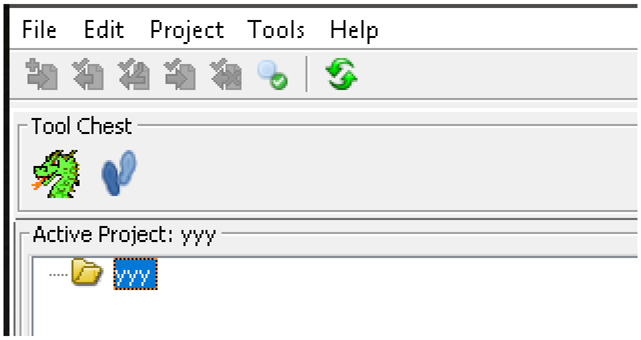
Quick Analysis of a Trickbot Sample with NSA's Ghidra SRE Framework

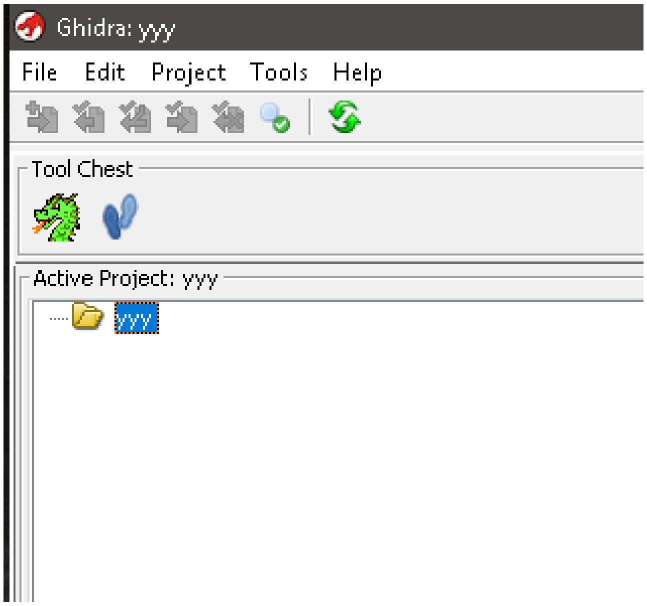
peppermalware.com/2019/03/quick-analysis-of-trickbot-sample-with.html



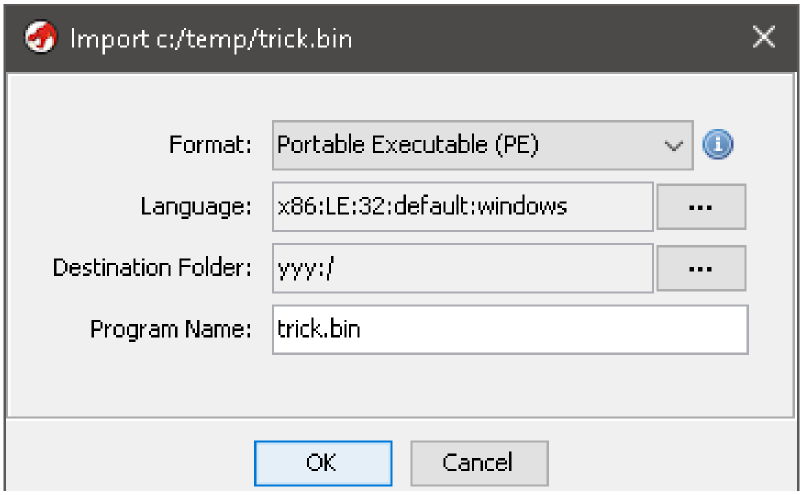
This post is not a deep analysis of TrickBot. Here, I did a quick analysis of a TrickBot sample from early 2019 by using the Ghidra Software Reverse Engineering (SRE) Framework, developed by the NSA, that was released some hours ago. This is not a deep analysis of TrickBot, I only wanted to learn a bit about Ghidra and I used this framework to find some interesting parts of the code of TrickBot that were introduced in the newer versions of the malware. Hope you enjoy it!

Starting with Ghidra Framework

About Ghidra, when you start the framework, you should create a project and a workspace:



Then, we can import files, for example PE files:



Import c:/temp/trick.bin

Format: Portable Executable (PE)

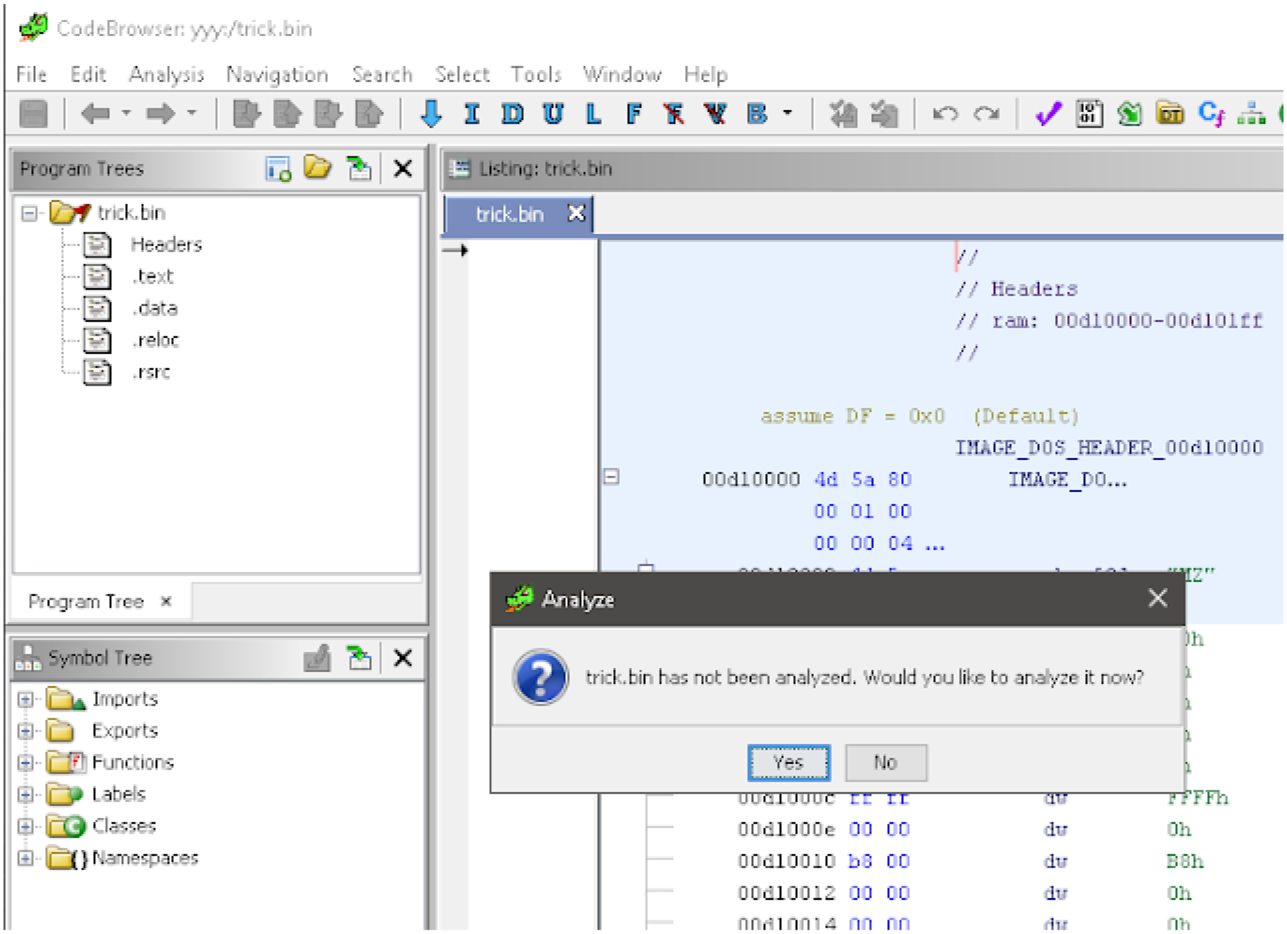
Language: x86:LE:32:default:windows

Destination Folder: yyy:/

Program Name: trick.bin

Ghidra CodeBrowser

Once PE file is imported, CodeBrowser can be launched:



