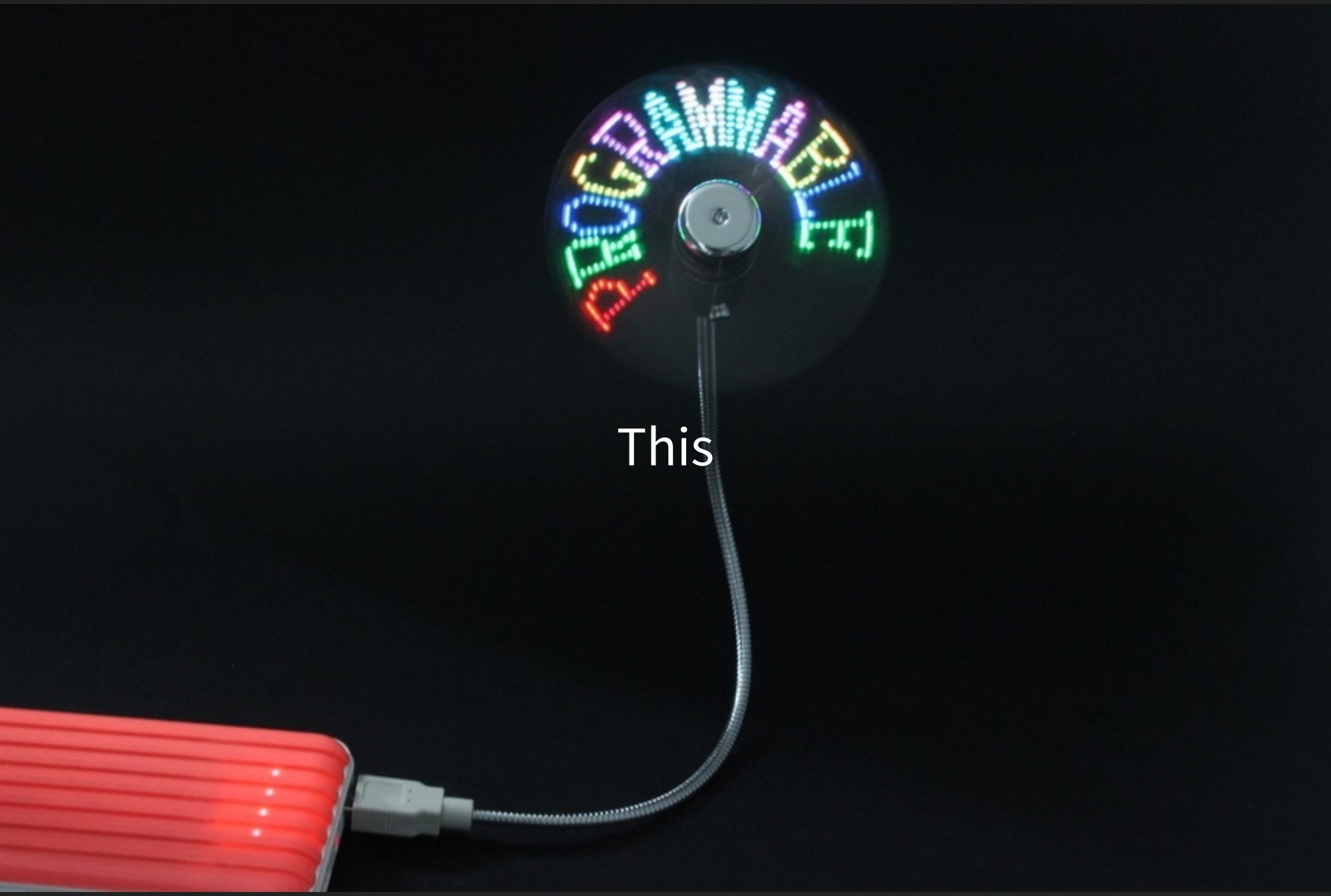
UNDERSTANDING MICRO WAVE

Fergus Symon


Not this...
12:22
Time Cook
Time Defrost
Weight Defrost
QUICK TOUCH PRESETS
Popcorn
Potato
Pizza
Frozen Vegetable
Beverage
Dinner Plate
ONE TOUCH START
1 2 3
4 5 6
7 8 9
START
0 Memory
STOP
ADD 30 SEC
Power Level
Clock
Kitchen Timer
Kenmore


This

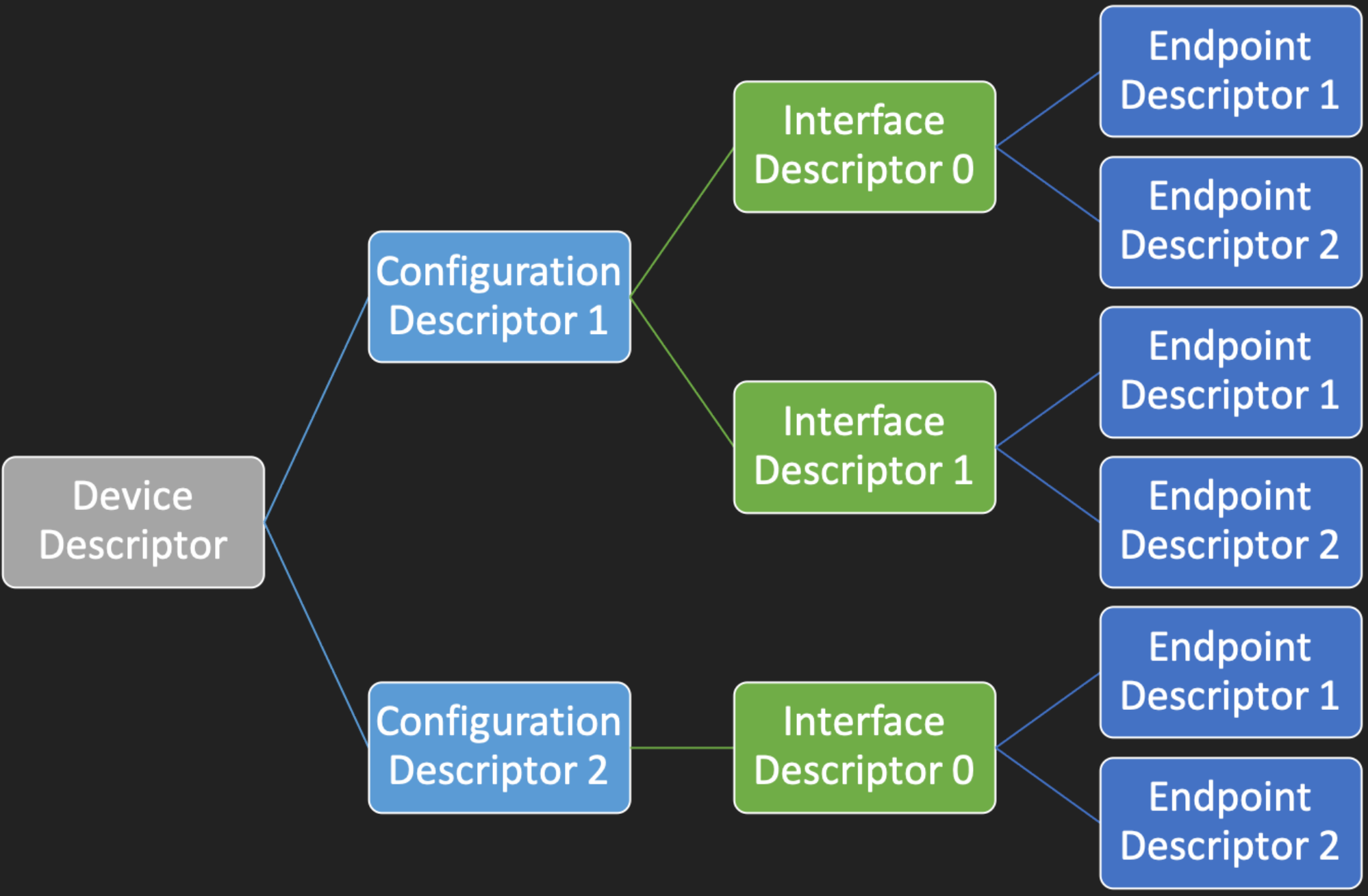
WHY?

I wanted a system for home notifications

- RGB LED Display
- "Computer Programmed" (USB)
- No Driver

UNIVERSAL SERIAL BUS

- Four pins
  - One power (5V)
  - One ground
  - Two transmission (differential pair)
- Brilliant versioning scheme
- Highly standardised


Device Descriptor
Configuration Descriptor 1
Configuration Descriptor 2
Interface Descriptor 0
Interface Descriptor 1
Interface Descriptor 0
Endpoint Descriptor 1
Endpoint Descriptor 2
Endpoint Descriptor 1
Endpoint Descriptor 2
Endpoint Descriptor 1
Endpoint Descriptor 2

- Transactions
- Pipes
  - Control: simple messages
  - Bulk: maximum bandwidth, no guarantees on bandwidth or latency
  - Isochronous: guaranteed bandwidth (best effort)
  - Interrupt: low latency


Back Here

"No Driver" is an interesting claim 🤔

- Can't be a vendor specific device class... right?
- Is there a communications device that has drivers on Windows?
- Is it a mass storage device and use file transfers to operate?
- Can something act as a HID device and just use custom control messages?

