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

medium.com/@Sebdraven/how-to-unpack-chinoxy-backdoor-and-decipher-the-configuration-of-the-backdoor-4ffd98ca2a02

Sebdraven July 8, 2020

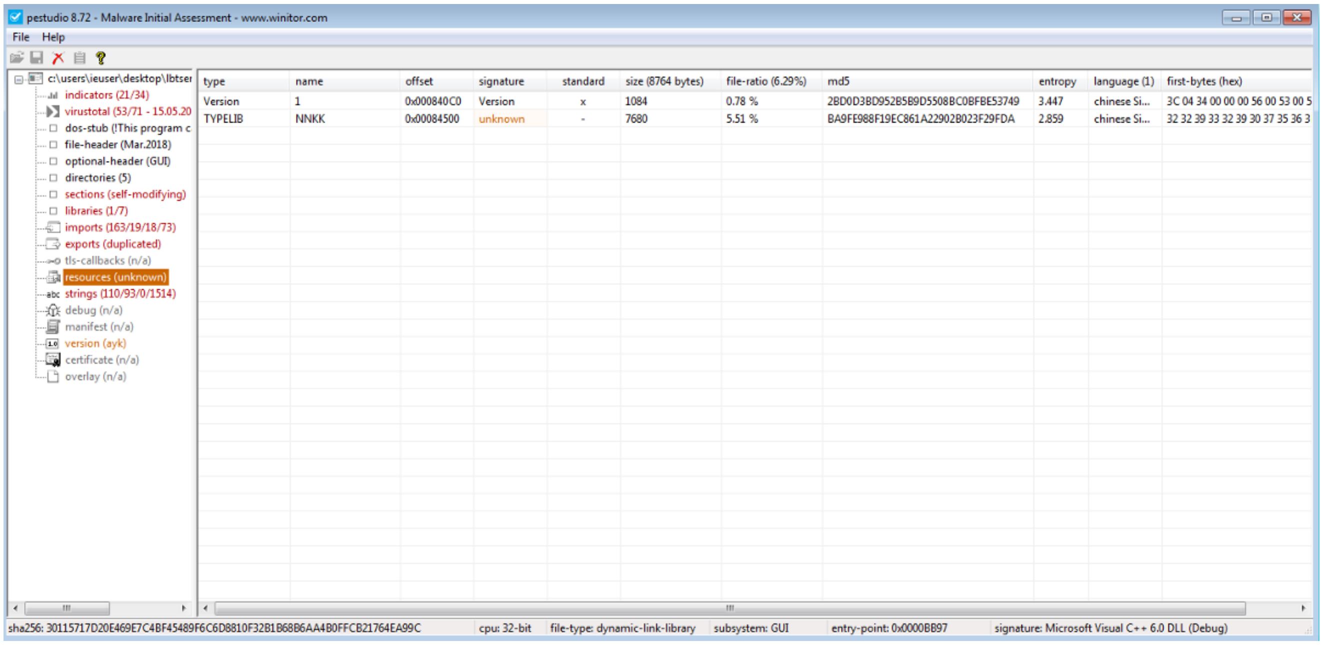
Sebdraven

Jul 8, 2020

·

3 min read

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.