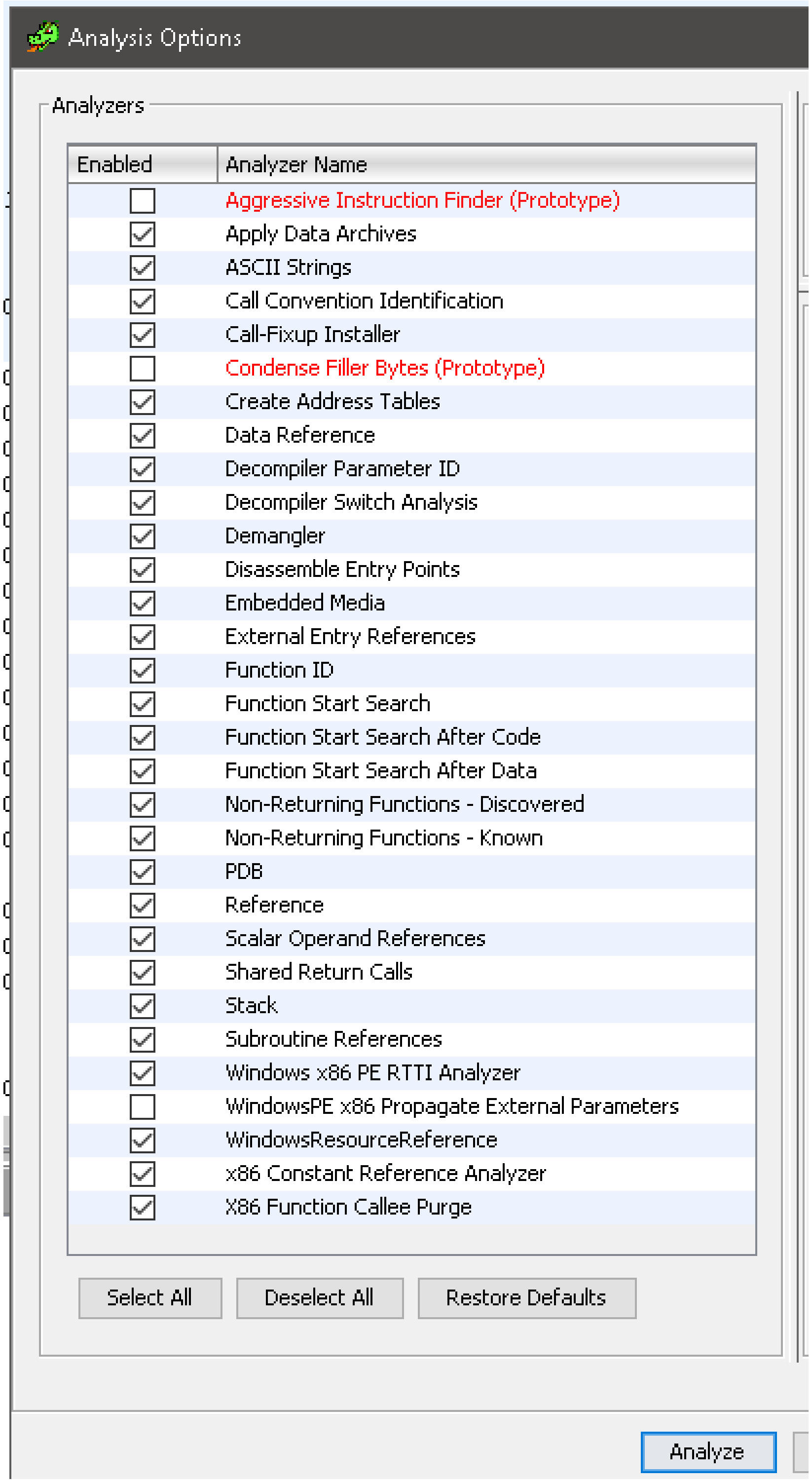
CodeBrowser: yyy:/trick.bin

```
                          //
                          // Headers
                          // ram: 00d10000-00d101ff
                          //

               assume DF = 0x0  (Default)
                          IMAGE_DOS_HEADER_00d10000
    00d10000 4d 5a 80         IMAGE_DO...
             00 01 00
             00 00 04 ...
      00d1000c ff ff      dw      FFFFh
      00d1000e 00 00      dw      0h
      00d10010 b8 00      dw      B8h
      00d10012 00 00      dw      0h
      00d10014 00 00      dw      0h
```

Analyze

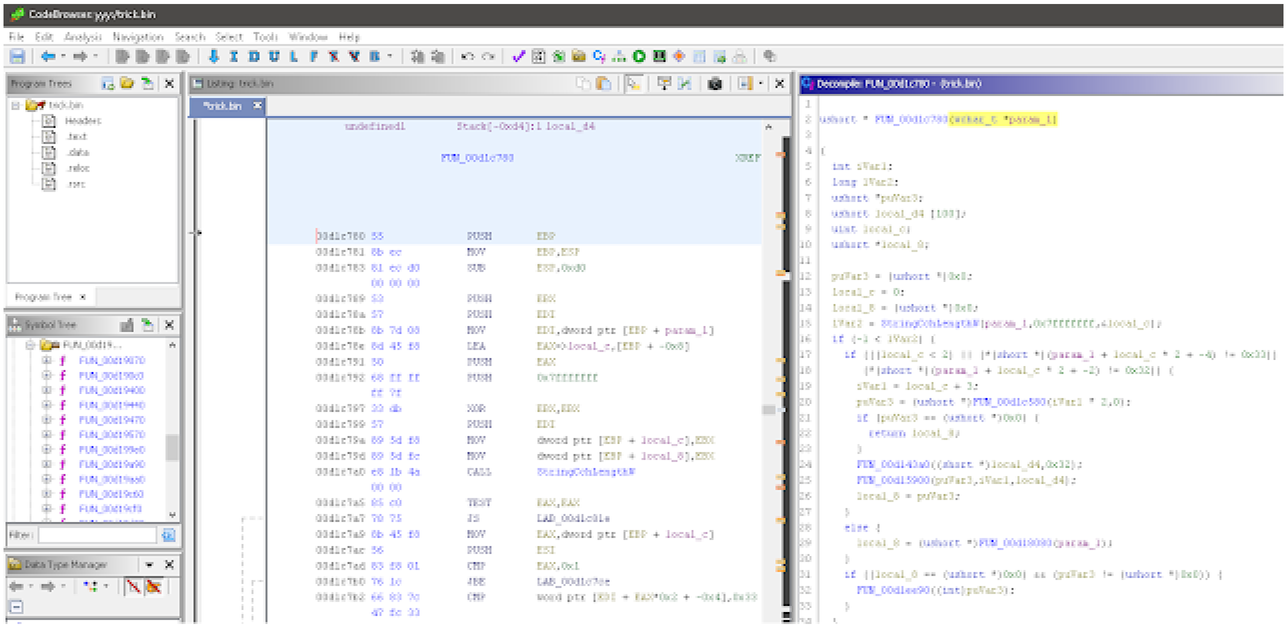
trick.bin has not been analyzed. Would you like to analyze it now?

Initially, PE headers are parsed but code is not analyzed, the framework asks you if analyzers should be launched, and what analyzers should be launched. This is the list of analyzers (they are marked the analyzers that are marked by default):



Enabled	Analyzer Name
☐	Aggressive Instruction Finder (Prototype)
☑	Apply Data Archives
☑	ASCII Strings
☑	Call Convention Identification
☑	Call-Fixup Installer
☐	Condense Filler Bytes (Prototype)
☑	Create Address Tables
☑	Data Reference
☑	Decompiler Parameter ID
☑	Decompiler Switch Analysis
☑	Demangler
☑	Disassemble Entry Points
☑	Embedded Media
☑	External Entry References
☑	Function ID
☑	Function Start Search
☑	Function Start Search After Code
☑	Function Start Search After Data
☑	Non-Returning Functions - Discovered
☑	Non-Returning Functions - Known
☑	PDB
☑	Reference
☑	Scalar Operand References
☑	Shared Return Calls
☑	Stack
☑	Subroutine References
☑	Windows x86 PE RTTI Analyzer
☐	WindowsPE x86 Propagate External Parameters
☑	WindowsResourceReference
☑	x86 Constant Reference Analyzer
☑	X86 Function Callee Purge

Once analyzers finish, CodeBrowser interface is like this:



```
                undefined1      Stack[-0xd4]:1 local_d4

                        FUN_00d1c780                              XREF

00d1c780 55          PUSH    EBP
00d1c781 8b ec       MOV     EBP,ESP
00d1c783 81 ec d0    SUB     ESP,0xd0
         00 00 00
00d1c789 53          PUSH    EBX
00d1c78a 57          PUSH    EDI
00d1c78b 8b 7d 08    MOV     EDI,dword ptr [EBP + param_1]
00d1c78e 8d 45 f8    LEA     EAX=>local_c,[EBP + -0x8]
00d1c791 50          PUSH    EAX
00d1c792 68 ff ff    PUSH    0x7FFFFFFF
         ff 7f
00d1c797 33 db       XOR     EBX,EBX
00d1c799 57          PUSH    EDI
00d1c79a 89 5d f8    MOV     dword ptr [EBP + local_c],EBX
00d1c79d 89 5d fc    MOV     dword ptr [EBP + local_8],EBX
00d1c7a0 e8 1b 4a    CALL    StringCchLengthW
         00 00
00d1c7a5 85 c0       TEST    EAX,EAX
00d1c7a7 78 75       JS      LAB_00d1c81e
00d1c7a9 8b 45 f8    MOV     EAX,dword ptr [EBP + local_c]
00d1c7ac 56          PUSH    ESI
00d1c7ad 83 f8 01    CMP     EAX,0x1
00d1c7b0 76 1c       JBE     LAB_00d1c7ce
00d1c7b2 66 83 7c    CMP     word ptr [EDI + EAX*0x2 + -0x4],0x33
         47 fc 33
```