Almost! 😐

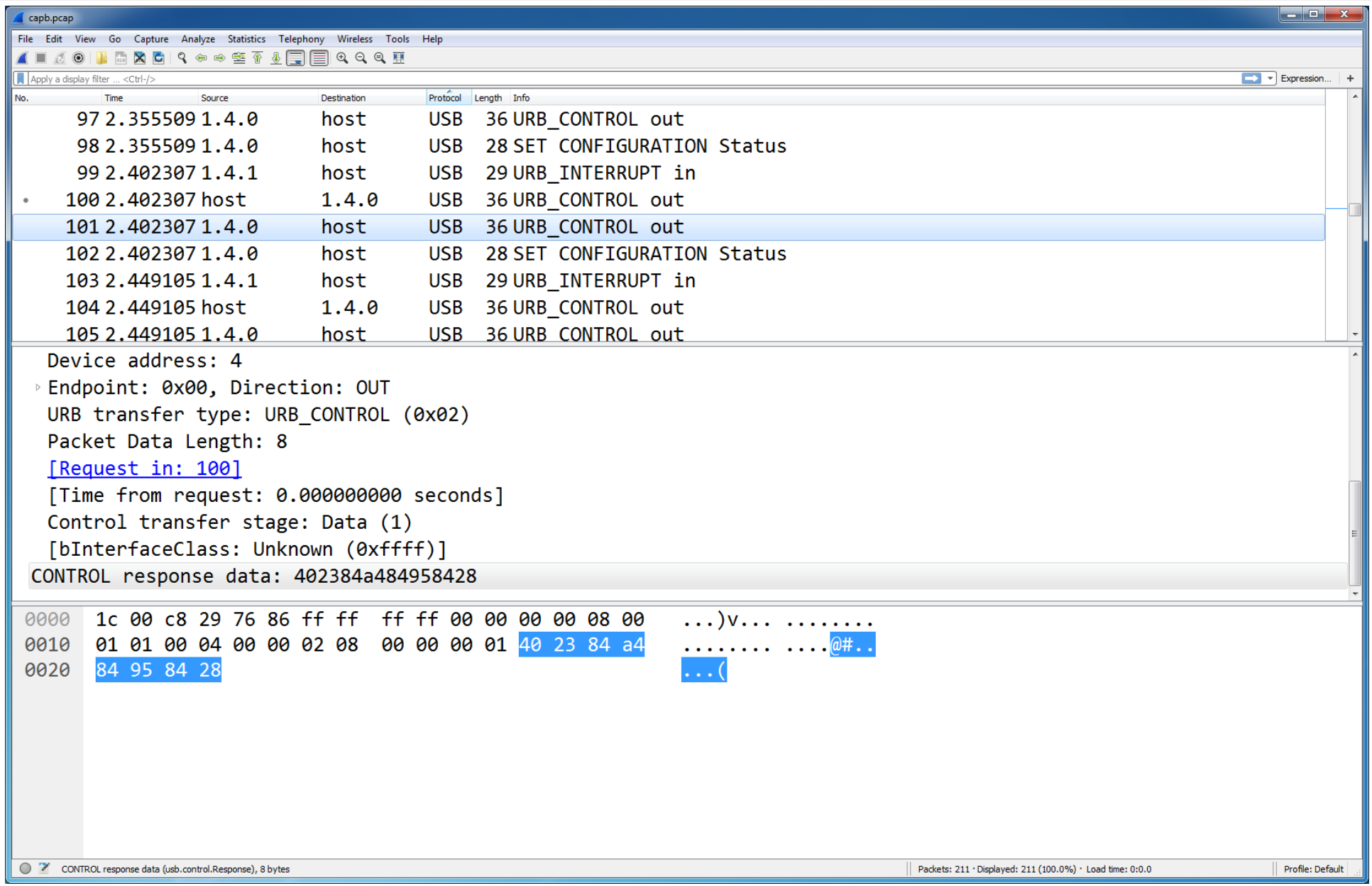
REVERSING USB PROTOCOLS

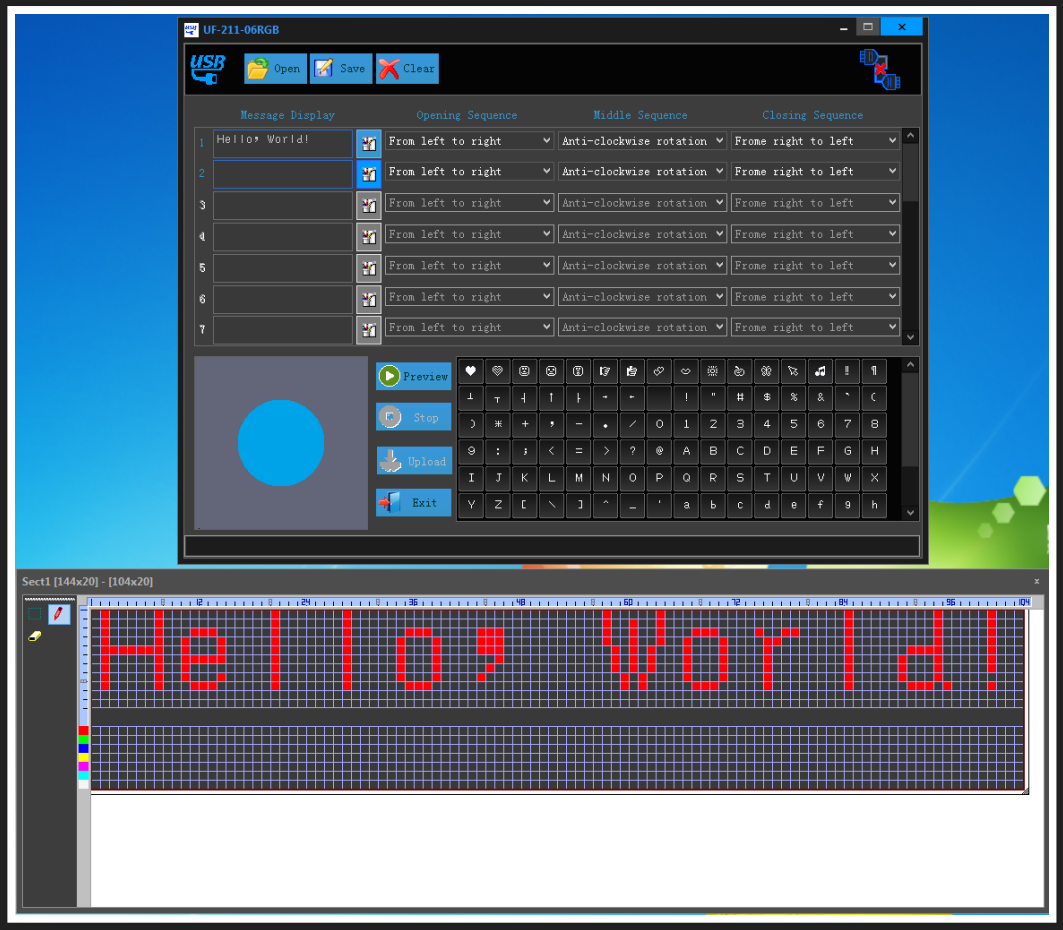
Two approaches:

1. Black-box analysis: capture USB messages determine differences
2. White-box analysis: reverse-engineer the software on one of the end-points
3. ... or both 🤦