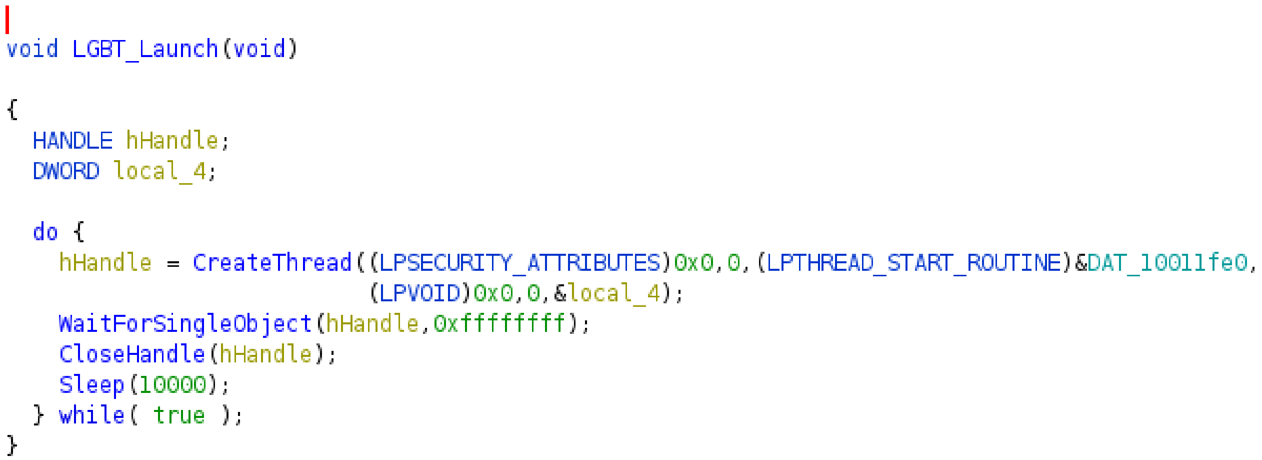
type	name	offset	signature	standard	size (8764 bytes)	file-ratio (6.29%)	md5	entropy	language (1)	first-bytes (hex)
Version	1	0x000840C0	Version	x	1084	0.78 %	2BD0D3BD952B5B9D5508BC0BFBE53749	3.447	chinese Si...	3C 04 34 00 00 00 56 00 53 00 5
TYPELIB	NNKK	0x00084500	unknown	-	7680	5.51 %	BA9FE988F19EC861A22902B023F29FDA	2.859	chinese Si...	32 32 39 33 32 39 30 37 35 36 3

sha256: 30115717D20E469E7C4BF45489F6C6D8810F32B1B68B6AA4B0FFCB21764EA99C cpu: 32-bit file-type: dynamic-link-library subsystem: GUI entry-point: 0x0000BB97 signature: Microsoft Visual C++ 6.0 DLL (Debug)

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

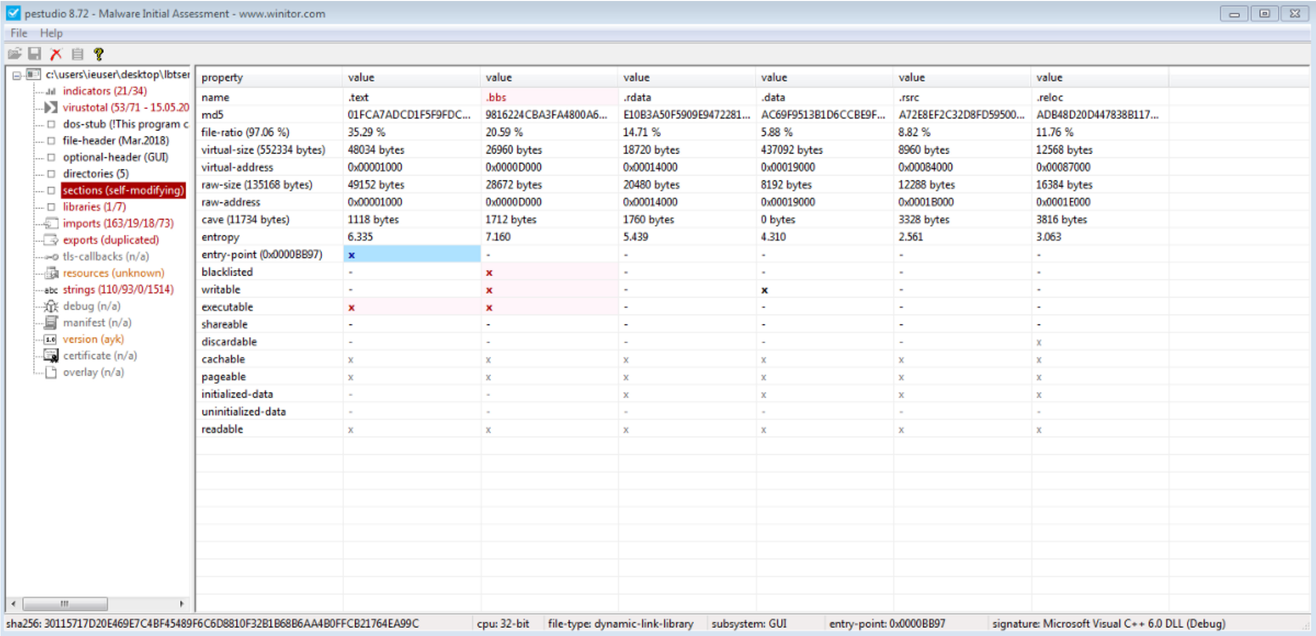
The function called by confax.exe is LGBT_Launch.