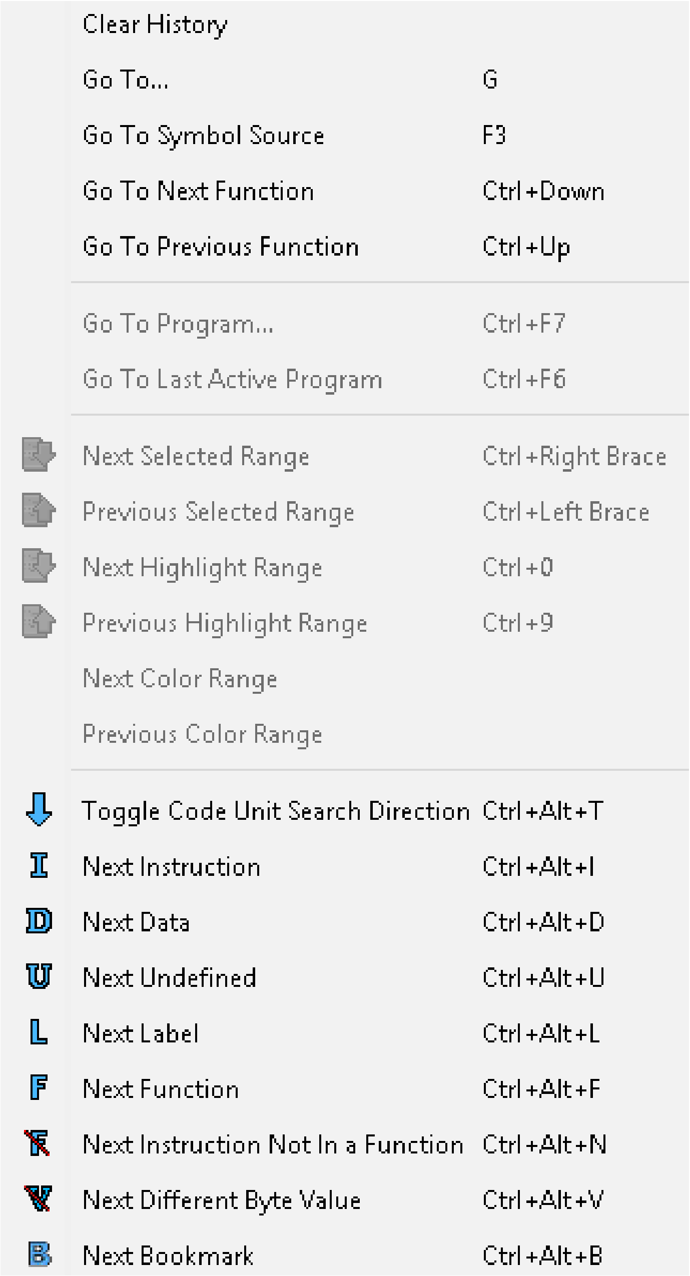
```
ushort * FUN_00d1c780(wchar_t *param_1)

{
  int iVar1;
  long lVar2;
  ushort *puVar3;
  ushort local_d4 [100];
  uint local_c;
  ushort *local_8;

  puVar3 = (ushort *)0x0;
  local_c = 0;
  local_8 = (ushort *)0x0;
  lVar2 = StringCchLengthW(param_1,0x7fffffff,&local_c);
  if (-1 < lVar2) {
    if (((local_c < 2) || (*(short *)(param_1 + local_c * 2 + -4) != 0x33)) ||
       (*(short *)(param_1 + local_c * 2 + -2) != 0x32)) {
      iVar1 = local_c + 3;
      puVar3 = (ushort *)FUN_00d1c580(iVar1 * 2,0);
      if (puVar3 == (ushort *)0x0) {
        return local_8;
      }
      FUN_00d142a0((short *)local_d4,0x32);
      FUN_00d15900(puVar3,iVar1,local_d4);
      local_8 = puVar3;
    }
    else {
      local_8 = (ushort *)FUN_00d18080(param_1);
    }
    if ((local_8 == (ushort *)0x0) && (puVar3 != (ushort *)0x0)) {
      FUN_00d1ee90((int)puVar3);
    }
```

Code is fully decompiled and while you browse each function, the decompiled code is showed in the right window.

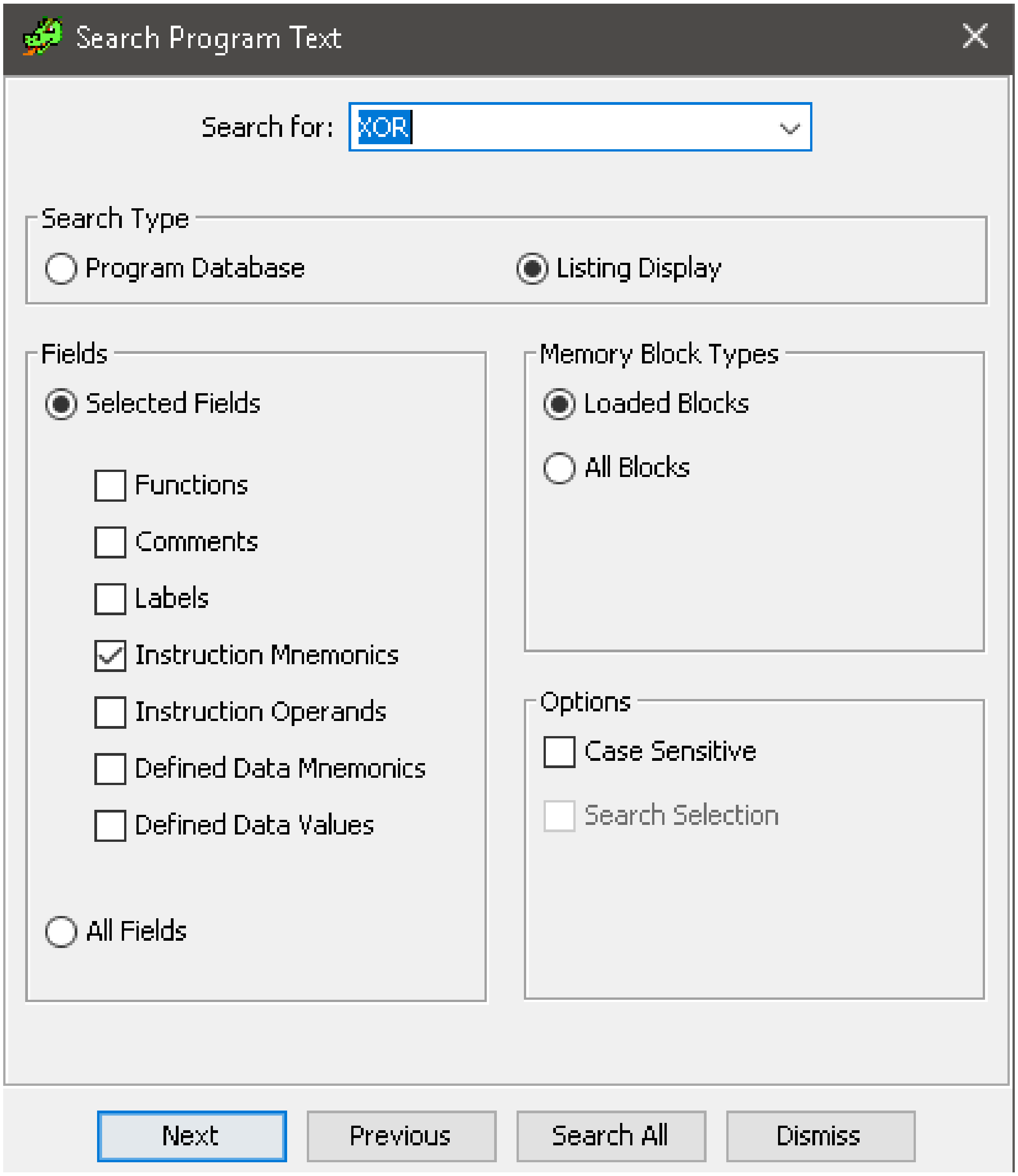
Browsing Code

Browsing code is similar to IDA, you can double-click a name to jump there (for example double-clicking the destination of a call <destination>, would take you to the destination function). You can move easily to the previous location with Alt+left (equivalent to Esc in IDA) and next location with Alt+right (equivalent to Ctrl+Enter in IDA).

