = 0x75;
  local_7 = 0x72;
  local_6 = 99;
  local_4 = (undefined4 *)((uint)local_4 & 0xffffff00);
  pFVar2 = GetProcAddress(hModule,&local_10);
  hResData = (HGLOBAL)(*pFVar2)(param_2,pHVar1);
  FreeLibrary(hModule);
  if (hResData != (HGLOBAL)0x0) {
    puVar4 = (undefined4 *)LockResource(hResData);
    iVar3 = 0x4c0;
    puVar5 = local_4;
    while (iVar3 != 0) {
      iVar3 = iVar3 + -1;
      *puVar5 = *puVar4;
      puVar4 = puVar4 + 1;
      puVar5 = puVar5 + 1;
    }
    FUN_10006bf0((int)local_4,0x1300);
    return 1;
```

The keys are just top of the resource of TYPELIB.

```
                    DAT_1001a278                                    XREF[1]:
1001a278 32             undefined1  32h

                    s_2135987565_1001a279                           XREF[1]:
1001a279 32 31 33       ds          "2135987565"
         35 39 38
         37 35 36 ...

                    DAT_1001a284                                    XREF[1]:
1001a284 33             undefined1  33h

                    s_6969856569_1001a285                           XREF[1]:
1001a285 36 39 36       ds          "6969856569"
         39 38 35
         36 35 36 ...

                    u_TYPELIB_1001a290                              XREF[1]:
1001a290 54 00 59       unicode     u"TYPELIB"
         00 50 00
         45 00 4c ...

                    ****************************************************************
                    * class CCursorInfo RTTI Type Descriptor                       *
                    ****************************************************************
```

And the deciphered function is the function 10005bf0.

Press enter or click to view image in full size

```
1
2  void __cdecl FUN_10005bf0(int param_1,uint param_2)
3
4  {
5    uint uVar1;
6    uint uVar2;
7    uint uVar3;
8
9    if ((param_1 != 0) && (param_2 != 0)) {
10     uVar1 = 0;
11     uVar2 = 0;
12     uVar3 = 0;
13     if (param_2 != 0) {
14       do {
15         *(byte *)(uVar1 + param_1) =
16              *(byte *)(uVar1 + param_1) ^
17              ((&DAT_1001a278)[uVar3] ^ (&DAT_1001a284)[uVar2]) & 0x27 ^ (&DAT_1001a284)[uVar2];
18         uVar2 = (uVar2 + 1) % 0xc;
19         uVar3 = (uVar3 + 1) % 0xc;
20         uVar1 = uVar1 + 1;
21       } while (uVar1 < param_2);
22     }
23   }
24   return;
25 }
```

the param1 is the pointer on the resource and the param 2 the number of the step for the loop deciphering.

In python, the algorithm is the following. with the RatDecoders, we found the resource

from malwareconfig import fileparser
import binascii

rsc = file_info.pe_resource_by_name('NNKK')

key_1= b’369698565690'
key_2= b’221359875650'

res=b’’.join([chr((key_1[i%12] ^ key_2[i%12]) & 0x27 ^ key_1[i%12] ^ rsc[i]).encode() for i in range(0,0x1300)])

res.replace(b’\x00',b’’).replace(b’0',b’’)

we have like result:

```
b'\x0f0\xc2\xb7\xc2\x99\x03YnJh0bmRzLm05ld3N00LmRuc20Fici5j0b206Mz0AxMHxi0cmFuZH0MubmV30c3QuZG05zYWJ
```

with a little cleaning, and base64 ninja, we have the result.

> brands.newst.dnsabr.com:3010|brands.newst.dnsabr.com:3010|ru.mst.dns-cloud.net:3010|

This IOCs has been already done.

Conclusion

the purpose of this article is to explain to unpack quickly the Chinoxy backdoor and retrieve the configuration without reverse the backdoor.