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.



property	value	value	value	value	value	value
name	.text	.bbs	.rdata	.data	.rsrc	.reloc
md5	01FCA7ADCD1F5F9FDC...	9816224CBA3FA4800A6...	E10B3A50F5909E9472281...	AC69F9513B1D6CCBE9F...	A72E8EF2C32D8FD59500...	ADB48D20D447838B117...
file-ratio (97.06 %)	35.29 %	20.59 %	14.71 %	5.88 %	8.82 %	11.76 %
virtual-size (552334 bytes)	48034 bytes	26960 bytes	18720 bytes	437092 bytes	8960 bytes	12568 bytes
virtual-address	0x00001000	0x0000D000	0x00014000	0x00019000	0x00084000	0x00087000
raw-size (135168 bytes)	49152 bytes	28672 bytes	20480 bytes	8192 bytes	12288 bytes	16384 bytes
raw-address	0x00001000	0x0000D000	0x00014000	0x00019000	0x0001B000	0x0001E000
cave (11734 bytes)	1118 bytes	1712 bytes	1760 bytes	0 bytes	3328 bytes	3816 bytes
entropy	6.335	7.160	5.439	4.310	2.561	3.063
entry-point (0x0000BB97)	x	-	-	-	-	-
blacklisted	-	x	-	-	-	-
writable	-	x	-	x	-	-
executable	x	x	-	-	-	-
shareable	-	-	-	-	-	-
discardable	-	-	-	-	-	x
cachable	x	x	x	x	x	x
pageable	x	x	x	x	x	x
initialized-data	-	-	x	x	x	x
uninitialized-data	-	-	-	-	-	-
readable	x	x	x	x	x	x

sha256: 30115717D20E469E7C4BF45489F6C6D8810F32B1B68B6AA4B0FFCB21764EA99C   cpu: 32-bit   file-type: dynamic-link-library   subsystem: GUI   entry-point: 0x0000BB97   signature: Microsoft Visual C++ 6.0 DLL (Debug)

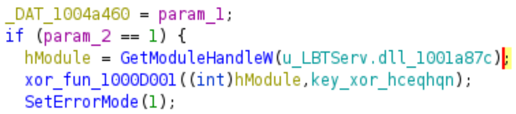
Before the unpack, there are not a call with the function using this resource.

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.