Other navigation options:


Clear History
Go To... G
Go To Symbol Source F3
Go To Next Function Ctrl+Down
Go To Previous Function Ctrl+Up
Go To Program... Ctrl+F7
Go To Last Active Program Ctrl+F6
Next Selected Range Ctrl+Right Brace
Previous Selected Range Ctrl+Left Brace
Next Highlight Range Ctrl+0
Previous Highlight Range Ctrl+9
Next Color Range
Previous Color Range
Toggle Code Unit Search Direction Ctrl+Alt+T
Next Instruction Ctrl+Alt+I
Next Data Ctrl+Alt+D
Next Undefined Ctrl+Alt+U
Next Label Ctrl+Alt+L
Next Function Ctrl+Alt+F
Next Instruction Not In a Function Ctrl+Alt+N
Next Different Byte Value Ctrl+Alt+V
Next Bookmark Ctrl+Alt+B

You can search for text, like IDA Alt+t, however (and I found this an interesting characteristic), you can select where do you want the text is going to be searched:



Search Program Text

Search for: XOR

Search Type

○ Program Database ● Listing Display

Fields

● Selected Fields

□ Functions

□ Comments

□ Labels

☑ Instruction Mnemonics

□ Instruction Operands

□ Defined Data Mnemonics

□ Defined Data Values

○ All Fields

Memory Block Types

● Loaded Blocks

○ All Blocks

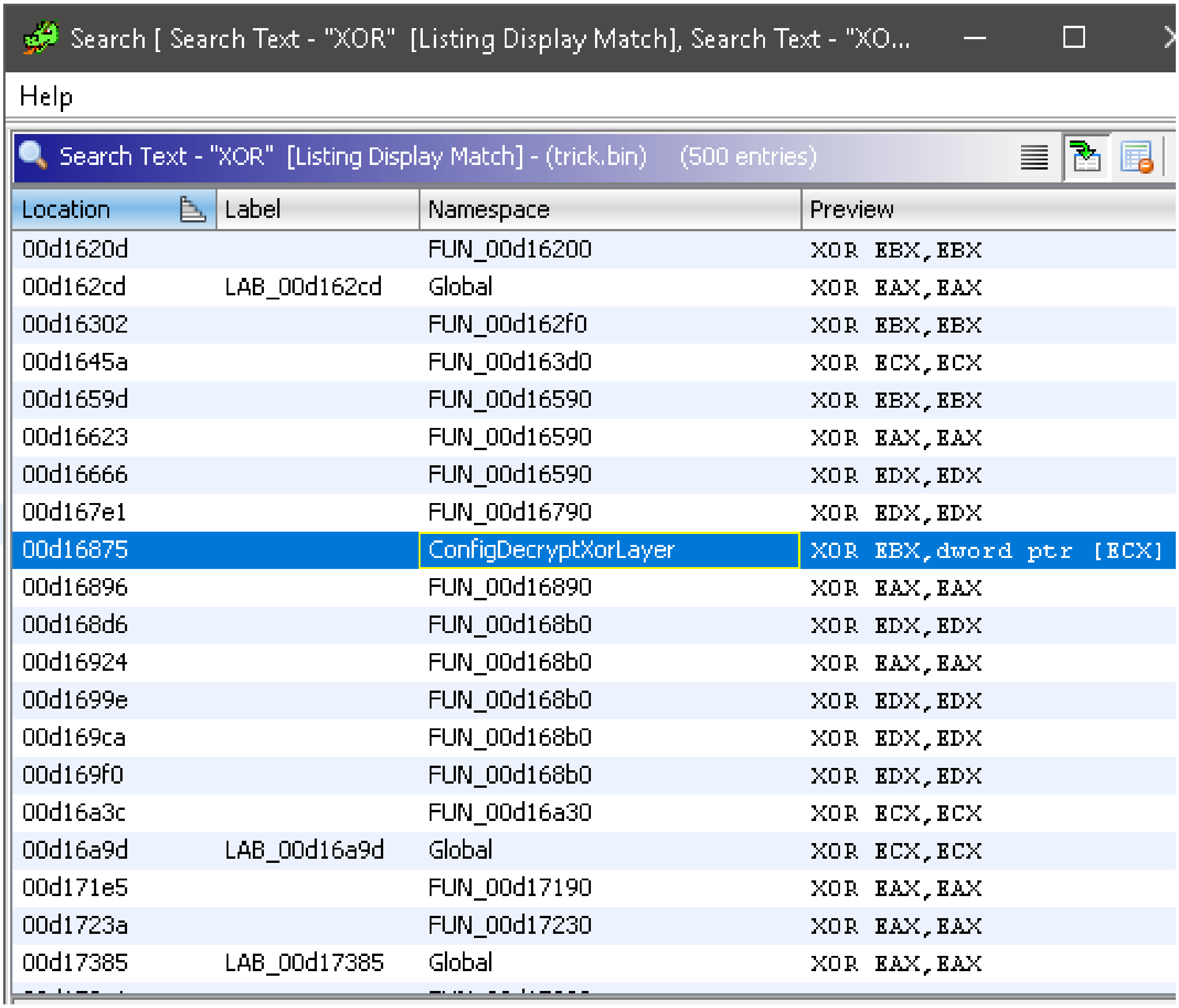
Options

□ Case Sensitive

□ Search Selection

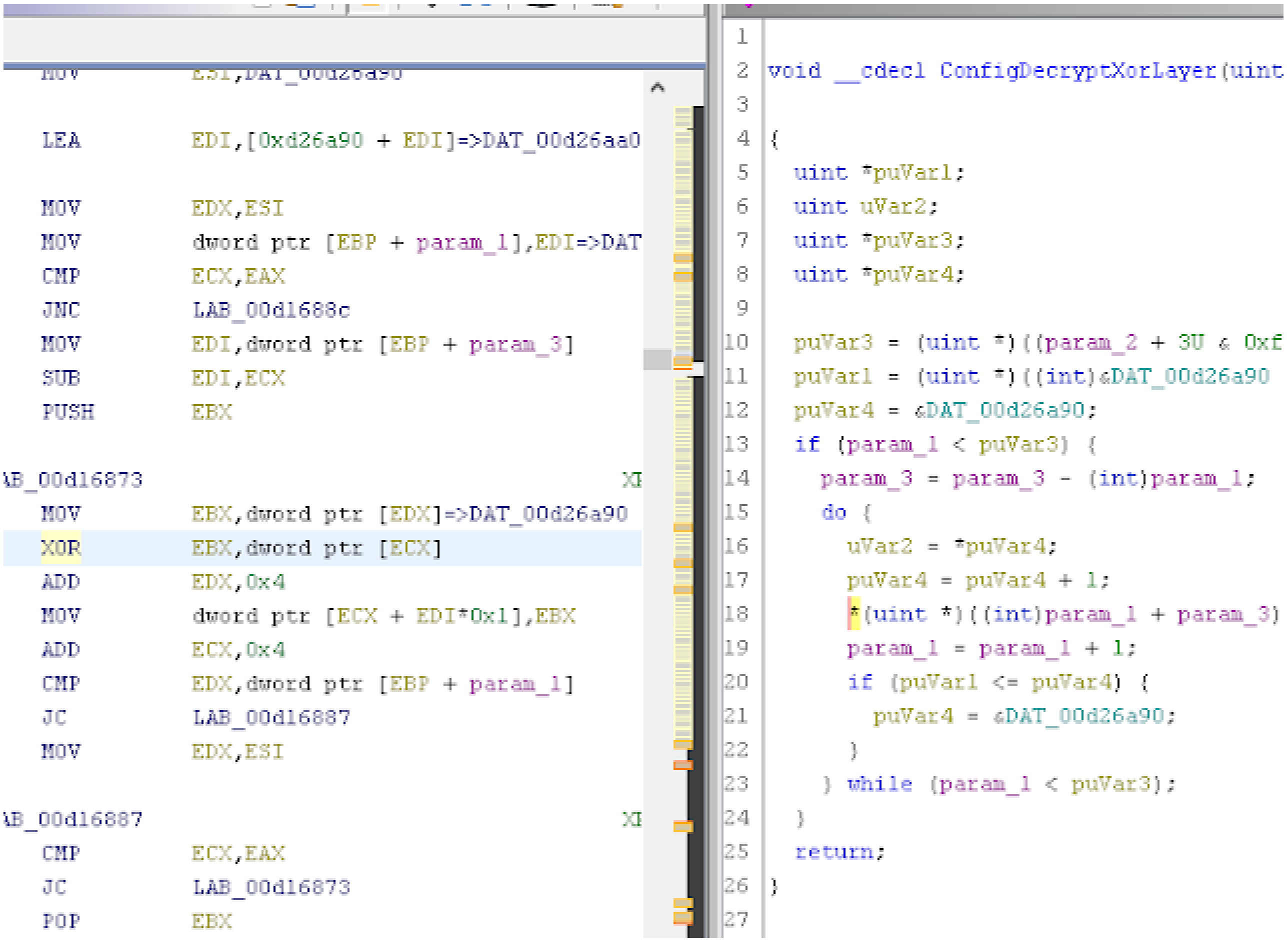
Find TrickBot Config Xor-layer Decryptor