Capturing USB messages: 💰 mode

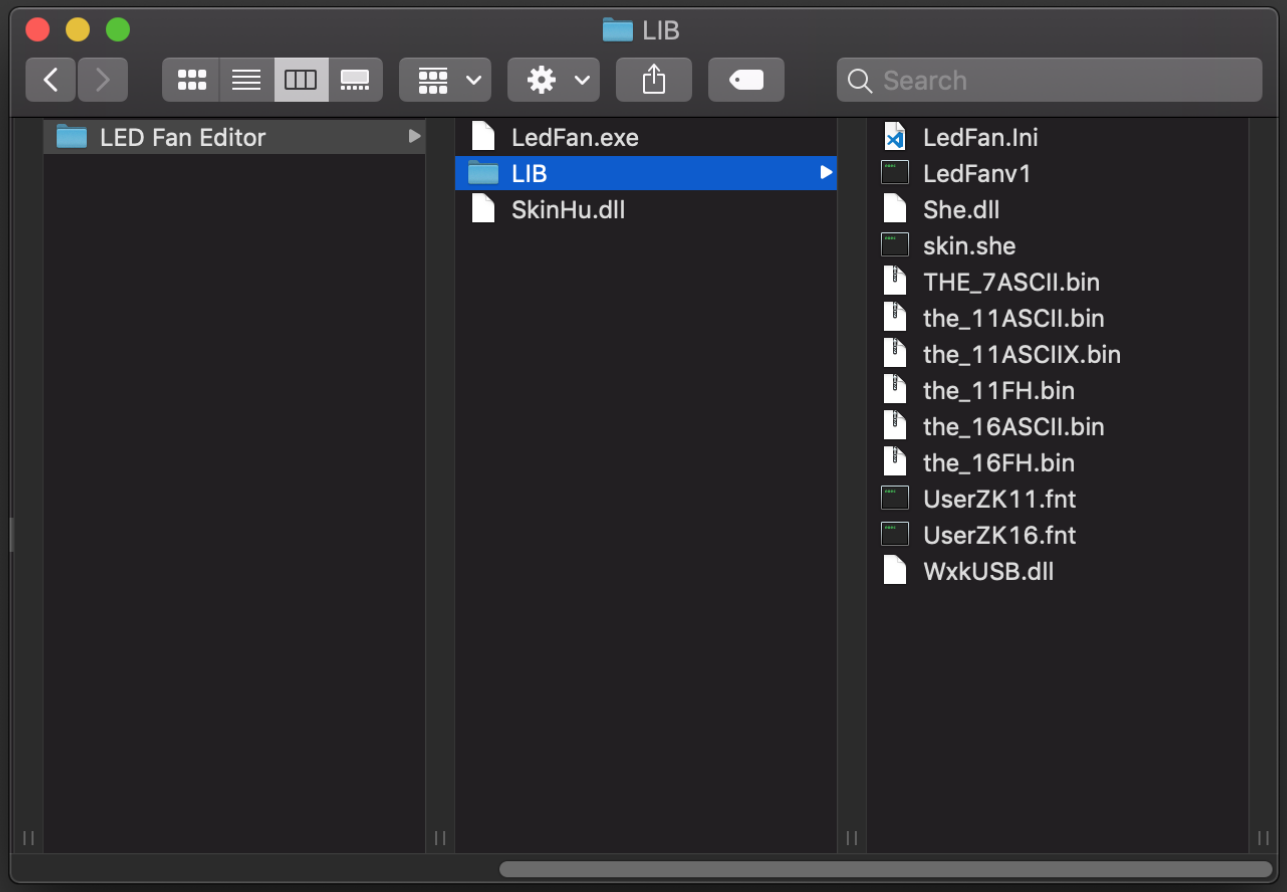
Capturing USB messages: 🙃 mode



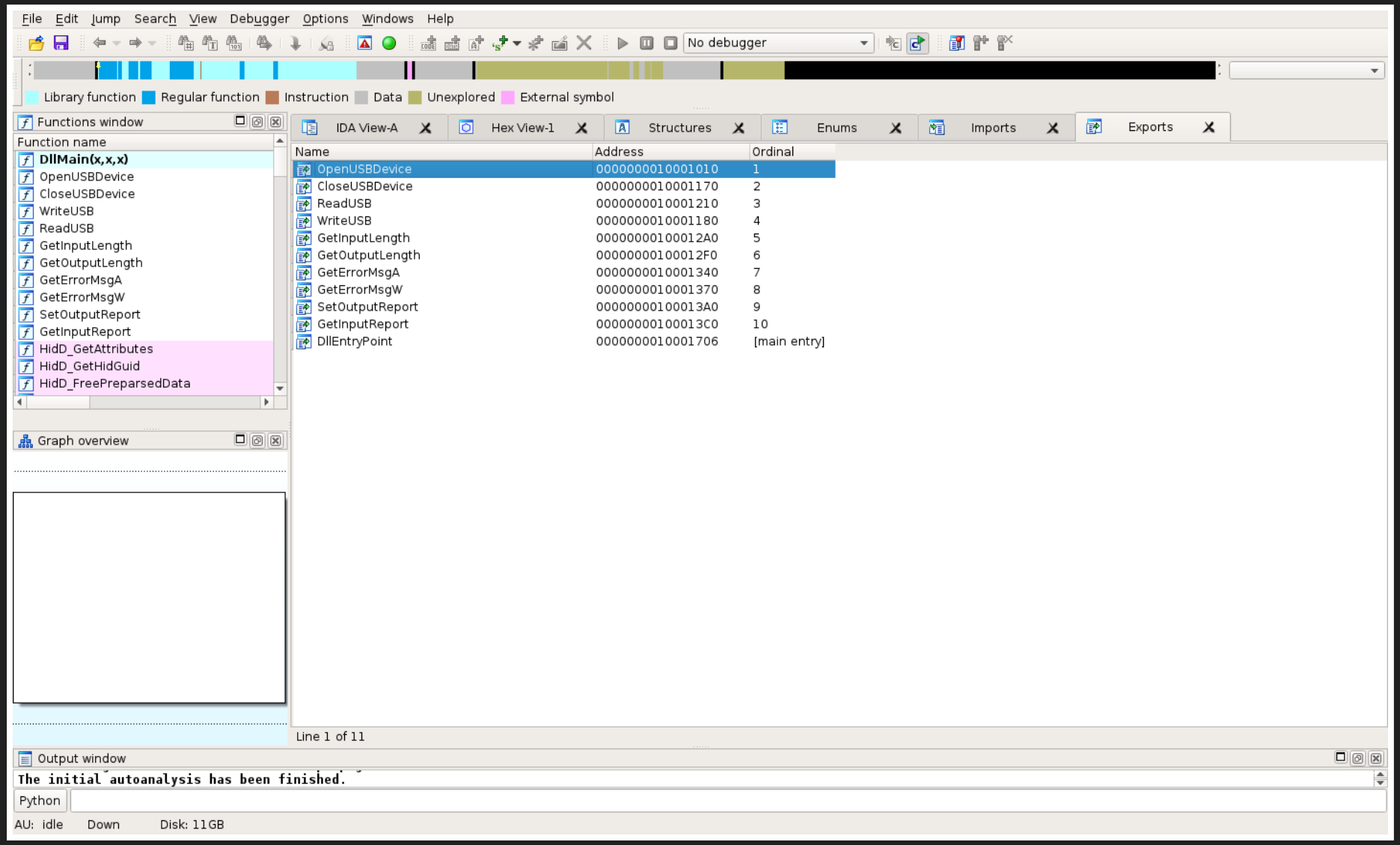


Dynamic analysis alone is useless

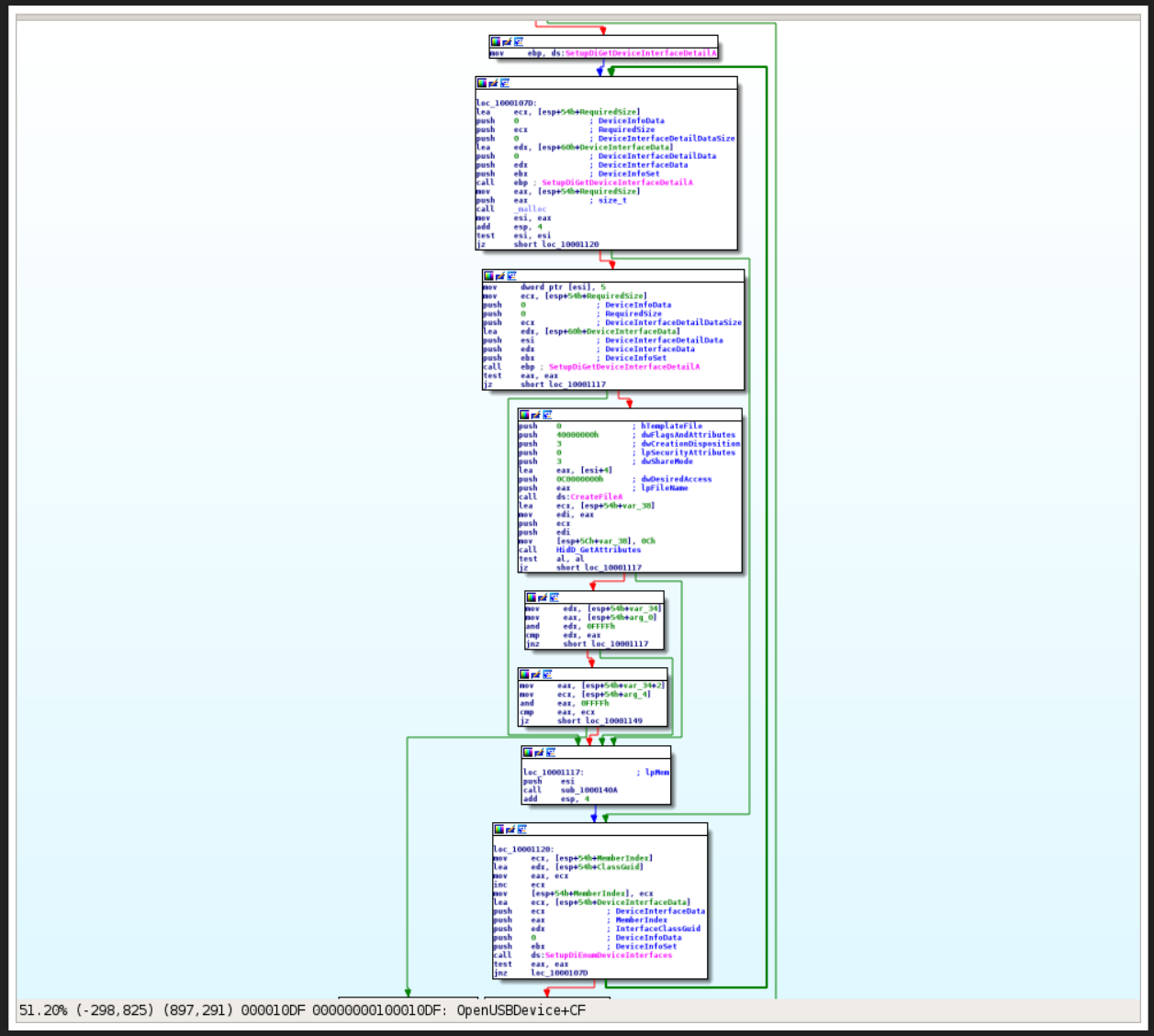
What else can we find?



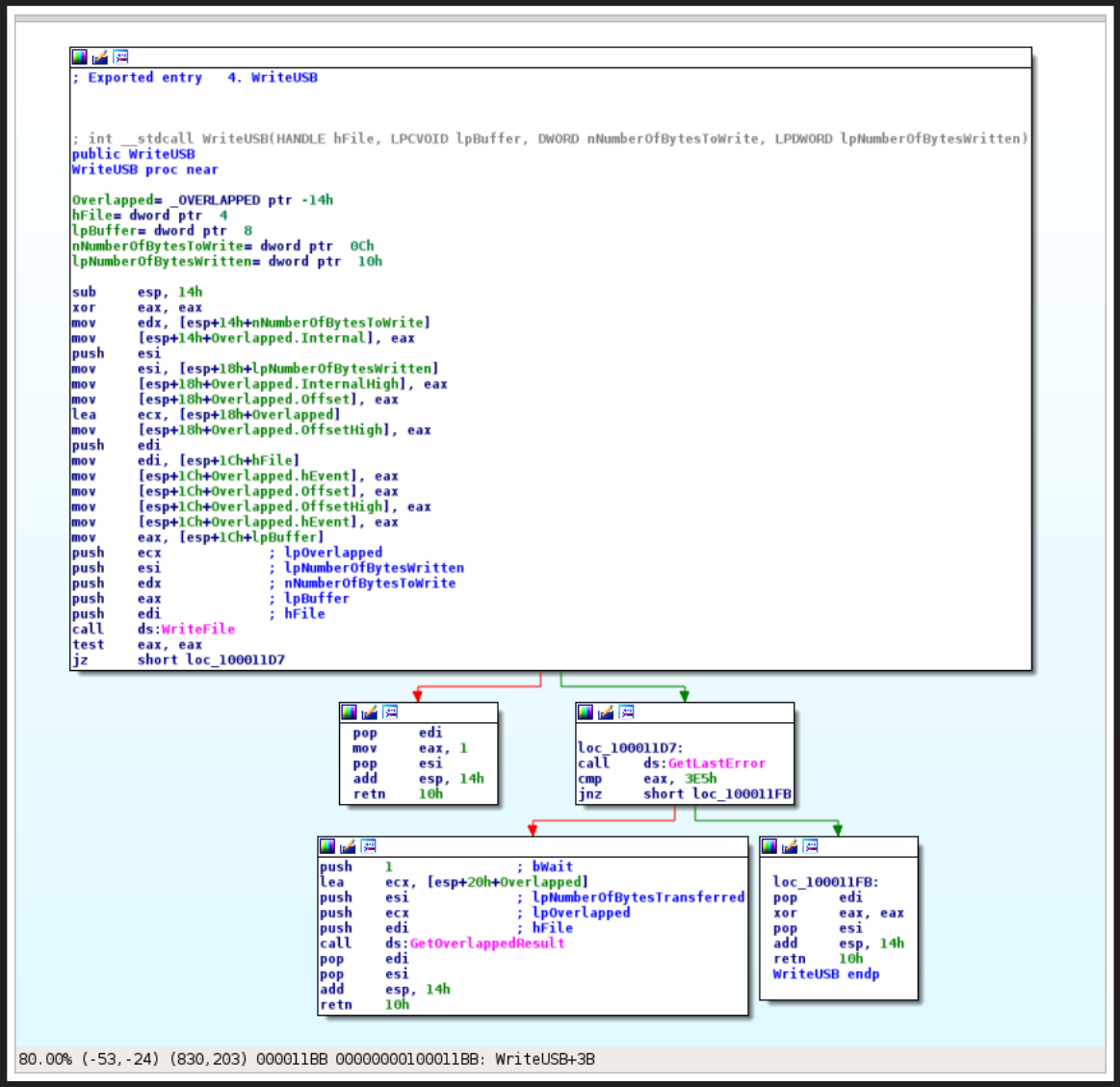
WxkUSB.dll


File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol
Functions window
Function name
DllMain(x,x,x)
OpenUSBDevice
CloseUSBDevice
WriteUSB
ReadUSB
GetInputLength
GetOutputLength
GetErrorMsgA
GetErrorMsgW
SetOutputReport
GetInputReport
HidD_GetAttributes
HidD_GetHidGuid
HidD_FreePreparsedData
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
Name Address Ordinal
OpenUSBDevice 0000000010001010 1
CloseUSBDevice 0000000010001170 2
ReadUSB 0000000010001210 3
WriteUSB 0000000010001180 4
GetInputLength 00000000100012A0 5
GetOutputLength 00000000100012F0 6
GetErrorMsgA 0000000010001340 7
GetErrorMsgW 0000000010001370 8
SetOutputReport 00000000100013A0 9
GetInputReport 00000000100013C0 10
DllEntryPoint 0000000010001706 [main entry]
Line 1 of 11
Output window
The initial autoanalysis has been finished.
Python
AU: idle Down Disk: 11GB

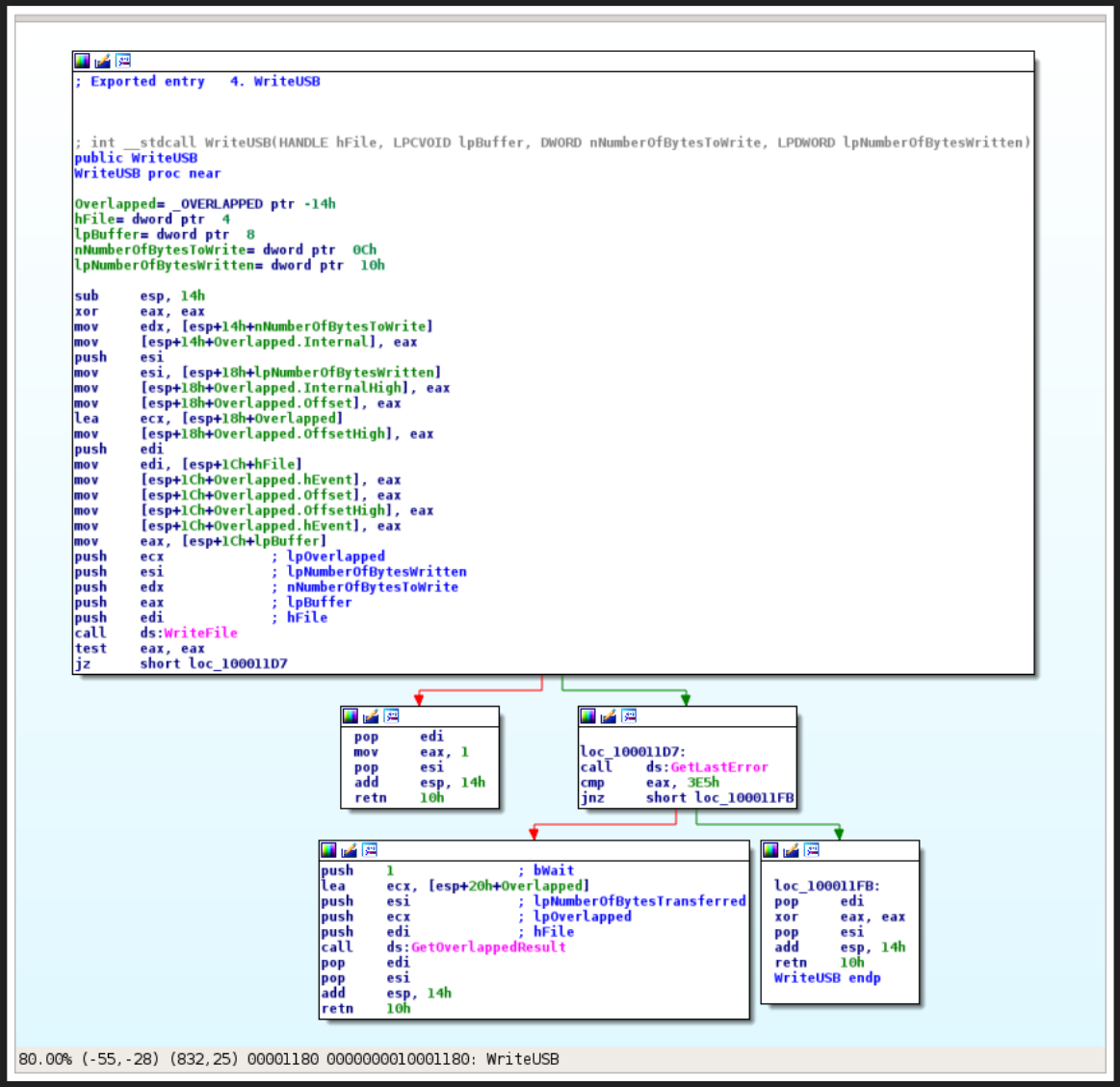
OpenUSBDevice(WORD, WORD)