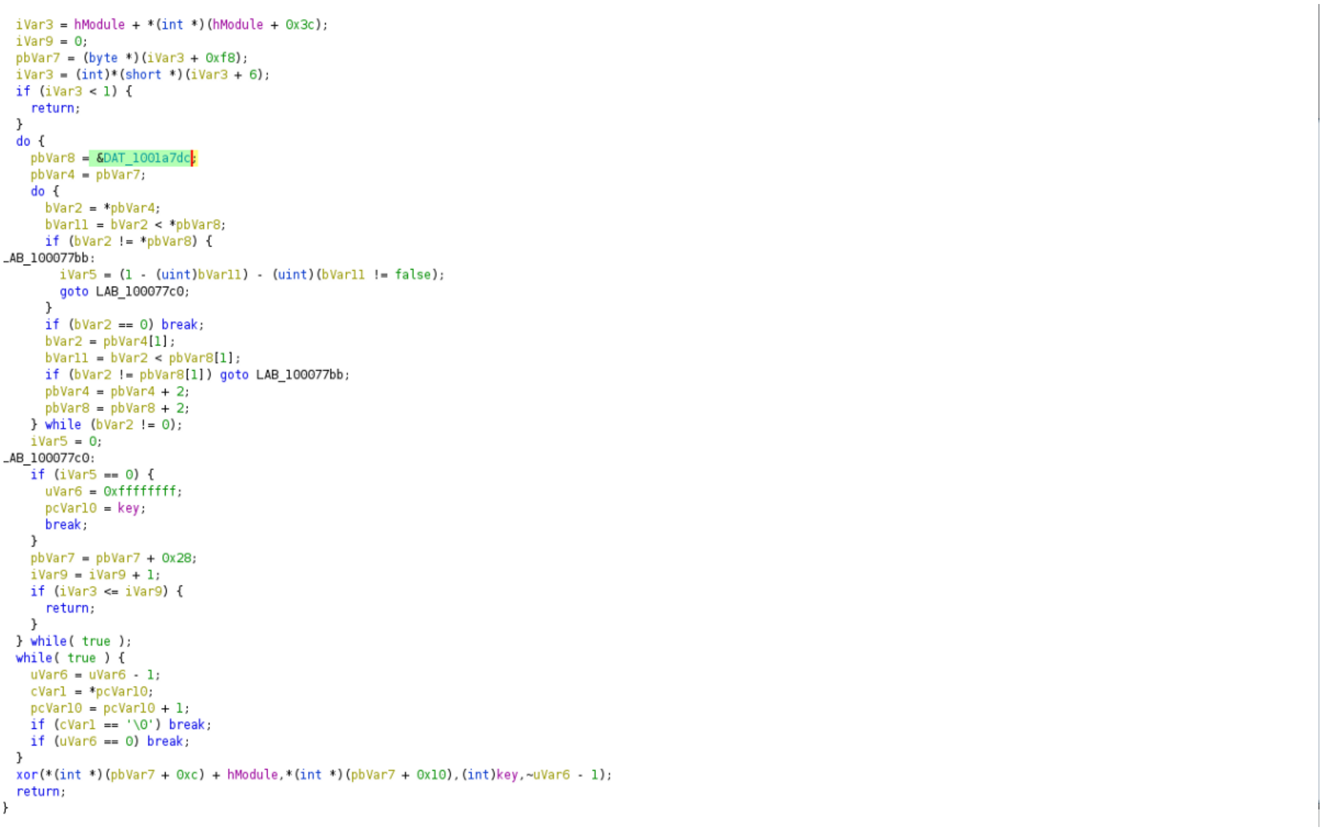
The entrypoint of the dll, the function interesting is FUN_10007800.

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



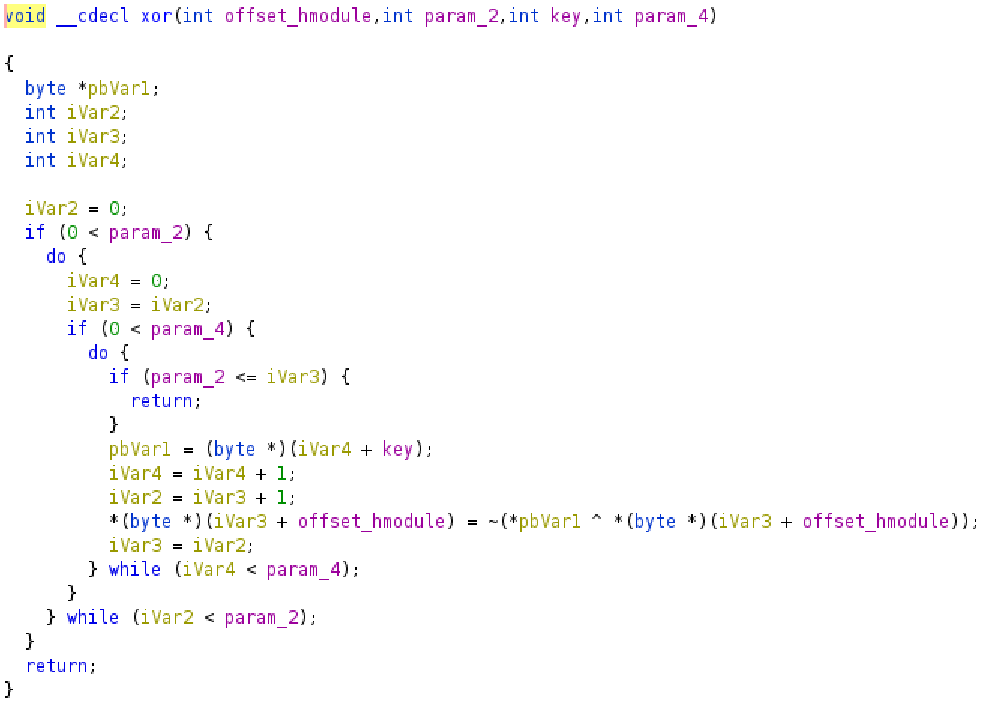
```
_DAT_1004a460 = param_1;
if (param_2 == 1) {
  hModule = GetModuleHandleW(u_LBTServ.dll_1001a87c);
  xor_fun_1000D001((int)hModule,key_xor_hceqhqn);
  SetErrorMode(1);
```