For example, we can try to search for XOR instructions, and we get a list of matches:


Search [Search Text - "XOR" [Listing Display Match], Search Text - "XO...
Help
Search Text - "XOR" [Listing Display Match] - (trick.bin) (500 entries)

Location	Label	Namespace	Preview
00d1620d		FUN_00d16200	XOR EBX,EBX
00d162cd	LAB_00d162cd	Global	XOR EAX,EAX
00d16302		FUN_00d162f0	XOR EBX,EBX
00d1645a		FUN_00d163d0	XOR ECX,ECX
00d1659d		FUN_00d16590	XOR EBX,EBX
00d16623		FUN_00d16590	XOR EAX,EAX
00d16666		FUN_00d16590	XOR EDX,EDX
00d167e1		FUN_00d16790	XOR EDX,EDX
00d16875		ConfigDecryptXorLayer	XOR EBX,dword ptr [ECX]
00d16896		FUN_00d16890	XOR EAX,EAX
00d168d6		FUN_00d168b0	XOR EDX,EDX
00d16924		FUN_00d168b0	XOR EAX,EAX
00d1699e		FUN_00d168b0	XOR EDX,EDX
00d169ca		FUN_00d168b0	XOR EDX,EDX
00d169f0		FUN_00d168b0	XOR EDX,EDX
00d16a3c		FUN_00d16a30	XOR ECX,ECX
00d16a9d	LAB_00d16a9d	Global	XOR ECX,ECX
00d171e5		FUN_00d17190	XOR EAX,EAX
00d1723a		FUN_00d17230	XOR EAX,EAX
00d17385	LAB_00d17385	Global	XOR EAX,EAX

In the analyzed sample (a trickbot from early 2019), if we look for XOR instructions, we can find easily some XOR instructions modifying memory, and one of them belongs to the function that decrypts the XOR layer of the trickbot config:



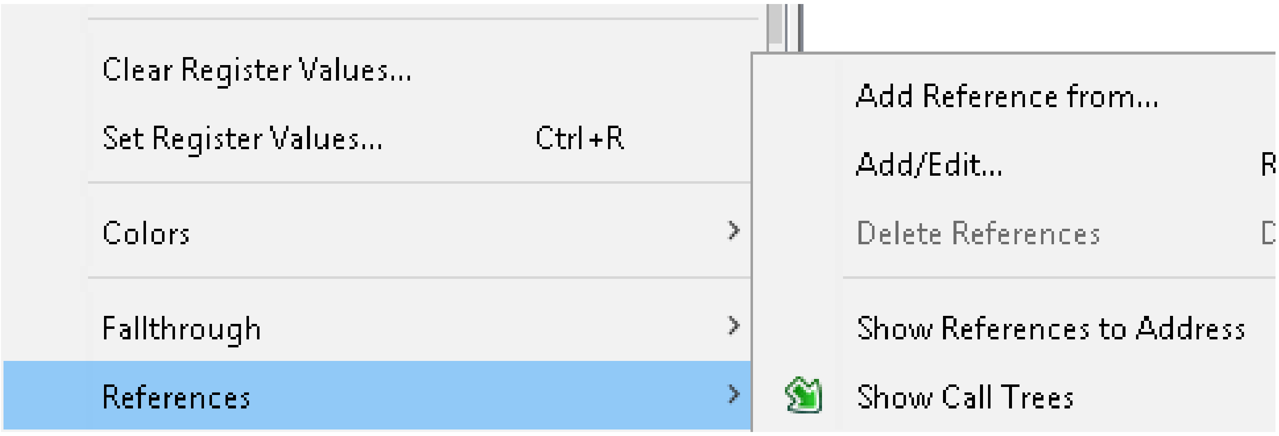
```
    MOV        ESI,DAT_00d26a90

    LEA        EDI,[0xd26a90 + EDI]=>DAT_00d26aa0

    MOV        EDX,ESI
    MOV        dword ptr [EBP + param_1],EDI=>DAT
    CMP        ECX,EAX
    JNC        LAB_00d1688c
    MOV        EDI,dword ptr [EBP + param_3]
    SUB        EDI,ECX
    PUSH       EBX

AB_00d16873                                     XI
    MOV        EBX,dword ptr [EDX]=>DAT_00d26a90
    XOR        EBX,dword ptr [ECX]
    ADD        EDX,0x4
    MOV        dword ptr [ECX + EDI*0x1],EBX
    ADD        ECX,0x4
    CMP        EDX,dword ptr [EBP + param_1]
    JC         LAB_00d16887
    MOV        EDX,ESI

AB_00d16887                                     XI
    CMP        ECX,EAX
    JC         LAB_00d16873
    POP        EBX
```

```

void __cdecl ConfigDecryptXorLayer(uint

{
  uint *puVar1;
  uint uVar2;
  uint *puVar3;
  uint *puVar4;

  puVar3 = (uint *)((param_2 + 3U & 0xf
  puVar1 = (uint *)((int)&DAT_00d26a90
  puVar4 = &DAT_00d26a90;
  if (param_1 < puVar3) {
    param_3 = param_3 - (int)param_1;
    do {
      uVar2 = *puVar4;
      puVar4 = puVar4 + 1;
      *(uint *)((int)param_1 + param_3)
      param_1 = param_1 + 1;
      if (puVar1 <= puVar4) {
        puVar4 = &DAT_00d26a90;
      }
    } while (param_1 < puVar3);
  }
  return;
}

```

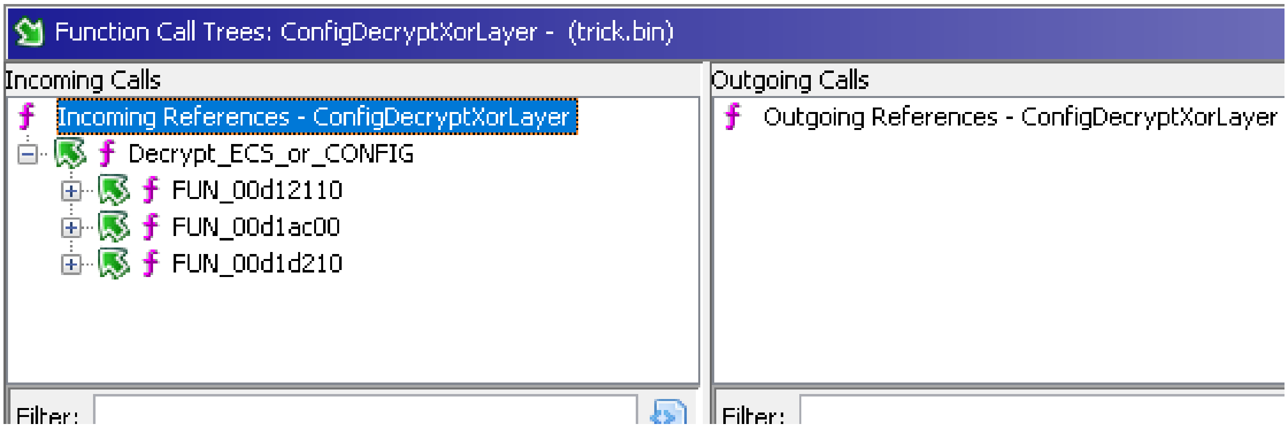
(Btw, as we can see in the image, when you select with the mouse a line in the disassembly window, the equivalent line is highlighted in the decompiled window).