WriteUSB(HANDLE, LPCVOID, DWORD, LPDWORD)



ReadUSB(HANDLE, LPCVOID, DWORD, LPDWORD)



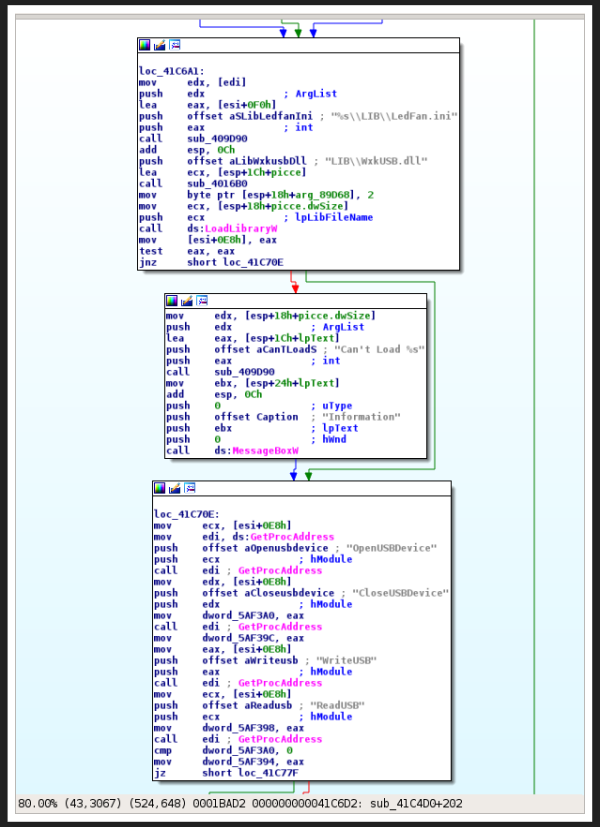
This has been great!

We know it's using HID reports, but nothing about the protocol 😕

What uses `WxkUSB.dll`?

Looking at the import tables of the other binaries: nothing 😞

Hold up, I know this game... 🧐



```
loc_41C6A1:
mov     edx, [edi]
push    edx             ; ArgList
lea     eax, [esi+0F0h]
push    offset aSLibLedfanIni ; "%s\\LIB\\LedFan.ini"
push    eax             ; int
call    sub_409D90
add     esp, 0Ch
push    offset aLibWrkusbDll ; "LIB\\WrkUSB.dll"
lea     ecx, [esp+1Ch+picce]
call    sub_4016B0
mov     byte ptr [esp+18h+arg_89D68], 2
mov     ecx, [esp+18h+picce.dwSize]
push    ecx             ; lpLibFileName
call    ds:LoadLibraryW
mov     [esi+0E8h], eax
test    eax, eax
jnz     short loc_41C70E
```

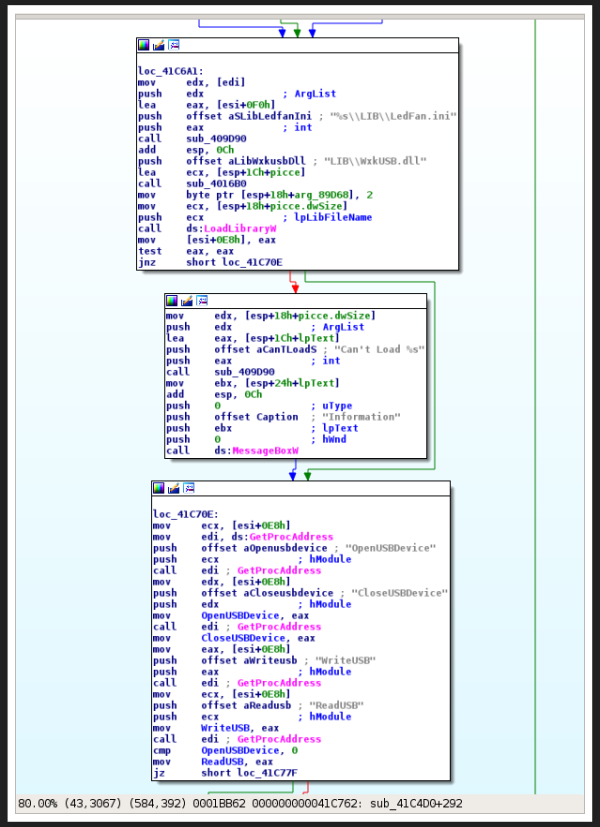
```
mov     edx, [esp+18h+picce.dwSize]
push    edx             ; ArgList
lea     eax, [esp+1Ch+lpText]
push    offset aCanTLoadS ; "Can't Load %s"
push    eax             ; int
call    sub_409D90
mov     ebx, [esp+24h+lpText]
add     esp, 0Ch
push    0               ; uType
push    offset Caption  ; "Information"
push    ebx             ; lpText
push    0               ; hWnd
call    ds:MessageBoxW
```

```
loc_41C70E:
mov     ecx, [esi+0E8h]
mov     edi, ds:GetProcAddress
push    offset aOpenusbdevice ; "OpenUSBDevice"
push    ecx             ; hModule
call    edi ; GetProcAddress
mov     edx, [esi+0E8h]
push    offset aCloseusbdevice ; "CloseUSBDevice"
push    edx             ; hModule
mov     dword_5AF3A0, eax
call    edi ; GetProcAddress
mov     dword_5AF39C, eax
mov     eax, [esi+0E8h]
push    offset aWriteusb ; "WriteUSB"
push    eax             ; hModule
call    edi ; GetProcAddress
mov     ecx, [esi+0E8h]
push    offset aReadusb ; "ReadUSB"
push    ecx             ; hModule
mov     dword_5AF398, eax
call    edi ; GetProcAddress
cmp     dword_5AF3A0, 0
mov     dword_5AF394, eax
jz      short loc_41C77F
```

```
80.00% (43,3067) (524,648) 0001BAD2 000000000041C6D2: sub_41C4D0+202
```

Just quickly clean this up...

Hold up, I know this game... 🧐



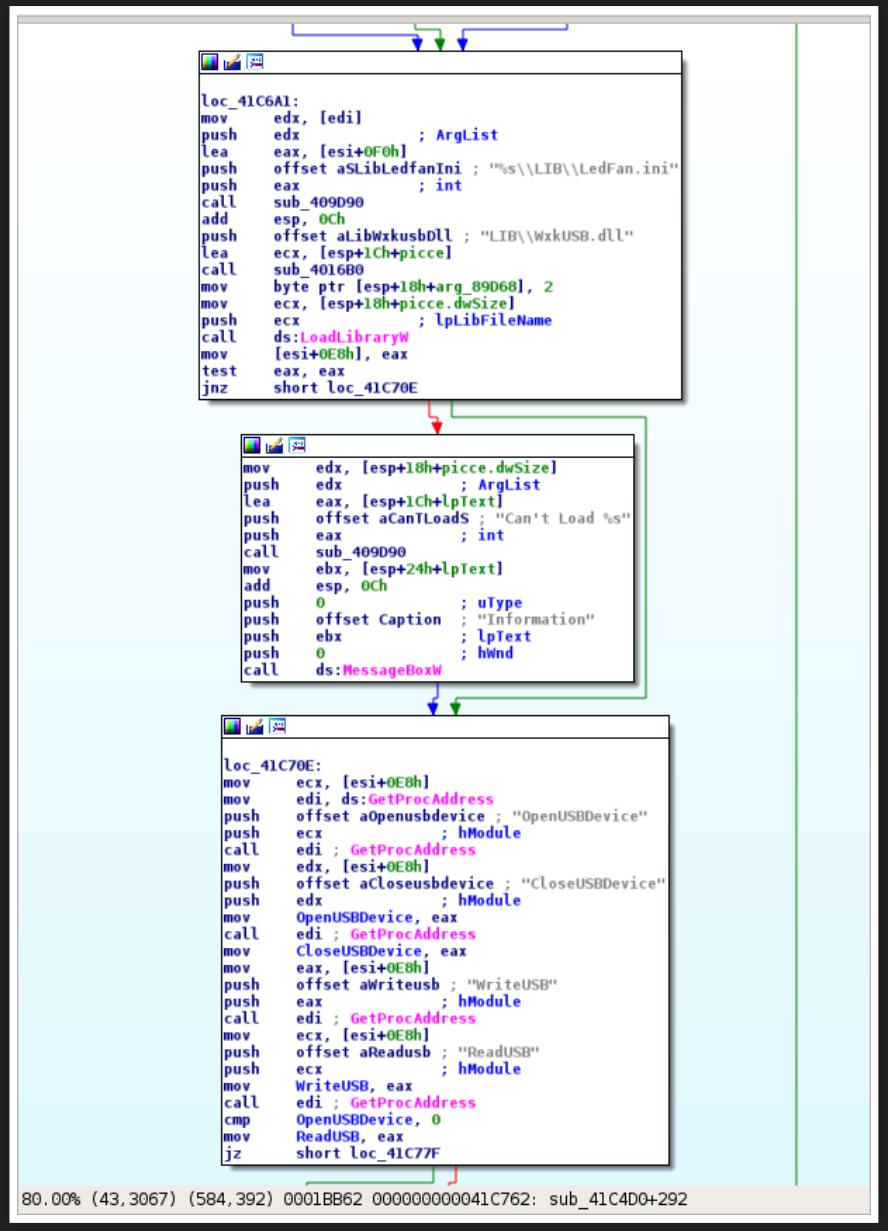
```
loc_41C6A1:
mov     edx, [edi]
push    edx             ; ArgList
lea     eax, [esi+0F0h]
push    offset aSLibLedfanIni ; "%s\\LIB\\LedFan.ini"
push    eax             ; int
call    sub_409D90
add     esp, 0Ch
push    offset aLibWxkusbDll ; "LIB\\WxkUSB.dll"
lea     ecx, [esp+1Ch+picce]
call    sub_401680
mov     byte ptr [esp+18h+arg_89D68], 2
mov     ecx, [esp+18h+picce.dwSize]
push    ecx             ; lpLibFileName
call    ds:LoadLibraryW
mov     [esi+0E8h], eax
test    eax, eax
jnz     short loc_41C70E
```

```
mov     edx, [esp+18h+picce.dwSize]
push    edx             ; ArgList
lea     eax, [esp+1Ch+lpText]
push    offset aCanTLoadS ; "Can't Load %s"
push    eax             ; int
call    sub_409D90
mov     ebx, [esp+24h+lpText]
add     esp, 0Ch
push    0               ; uType
push    offset Caption  ; "Information"
push    ebx             ; lpText
push    0               ; hWnd
call    ds:MessageBoxW
```