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



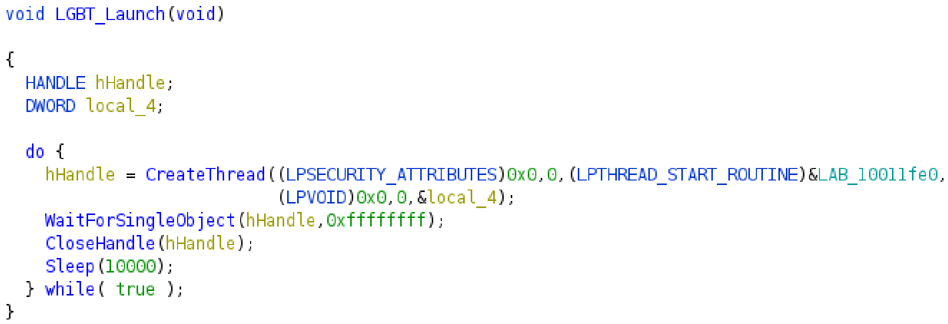
```
  iVar3 = hModule + *(int *)(hModule + 0x3c);
  iVar9 = 0;
  pbVar7 = (byte *)(iVar3 + 0xf8);
  iVar3 = (int)*(short *)(iVar3 + 6);
  if (iVar3 < 1) {
    return;
  }
  do {
    pbVar8 = &DAT_1001a7dc;
    pbVar4 = pbVar7;
    do {
      bVar2 = *pbVar4;
      bVar11 = bVar2 < *pbVar8;
      if (bVar2 != *pbVar8) {
_AB_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;
_AB_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.

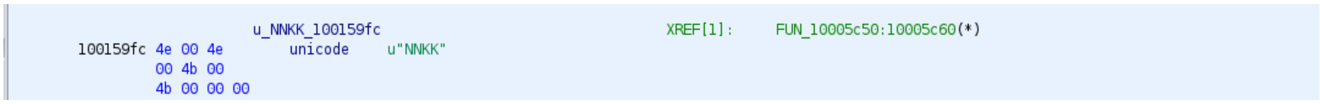


```
void __cdecl xor(int offset_hmodule,int param_2,int key,int param_4)

{
  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.

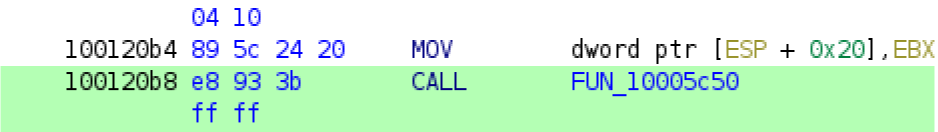


```
void LGBT_Launch(void)

{
  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 );
}
```



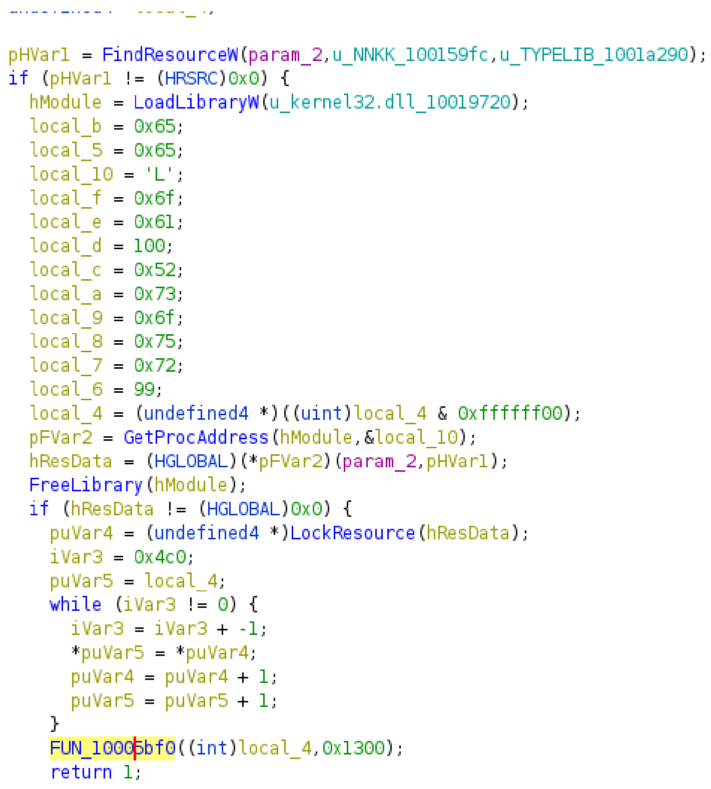
```
                         u_NNKK_100159fc                              XREF[1]:     FUN_10005c50:10005c60(*)
        100159fc 4e 00 4e         unicode     u"NNKK"
                 00 4b 00
                 4b 00 00 00
```