Using references to find more interesting parts of the code

Once you have located an interesting point in the code, you can show a tree of calls to that point:



The tree makes easy to follow the incoming or outgoing references to the interesting function:



Function Call Trees: ConfigDecryptXorLayer - (trick.bin)

Incoming Calls

```
Incoming References - ConfigDecryptXorLayer
  Decrypt_ECS_or_CONFIG
    FUN_00d12110
    FUN_00d1ac00
    FUN_00d1d210
```

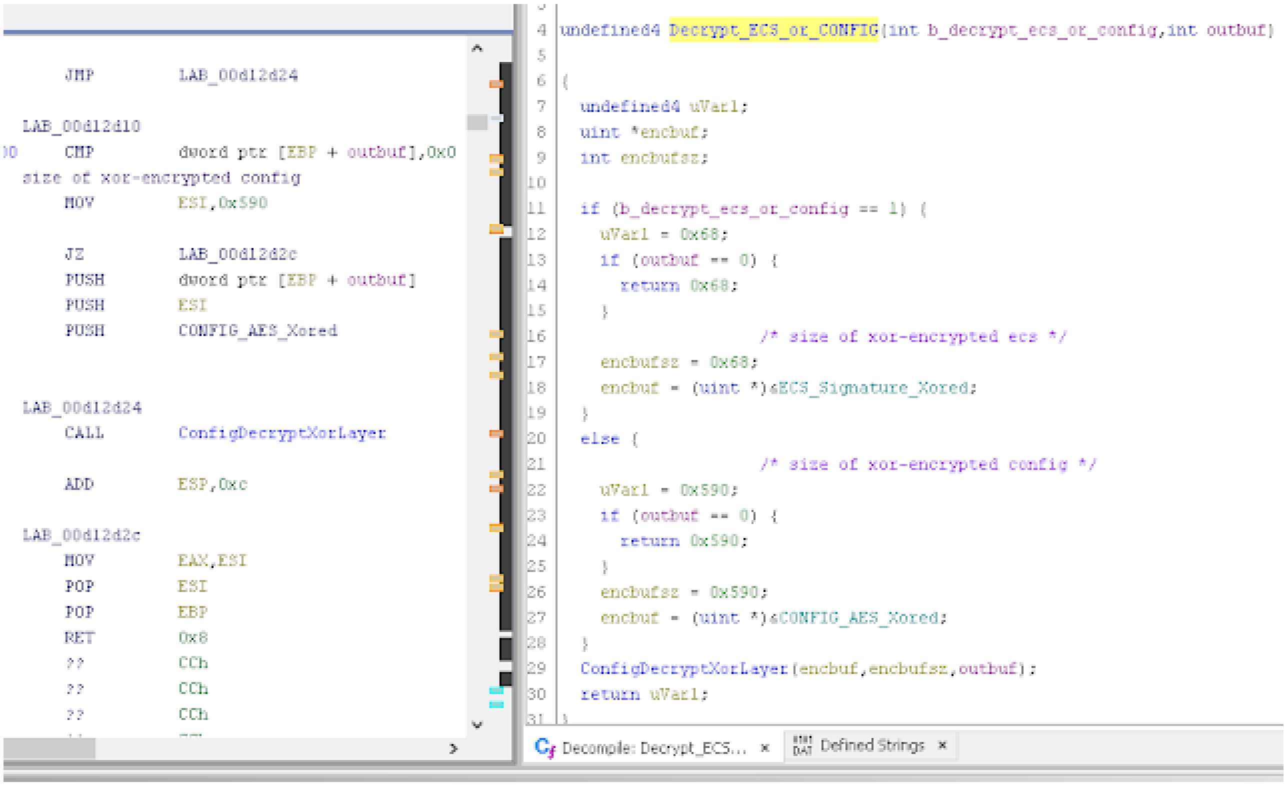
Outgoing Calls

```
Outgoing References - ConfigDecryptXorLayer
```

Additionally, you could highlight (select) back or forward refs to an address in the disassembly and decompiled windows.

TrickBot ECS signature and Config Xor Decryptor

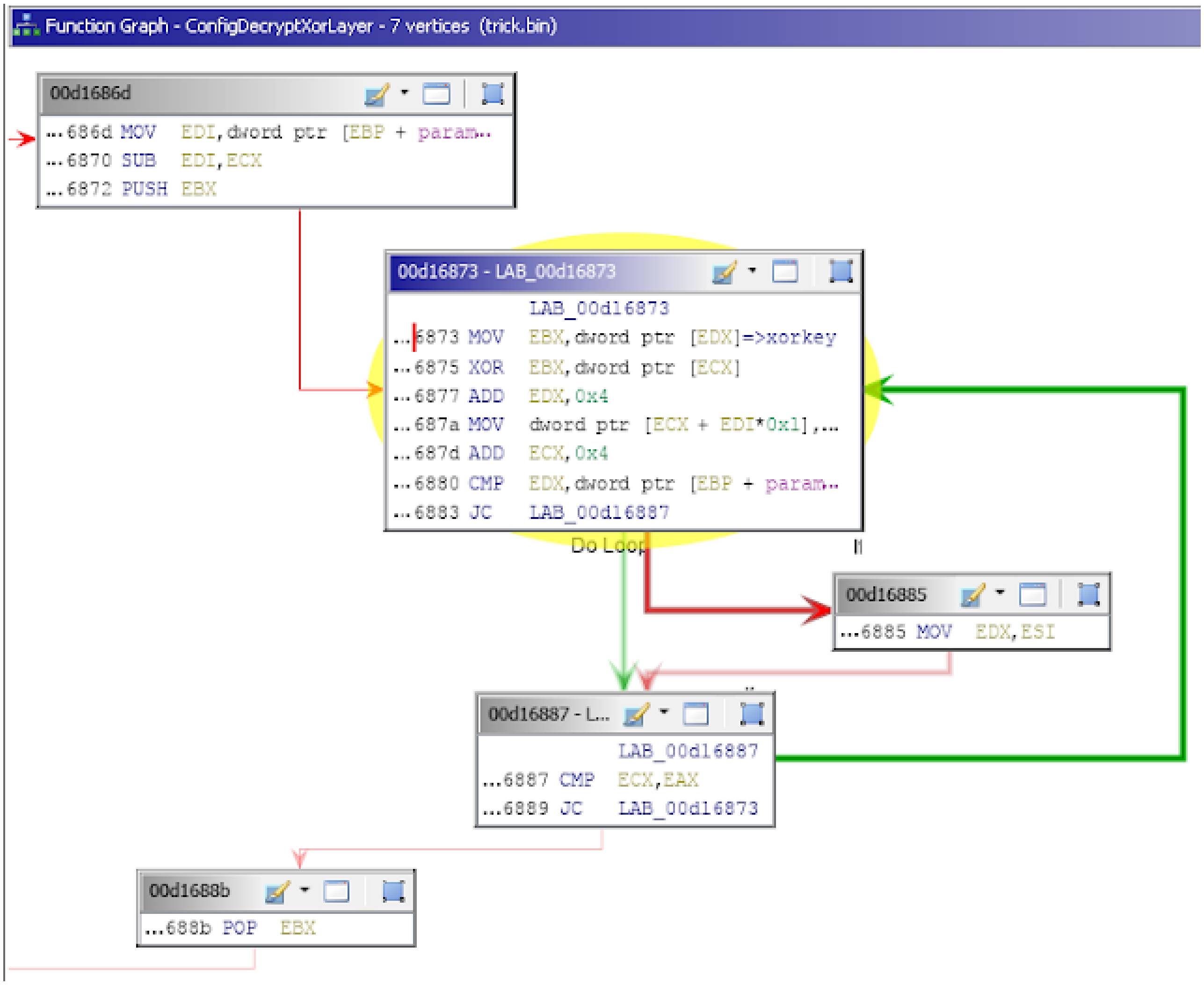
By using the call trees, we can find easily the functions that decrypts the XOR layer of the elliptic curve signature or the XOR layer of the TrickBot Config:



```
        JMP          LAB_00d12d24

   LAB_00d12d10
00      CMP          dword ptr [EBP + outbuf],0x0
   size of xor-encrypted config
        MOV          ESI,0x590

        JZ           LAB_00d12d2c
        PUSH         dword ptr [EBP + outbuf]
        PUSH         ESI
        PUSH         CONFIG_AES_Xored


   LAB_00d12d24
        CALL         ConfigDecryptXorLayer

        ADD          ESP,0xc

   LAB_00d12d2c
        MOV          EAX,ESI
        POP          ESI
        POP          EBP
        RET          0x8
        ??           CCh
        ??           CCh
        ??           CCh
```

```
undefined4 Decrypt_ECS_or_CONFIG(int b_decrypt_ecs_or_config,int outbuf)

{
  undefined4 uVar1;
  uint *encbuf;
  int encbufsz;

  if (b_decrypt_ecs_or_config == 1) {
    uVar1 = 0x68;
    if (outbuf == 0) {
      return 0x68;
    }
                    /* size of xor-encrypted ecs */
    encbufsz = 0x68;
    encbuf = (uint *)&ECS_Signature_Xored;
  }
  else {
                    /* size of xor-encrypted config */
    uVar1 = 0x590;
    if (outbuf == 0) {
      return 0x590;
    }
    encbufsz = 0x590;
    encbuf = (uint *)&CONFIG_AES_Xored;
  }
  ConfigDecryptXorLayer(encbuf,encbufsz,outbuf);
  return uVar1;
}
```