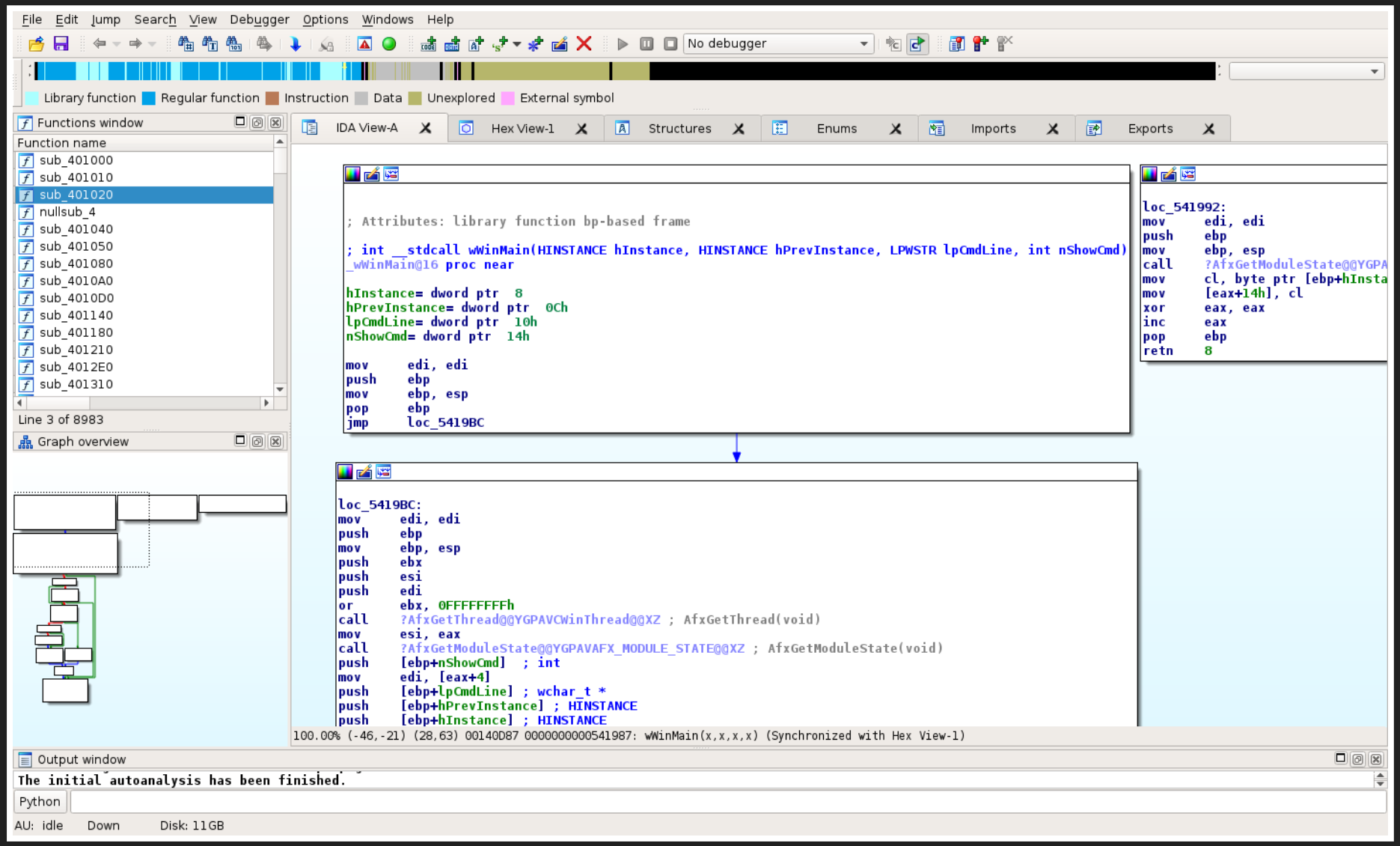
```
loc_41C70E:
mov     ecx, [esi+0E8h]
mov     edi, ds:GetProcAddress
push    offset aOpenusbdevice ; "OpenUSBDevice"
push    ecx             ; hModule
call    edi ; GetProcAddress
mov     edx, [esi+0E8h]
push    offset aCloseusbdevice ; "CloseUSBDevice"
push    edx             ; hModule
mov     OpenUSBDevice, eax
call    edi ; GetProcAddress
mov     CloseUSBDevice, eax
mov     eax, [esi+0E8h]
push    offset aWriteusb ; "WriteUSB"
push    eax             ; hModule
call    edi ; GetProcAddress
mov     ecx, [esi+0E8h]
push    offset aReadusb ; "ReadUSB"
push    ecx             ; hModule
mov     WriteUSB, eax
call    edi ; GetProcAddress
cmp     OpenUSBDevice, 0
mov     ReadUSB, eax
jz      short loc_41C77F
```

```
80.00% (43,3067) (584,392) 0001BB62 000000000041C762: sub_41C4D0+292
```

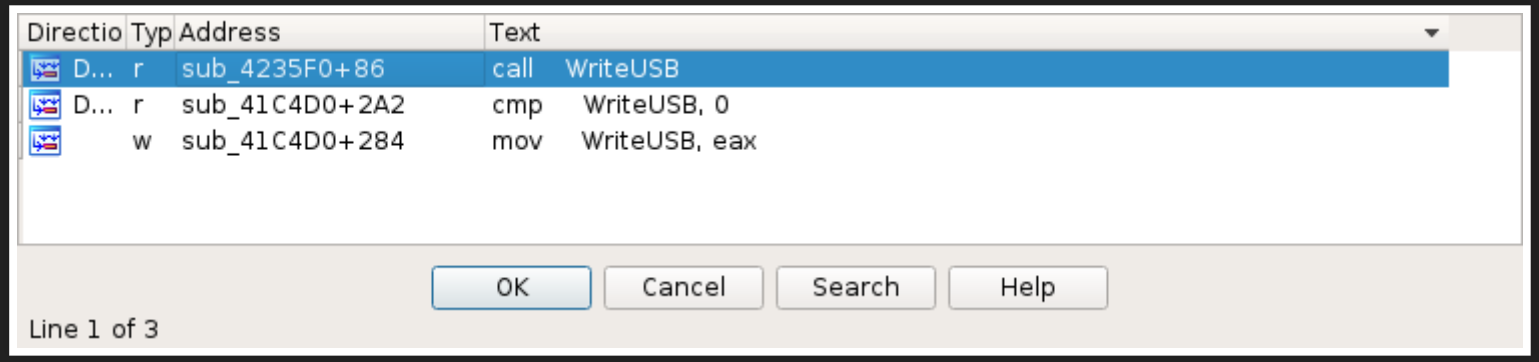
Just quickly clean this up...


loc_41C6A1:
mov edx, [edi]
push edx ; ArgList
lea eax, [esi+0F0h]
push offset aSLibLedfanIni ; "%s\\LIB\\LedFan.ini"
push eax ; int
call sub_409D90
add esp, 0Ch
push offset aLibWxkusbDll ; "LIB\\WxkUSB.dll"
lea ecx, [esp+1Ch+picce]
call sub_4016B0
mov byte ptr [esp+18h+arg_89D68], 2
mov ecx, [esp+18h+picce.dwSize]
push ecx ; lpLibFileName
call ds:LoadLibraryW
mov [esi+0E8h], eax
test eax, eax
jnz short loc_41C70E
mov edx, [esp+18h+picce.dwSize]
push edx ; ArgList
lea eax, [esp+1Ch+lpText]
push offset aCanTLoadS ; "Can't Load %s"
push eax ; int
call sub_409D90
mov ebx, [esp+24h+lpText]
add esp, 0Ch
push 0 ; uType
push offset Caption ; "Information"
push ebx ; lpText
push 0 ; hWnd
call ds:MessageBoxW
loc_41C70E:
mov ecx, [esi+0E8h]
mov edi, ds:GetProcAddress
push offset aOpenusbdevice ; "OpenUSBDevice"
push ecx ; hModule
call edi ; GetProcAddress
mov edx, [esi+0E8h]
push offset aCloseusbdevice ; "CloseUSBDevice"
push edx ; hModule
mov OpenUSBDevice, eax
call edi ; GetProcAddress
mov CloseUSBDevice, eax
mov eax, [esi+0E8h]
push offset aWriteusb ; "WriteUSB"
push eax ; hModule
call edi ; GetProcAddress
mov ecx, [esi+0E8h]
push offset aReadusb ; "ReadUSB"
push ecx ; hModule
mov WriteUSB, eax
call edi ; GetProcAddress
cmp OpenUSBDevice, 0
mov ReadUSB, eax
jz short loc_41C77F
80.00% (43,3067) (584,392) 0001BB62 000000000041C762: sub_41C4D0+292

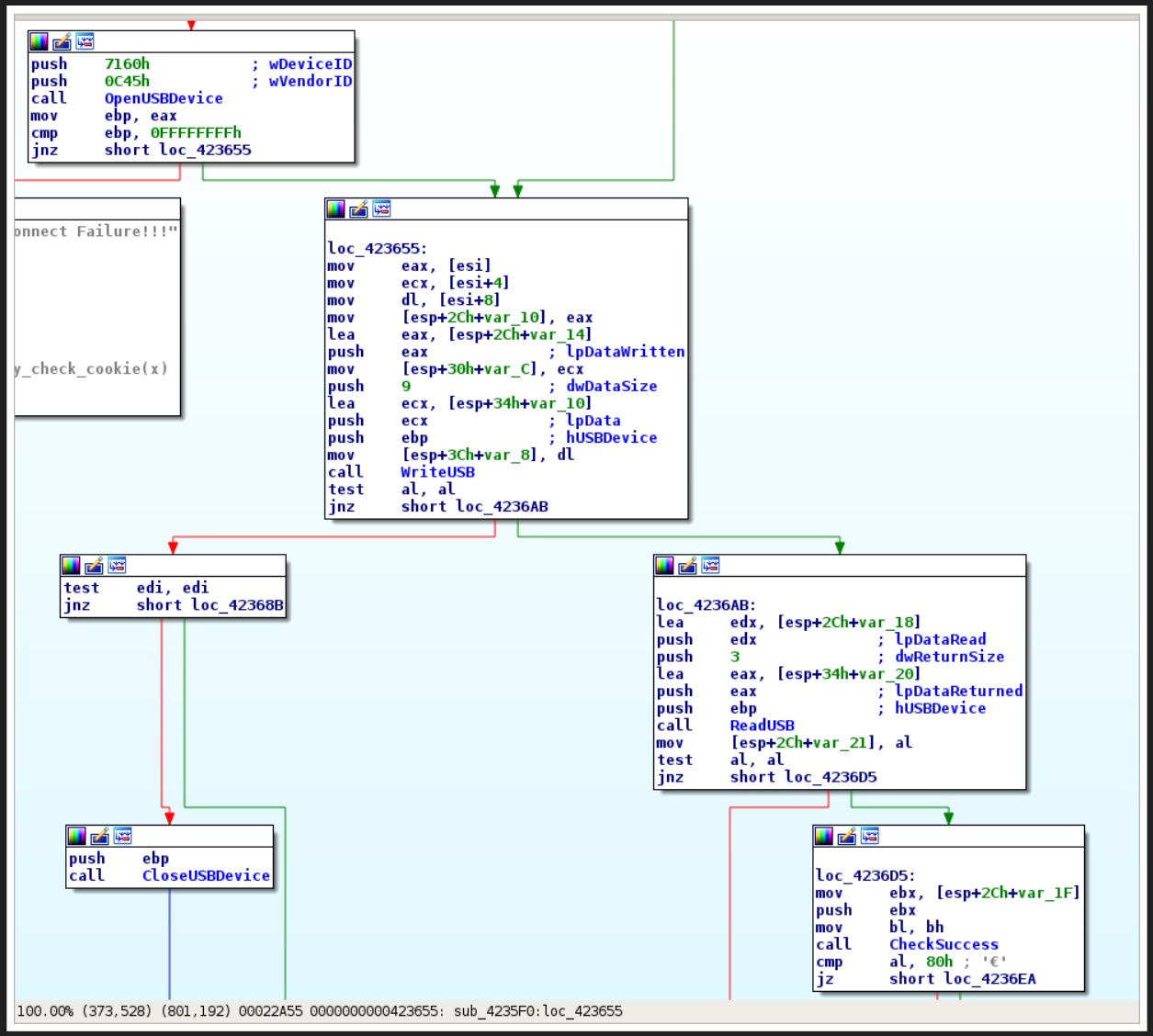
LedFan.exe: 2.4MB of GUI fun!