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



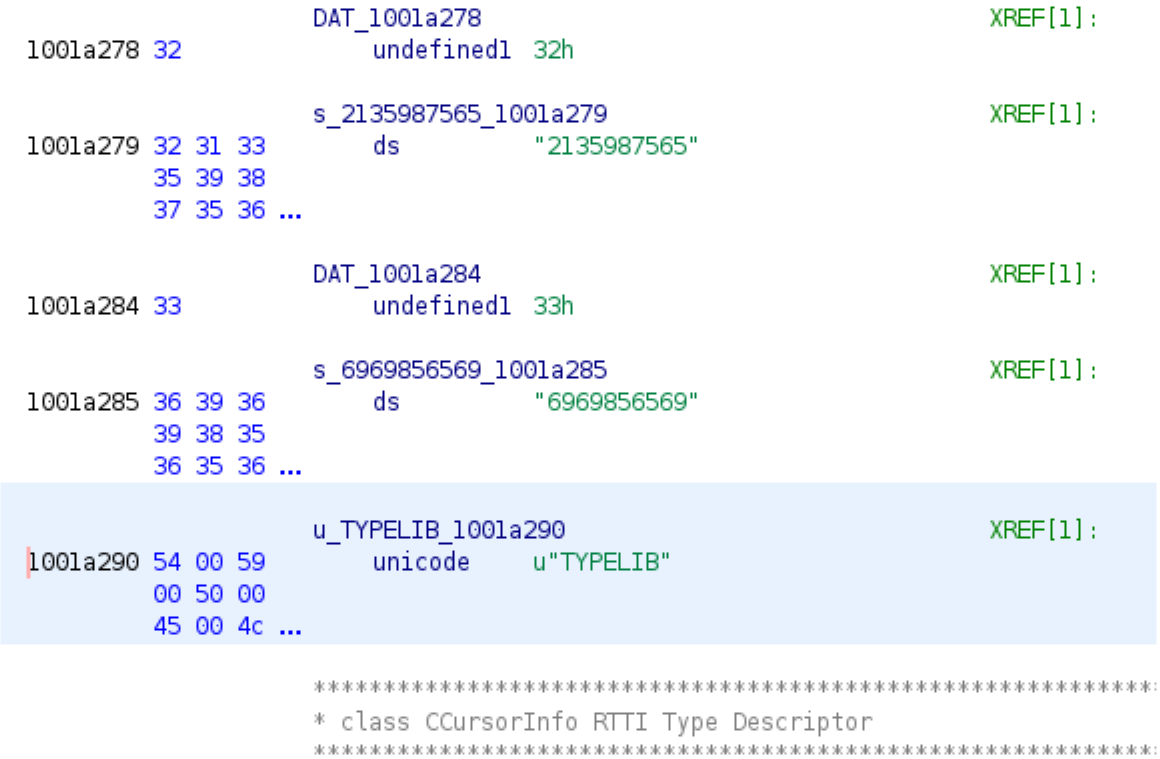
```
                 04 10
        100120b4 89 5c 24 20      MOV          dword ptr [ESP + 0x20],EBX
        100120b8 e8 93 3b         CALL         FUN_10005c50
                 ff ff
```