In addition, you can open a function graph window, similar to IDA graphs. Here is the XOR decryptor loop of TrickBot:



Function Graph - ConfigDecryptXorLayer - 7 vertices (trick.bin)

```
00d1686d
...686d MOV  EDI,dword ptr [EBP + param...
...6870 SUB  EDI,ECX
...6872 PUSH EBX
```

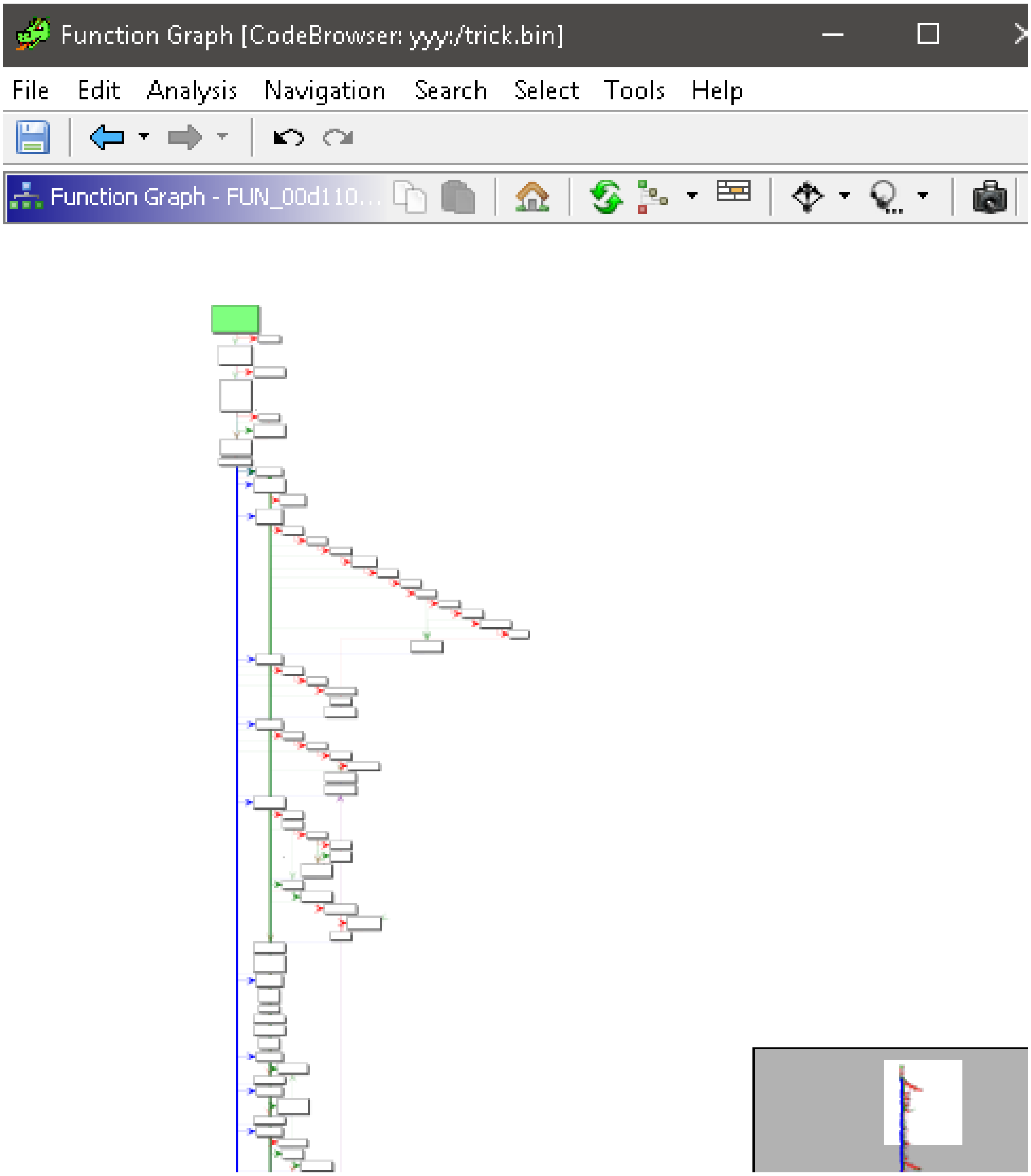
```
00d16873 - LAB_00d16873
             LAB_00d16873
...6873 MOV  EBX,dword ptr [EDX]=>xorkey
...6875 XOR  EBX,dword ptr [ECX]
...6877 ADD  EDX,0x4
...687a MOV  dword ptr [ECX + EDI*0x1],...
...687d ADD  ECX,0x4
...6880 CMP  EDX,dword ptr [EBP + param...
...6883 JC   LAB_00d16887
```

```
00d16885
...6885 MOV  EDX,ESI
```

```
00d16887 - L...
             LAB_00d16887
...6887 CMP  ECX,EAX
...6889 JC   LAB_00d16873
```

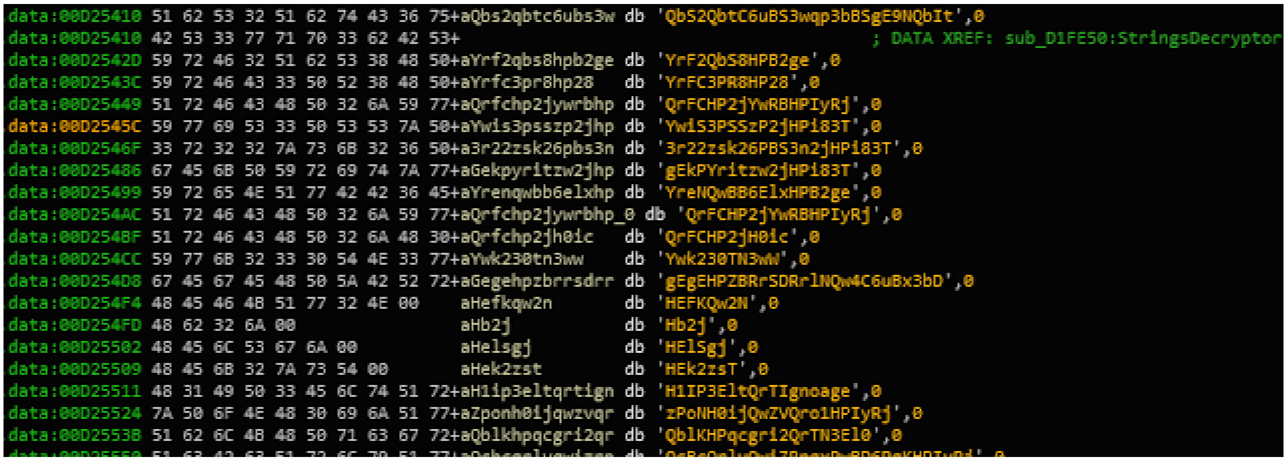
```
00d1688b
...688b POP  EBX
```

You can move easily on the graph, and zoom in/out with the mouse wheel:


Function Graph [CodeBrowser: yyy:/trick.bin]
File
Edit
Analysis
Navigation
Search
Select
Tools
Help
Function Graph - FUN_00d110...

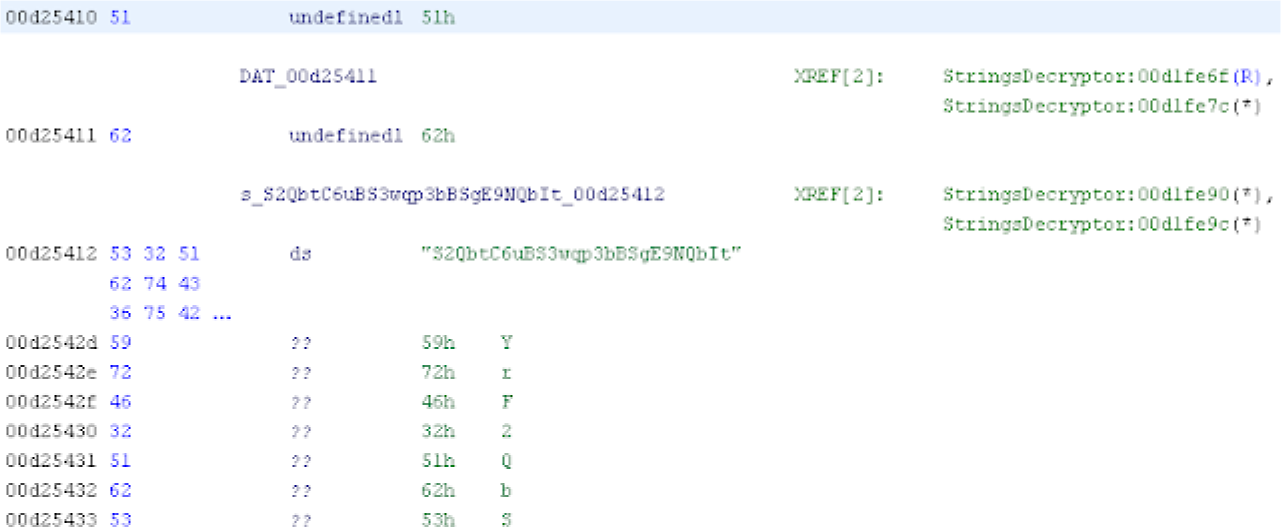
TrickBot Strings Decryptor

About strings.. All the strings used by the newer versions of TrickBot are encrypted. While IDA was able to construct a nice table of strings that makes easy to find the decryptor:



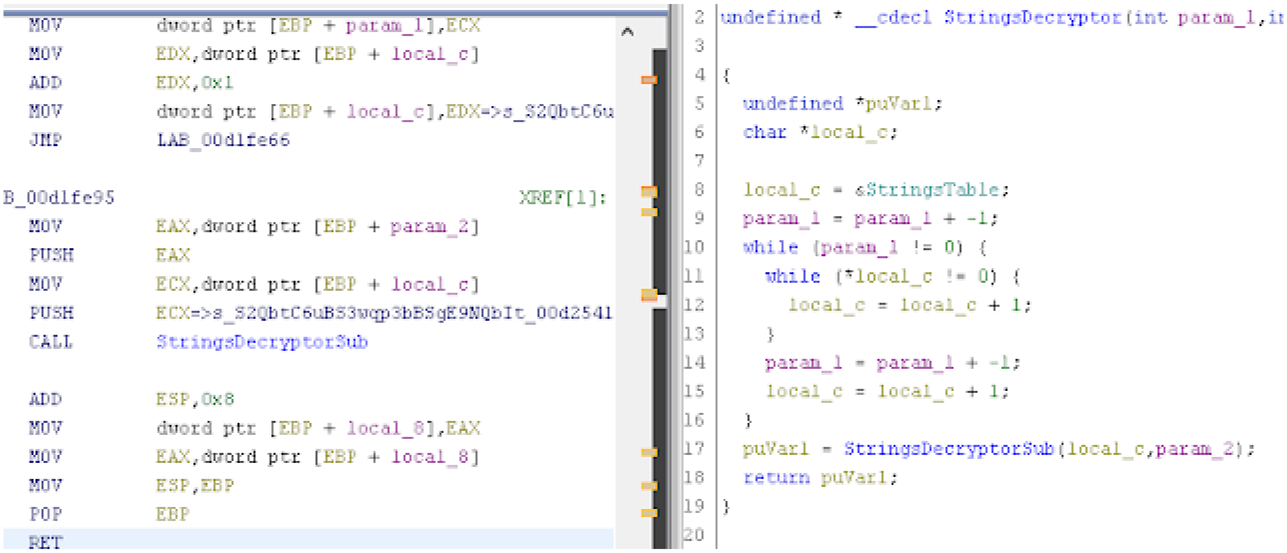
```
.data:00D25410 51 62 53 32 51 62 74 43 36 75+aQbs2qbtc6ubs3w db 'QbS2QbtC6uBS3wqp3bBSgE9NQbIt',0
.data:00D25410 42 53 33 77 71 70 33 62 42 53+                                      ; DATA XREF: sub_D1FE50:StringsDecryptor
.data:00D2542D 59 72 46 32 51 62 53 38 48 50+aYrf2qbs8hpb2ge db 'YrF2QbS8HPB2ge',0
.data:00D2543C 59 72 46 43 33 50 52 38 48 50+aYrfc3pr8hp28   db 'YrFC3PR8HP28',0
.data:00D25449 51 72 46 43 48 50 32 6A 59 77+aQrfchp2jywrbhp db 'QrFCHP2jYwRBHPIyRj',0
.data:00D2545C 59 77 69 53 33 50 53 53 7A 50+aYwis3psszp2jhp db 'YwiS3PSSzP2jHPi83T',0
.data:00D2546F 33 72 32 32 7A 73 6B 32 36 50+a3r22zsk26pbs3n db '3r22zsk26PBS3n2jHPi83T',0
.data:00D25486 67 45 6B 50 59 72 69 74 7A 77+aGekpyritzw2jhp db 'gEkPYritzw2jHPi83T',0
.data:00D25499 59 72 65 4E 51 77 42 42 36 45+aYrenqwbb6elxhp db 'YreNQwBB6ElxHPB2ge',0
.data:00D254AC 51 72 46 43 48 50 32 6A 59 77+aQrfchp2jywrbhp_0 db 'QrFCHP2jYwRBHPIyRj',0
.data:00D254BF 51 72 46 43 48 50 32 6A 48 30+aQrfchp2jh0ic   db 'QrFCHP2jH0ic',0
.data:00D254CC 59 77 6B 32 33 30 54 4E 33 77+aYwk230tn3ww    db 'Ywk230TN3wW',0
.data:00D254D8 67 45 67 45 48 50 5A 42 52 72+aGegehpzbrrsdrr db 'gEgEHPZBRrSDRrlNQw4C6uBx3bD',0
.data:00D254F4 48 45 46 4B 51 77 32 4E 00       aHefkqw2n       db 'HEFKQw2N',0
.data:00D254FD 48 62 32 6A 00                   aHb2j           db 'Hb2j',0
.data:00D25502 48 45 6C 53 67 6A 00             aHelsgj         db 'HElSgj',0
.data:00D25509 48 45 6B 32 7A 73 54 00          aHek2zst        db 'HEk2zsT',0
.data:00D25511 48 31 49 50 33 45 6C 74 51 72+aH1ip3eltqrtign db 'H1IP3EltQrTIgnoage',0
.data:00D25524 7A 50 6F 4E 48 30 69 6A 51 77+aZponh0ijqwzvqr db 'zPoNH0ijQwZVQro1HPIyRj',0
.data:00D25538 51 62 6C 4B 48 50 71 63 67 72+aQblkhpqcgri2qr db 'QblKHPqcgri2QrTN3El0',0
```

Ghidra were not able to identify all the strings and construct a nice table, it is much lesser intuitive:



```
00d25410 51              undefined1  51h

                    DAT_00d25411                          XREF[2]:     StringsDecryptor:00d1fe6f(R),
                                                                       StringsDecryptor:00d1fe7c(*)
00d25411 62              undefined1  62h

                    s_S2QbtC6uBS3wqp3bBSgE9NQbIt_00d25412  XREF[2]:     StringsDecryptor:00d1fe90(*),
                                                                       StringsDecryptor:00d1fe9c(*)
00d25412 53 32 51        ds          "S2QbtC6uBS3wqp3bBSgE9NQbIt"
         62 74 43
         36 75 42 ...
00d2542d 59              ??          59h    Y
00d2542e 72              ??          72h    r
00d2542f 46              ??          46h    F
00d25430 32              ??          32h    2
00d25431 51              ??          51h    Q
00d25432 62              ??          62h    b
00d25433 53              ??          53h    S
```

Maybe I missed something with Ghidra, but I selected the option Analysis->One shot->Ascii Strings, and these are the results. This makes difficult, for example, to find strings' decryptors.



```
  MOV        dword ptr [EBP + param_1],ECX
  MOV        EDX,dword ptr [EBP + local_c]
  ADD        EDX,0x1
  MOV        dword ptr [EBP + local_c],EDX=>s_S2QbtC6u
  JMP        LAB_00d1fe66

B_00d1fe95                                   XREF[1]:
  MOV        EAX,dword ptr [EBP + param_2]
  PUSH       EAX
  MOV        ECX,dword ptr [EBP + local_c]
  PUSH       ECX=>s_S2QbtC6uBS3wqp3bBSgE9NQbIt_00d2541
  CALL       StringsDecryptorSub

  ADD        ESP,0x8
  MOV        dword ptr [EBP + local_8],EAX
  MOV        EAX,dword ptr [EBP + local_8]
  MOV        ESP,EBP
  POP        EBP
  RET
```

```
undefined * __cdecl StringsDecryptor(int param_1,in

{
  undefined *puVar1;
  char *local_c;

  local_c = &StringsTable;
  param_1 = param_1 + -1;
  while (param_1 != 0) {
    while (*local_c != 0) {
      local_c = local_c + 1;
    }
    param_1 = param_1 + -1;
    local_c = local_c + 1;
  }
  puVar1 = StringsDecryptorSub(local_c,param_2);
  return puVar1;
}
```

Conclussion

in spite of the fact that I really love IDA (and WinDbg), I liked this framework, and I will continue using it.