Straight to the interesting bit! 🤠

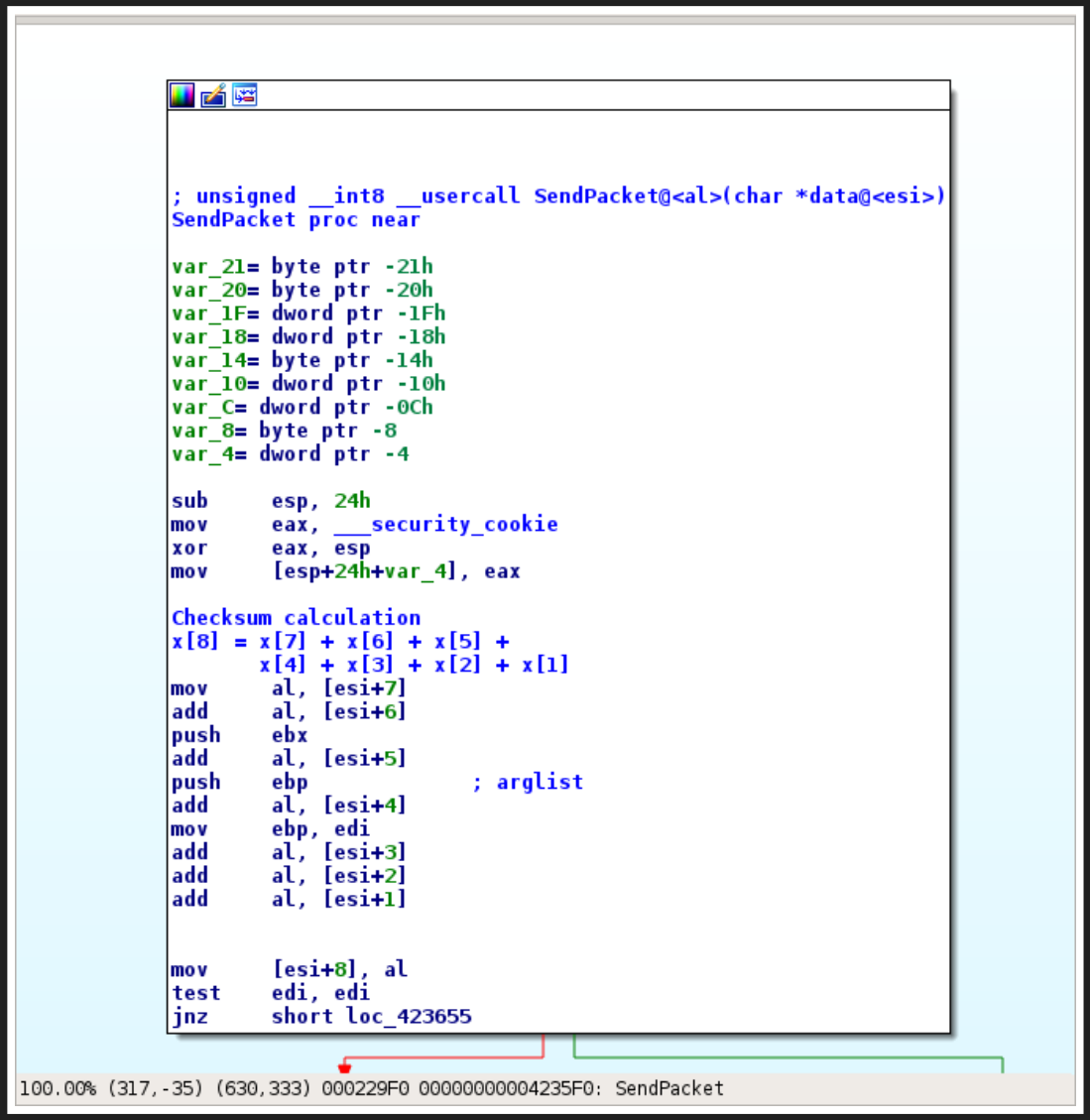


One caller looks promising


push 7160h ; wDeviceID
push 0C45h ; wVendorID
call OpenUSBDevice
mov ebp, eax
cmp ebp, 0FFFFFFFFh
jnz short loc_423655
onnect Failure!!!"
y_check_cookie(x)
loc_423655:
mov eax, [esi]
mov ecx, [esi+4]
mov dl, [esi+8]
mov [esp+2Ch+var_10], eax
lea eax, [esp+2Ch+var_14]
push eax ; lpDataWritten
mov [esp+30h+var_C], ecx
push 9 ; dwDataSize
lea ecx, [esp+34h+var_10]
push ecx ; lpData
push ebp ; hUSBDevice
mov [esp+3Ch+var_8], dl
call WriteUSB
test al, al
jnz short loc_4236AB
test edi, edi
jnz short loc_42368B
loc_4236AB:
lea edx, [esp+2Ch+var_18]
push edx ; lpDataRead
push 3 ; dwReturnSize
lea eax, [esp+34h+var_20]
push eax ; lpDataReturned
push ebp ; hUSBDevice
call ReadUSB
mov [esp+2Ch+var_21], al
test al, al
jnz short loc_4236D5
push ebp
call CloseUSBDevice
loc_4236D5:
mov ebx, [esp+2Ch+var_1F]
push ebx
mov bl, bh
call CheckSuccess
cmp al, 80h ; '€'
jz short loc_4236EA
100.00% (373,528) (801,192) 00022A55 0000000000423655: sub_4235F0:loc_423655

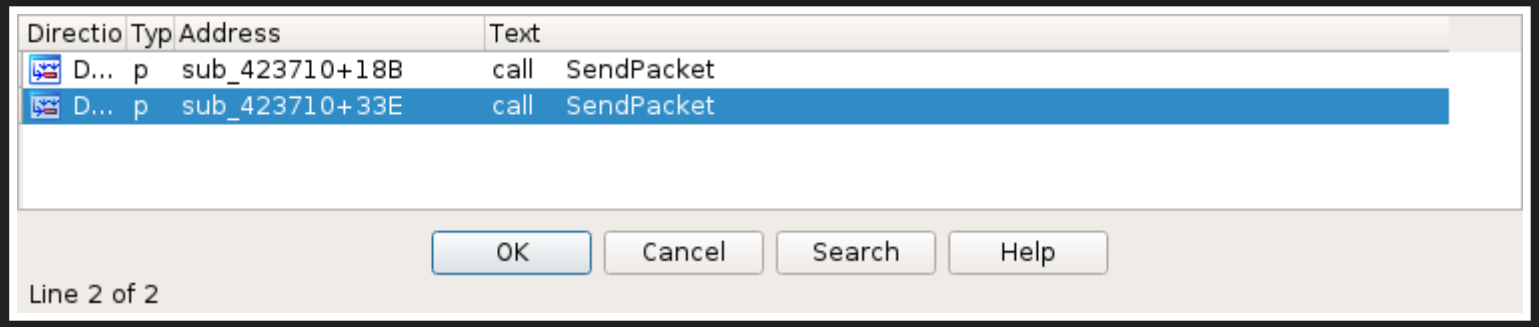
Bingo 😀

Starting to learn about the protocol

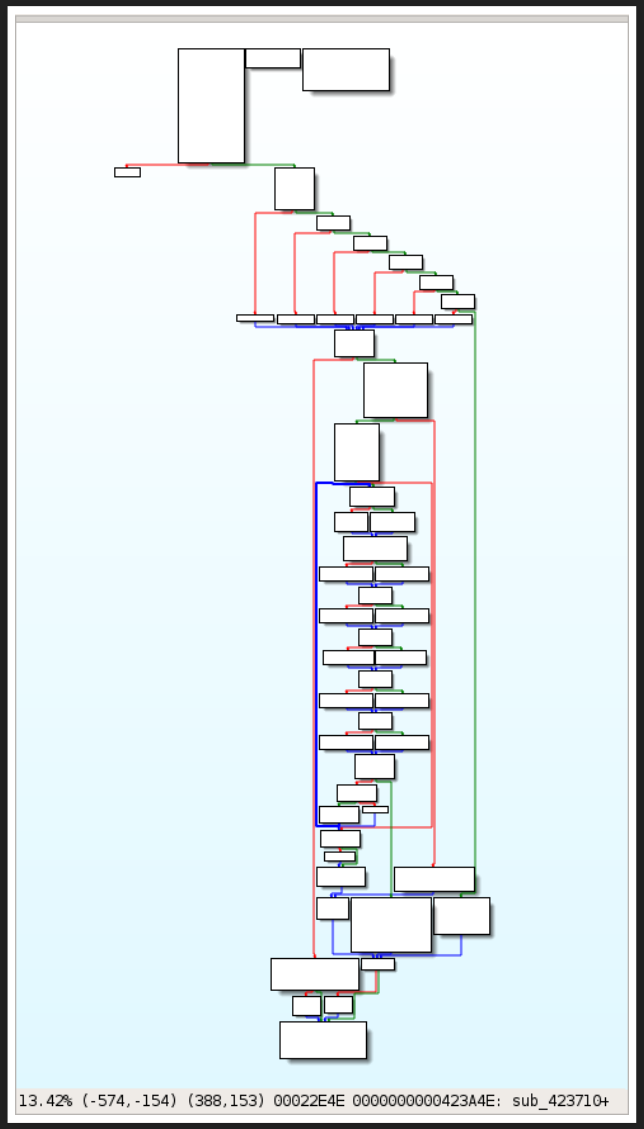


```
; unsigned __int8 __usercall SendPacket@<al>(char *data@<esi>)
SendPacket proc near

var_21= byte ptr -21h
var_20= byte ptr -20h
var_1F= dword ptr -1Fh
var_18= dword ptr -18h
var_14= byte ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= byte ptr -8
var_4= dword ptr -4

sub     esp, 24h
mov     eax, ___security_cookie
xor     eax, esp
mov     [esp+24h+var_4], eax

Checksum calculation
x[8] = x[7] + x[6] + x[5] +
       x[4] + x[3] + x[2] + x[1]
mov     al, [esi+7]
add     al, [esi+6]
push    ebx
add     al, [esi+5]
push    ebp             ; arglist
add     al, [esi+4]
mov     ebp, edi
add     al, [esi+3]
add     al, [esi+2]
add     al, [esi+1]


mov     [esi+8], al
test    edi, edi
jnz     short loc_423655
```