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



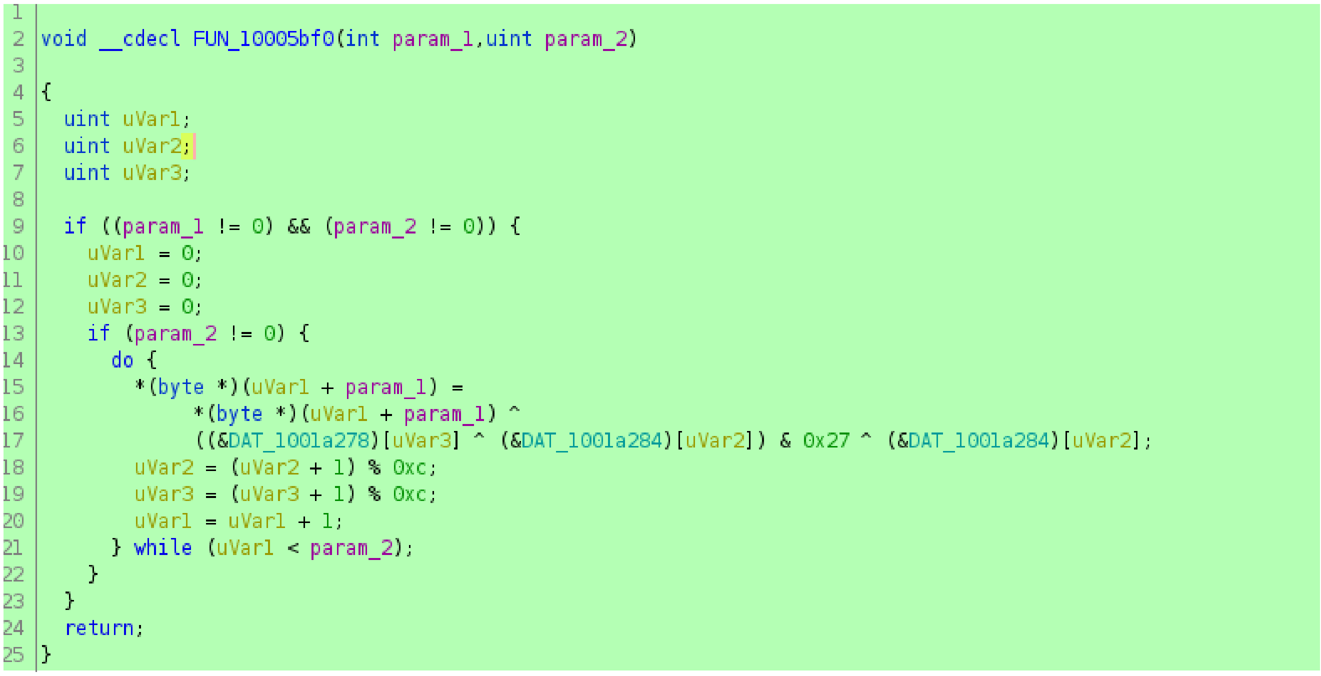
```
  pHVar1 = FindResourceW(param_2,u_NNKK_100159fc,u_TYPELIB_1001a290);
  if (pHVar1 != (HRSRC)0x0) {
    hModule = LoadLibraryW(u_kernel32.dll_10019720);
    local_b = 0x65;
    local_5 = 0x65;
    local_10 = 'L';
    local_f = 0x6f;
    local_e = 0x61;
    local_d = 100;
    local_c = 0x52;
    local_a = 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_10005bf0((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.



```

void __cdecl FUN_10005bf0(int param_1,uint param_2)

{
  uint uVar1;
  uint uVar2;
  uint uVar3;

  if ((param_1 != 0) && (param_2 != 0)) {
    uVar1 = 0;
    uVar2 = 0;
    uVar3 = 0;
    if (param_2 != 0) {
      do {
        *(byte *)(uVar1 + param_1) =
             *(byte *)(uVar1 + param_1) ^
             ((&DAT_1001a278)[uVar3] ^ (&DAT_1001a284)[uVar2]) & 0x27 ^ (&DAT_1001a284)[uVar2];
        uVar2 = (uVar2 + 1) % 0xc;
        uVar3 = (uVar3 + 1) % 0xc;
        uVar1 = uVar1 + 1;
      } while (uVar1 < param_2);
    }
  }
  return;
}
```

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\x03YnJh0bmRzLm05ld3N00LmRuc20Fici5j0b206Mz0AxMHxi0cmFuZH0MubmV
 0Group000005300000001.00000000000660489388000000000000'
```

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.