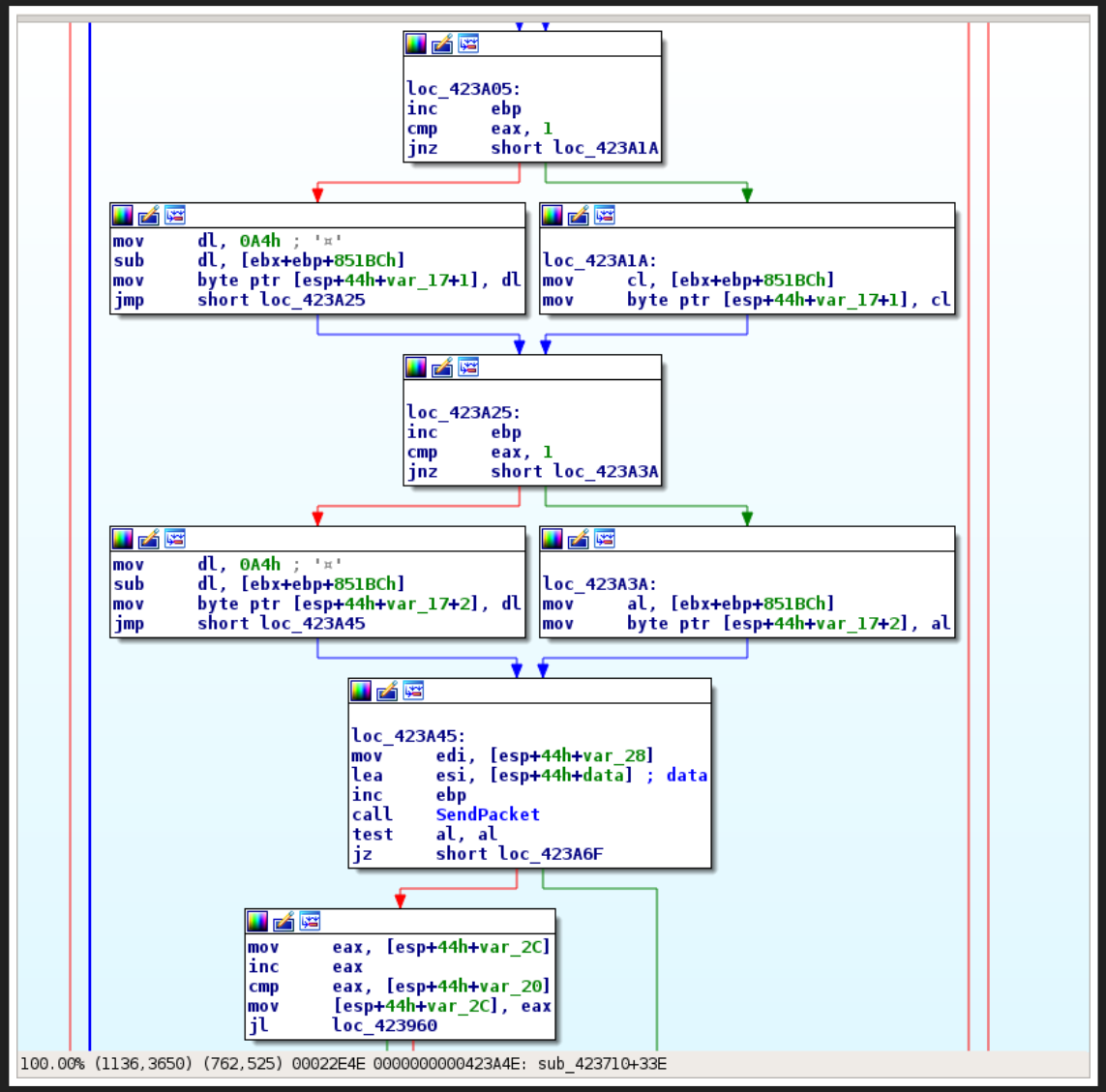
Only one function that calls into it 😄



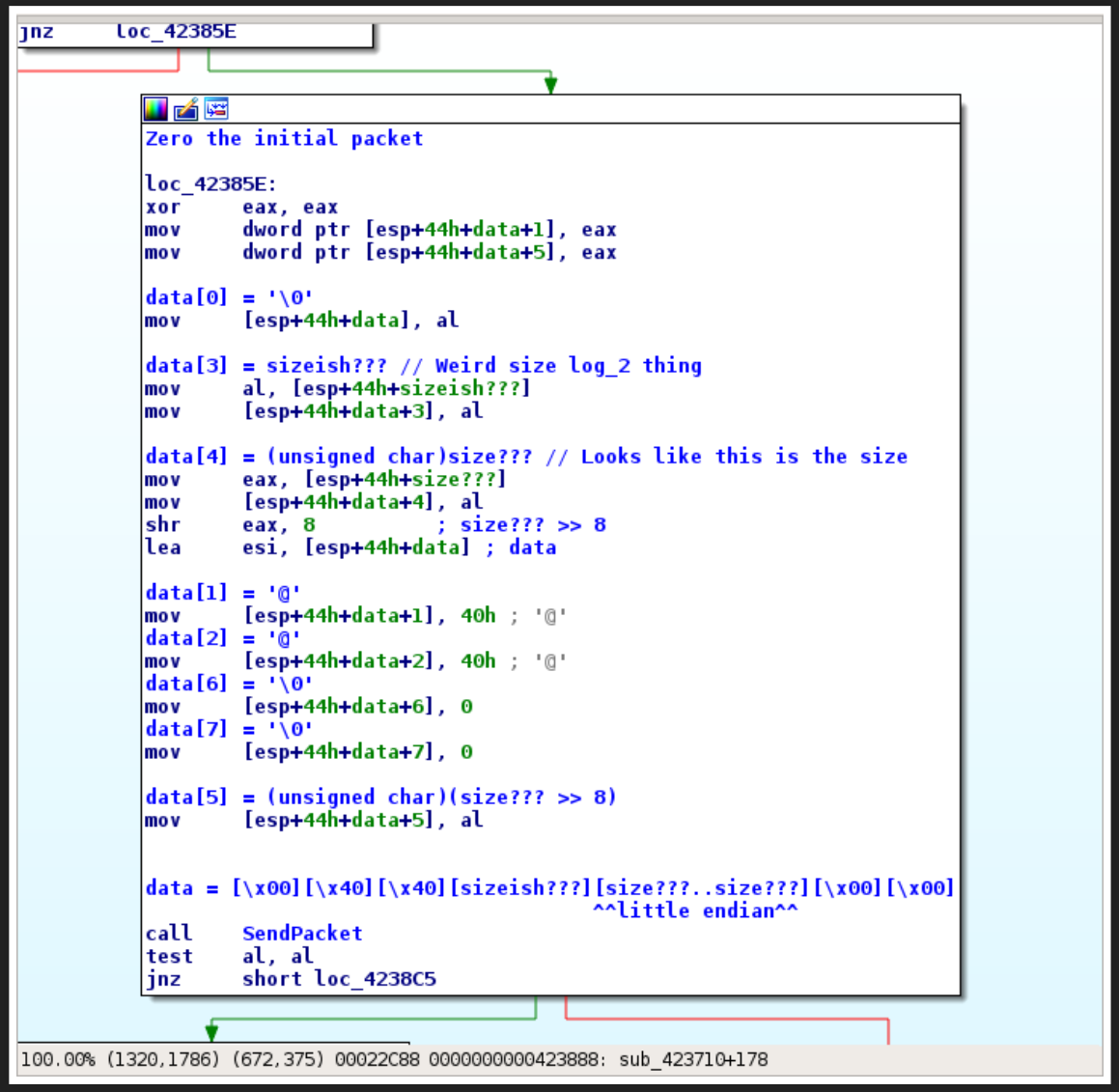
Big, but still learning!

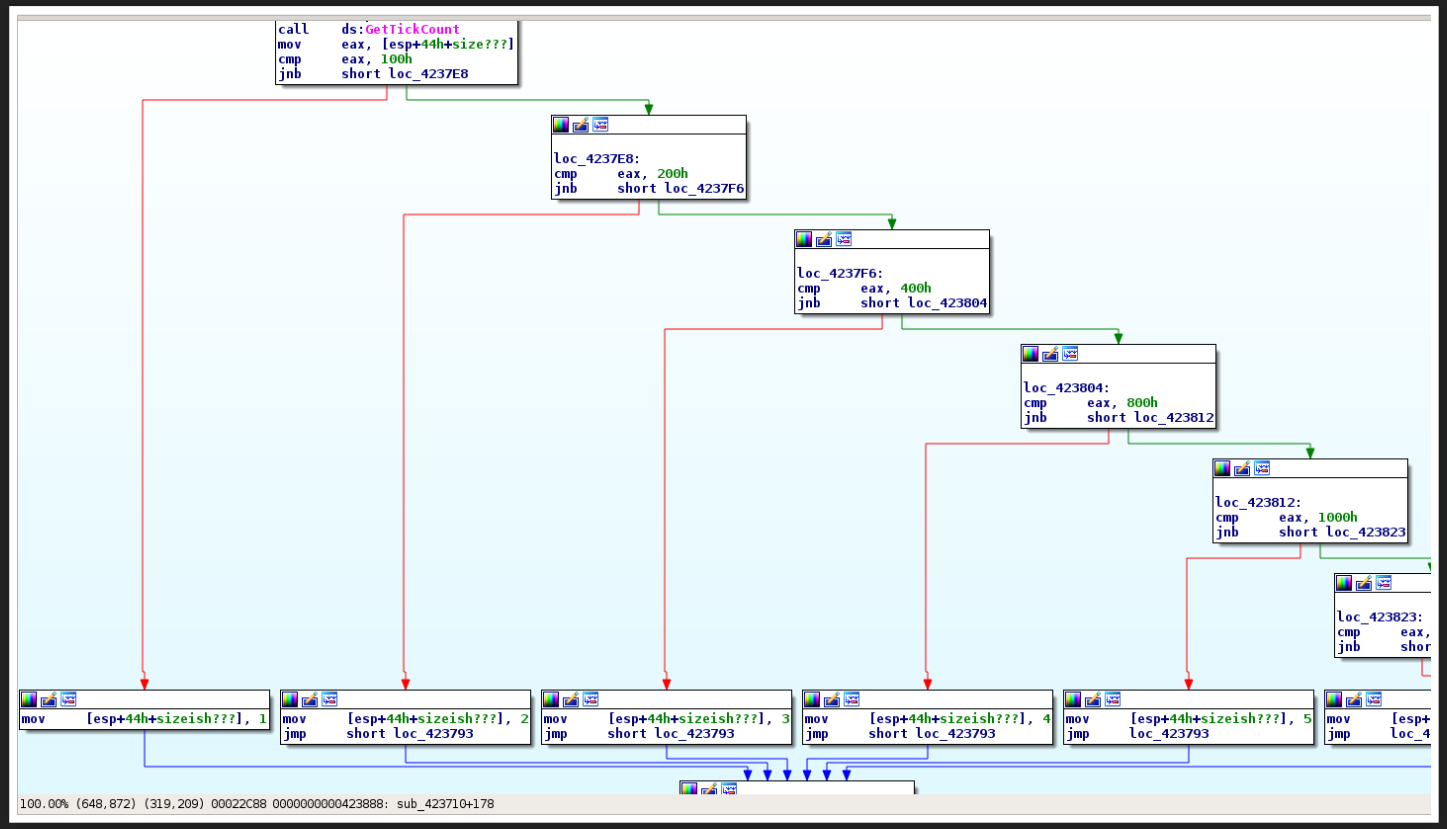


Protocol seems to have an optional subtraction?



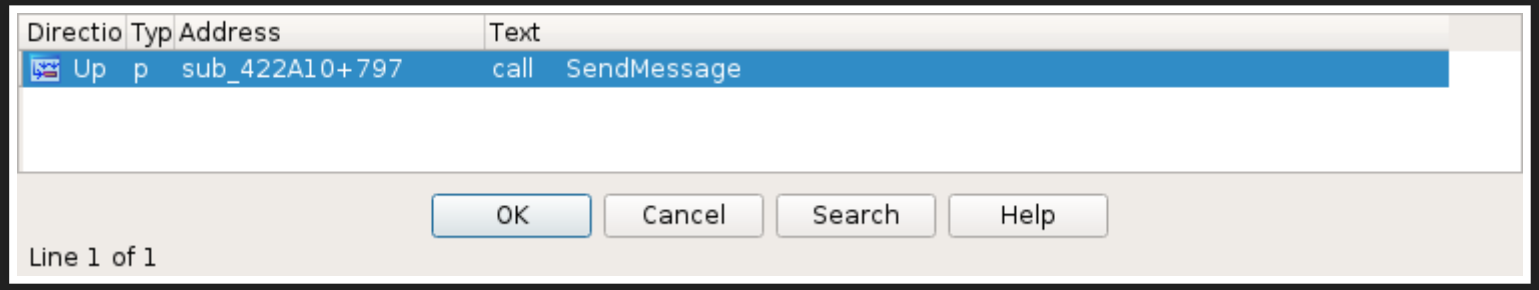
We also have the initial packet



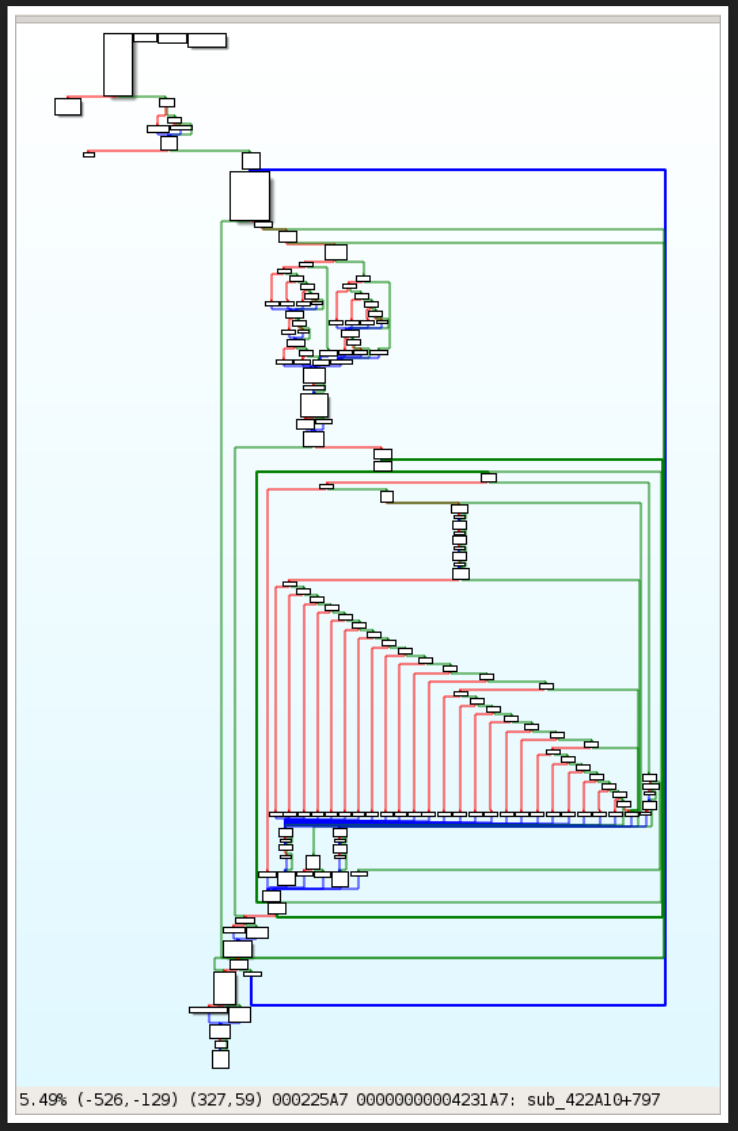


$$sizeish = \log_2\left(\frac{size}{64}\right)$$
$$= \log_2(size) - 6$$

Also only one caller!



But it's bigger, and uglier



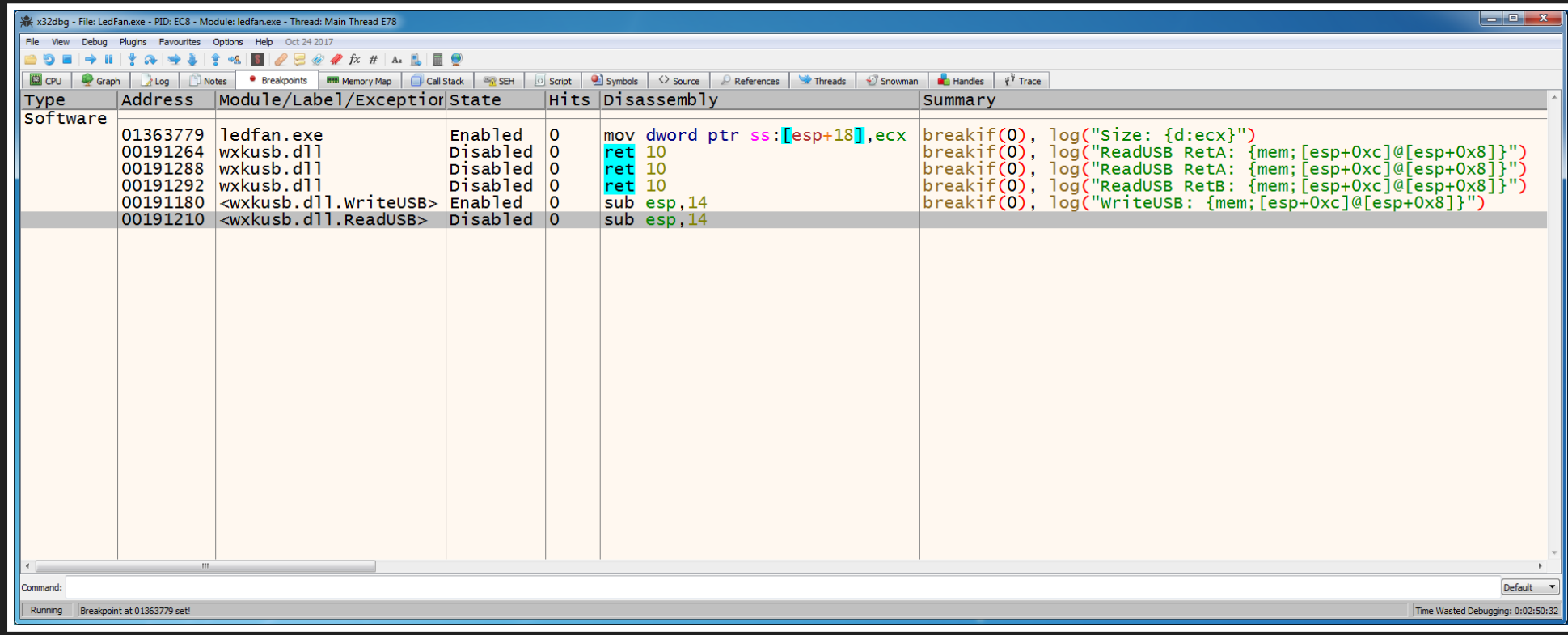
STEP BACK

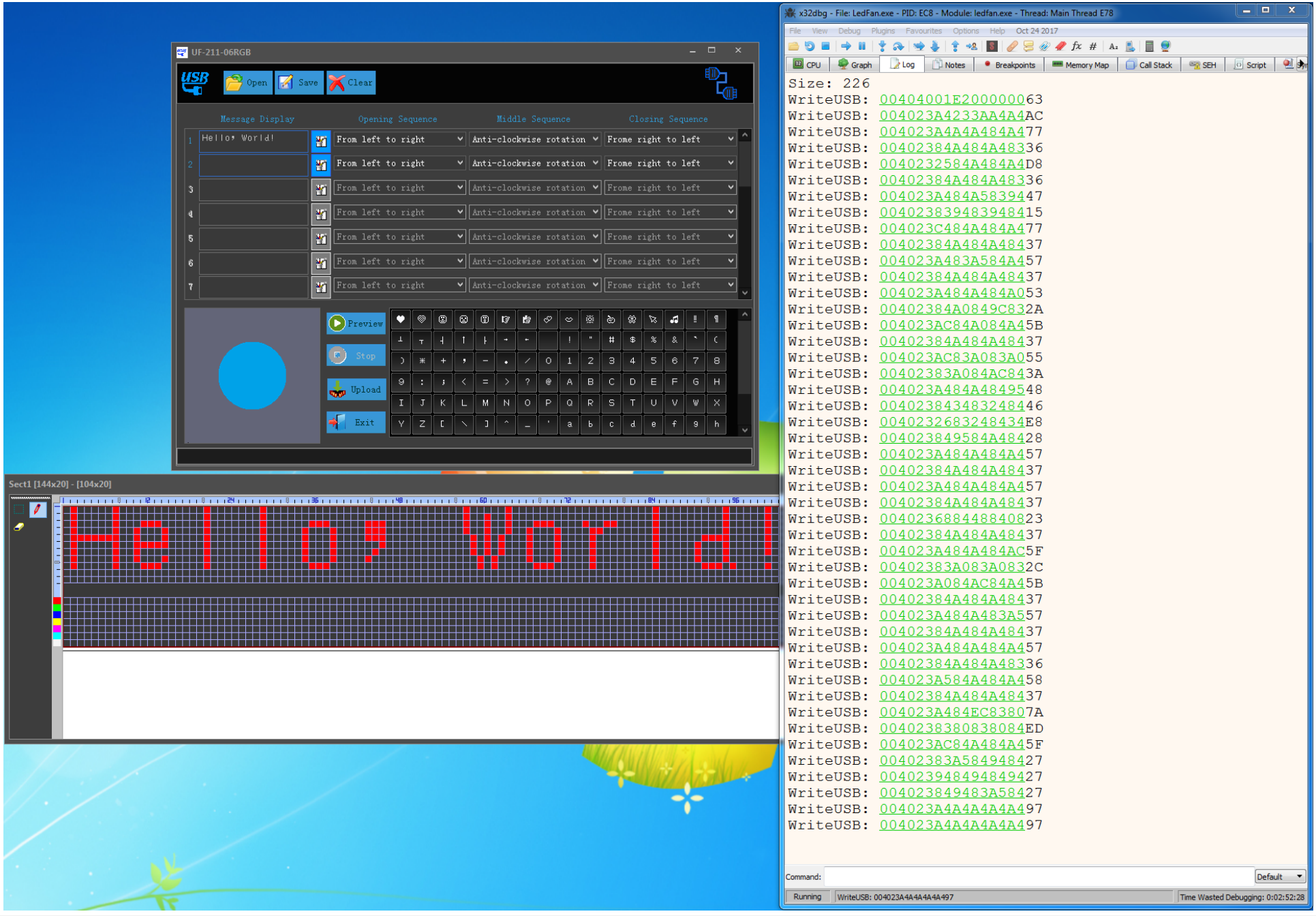
- Where are we?
  - A `virtual` function
  - A switch statement nested in two loops
- How is the function called?
  - Can't find any obvious call into its vtable offset
- How are the loops and switches controlled?
  - Members of the `this` object
  - Window messages

Time for another plan 😉

Remember that dynamic analysis?

Lets validate some assumptions and dump the messages!




UF-211-06RGB
Open
Save
Clear
Message Display
Opening Sequence
Middle Sequence
Closing Sequence
Hello, World!
From left to right
Anti-clockwise rotation
Frome right to left
Preview
Stop
Upload
Exit
Sect1 [144x20] - [104x20]
x32dbg - File: LedFan.exe - PID: EC8 - Module: ledfan.exe - Thread: Main Thread E78
File View Debug Plugins Favourites Options Help Oct 24 2017
CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script
Size: 226
WriteUSB: 00404001E2000000 63
WriteUSB: 004023A4233AA4A4 AC
WriteUSB: 004023A4A4A484A4 77
WriteUSB: 00402384A484A483 36
WriteUSB: 0040232584A484A4 D8
WriteUSB: 00402384A484A483 36
WriteUSB: 004023A484A58394 7
WriteUSB: 0040238394839484 15
WriteUSB: 004023C484A484A4 77
WriteUSB: 00402384A484A484 37
WriteUSB: 004023A483A584A4 57
WriteUSB: 00402384A484A484 37
WriteUSB: 004023A484A484A0 53
WriteUSB: 00402384A0849C83 2A
WriteUSB: 004023AC84A084A4 5B
WriteUSB: 00402384A484A484 37
WriteUSB: 004023AC83A083A0 55
WriteUSB: 00402383A084AC84 3A
WriteUSB: 004023A484A48495 48
WriteUSB: 0040238434832484 46
WriteUSB: 004023268324843 4E8
WriteUSB: 0040238495 84A484 28
WriteUSB: 004023A484A484A4 57
WriteUSB: 00402384A484A484 37
WriteUSB: 004023A484A484A4 57
WriteUSB: 00402384A484A484 37
WriteUSB: 0040236884488408 23
WriteUSB: 00402384A484A484 37
WriteUSB: 004023A484A484AC 5F
WriteUSB: 00402383A083A083 2C
WriteUSB: 004023A084AC84A4 5B
WriteUSB: 00402384A484A484 37
WriteUSB: 004023A484A483A5 57
WriteUSB: 00402384A484A484 37
WriteUSB: 004023A484A484A4 57
WriteUSB: 00402384A484A483 36
WriteUSB: 004023A584A484A4 58
WriteUSB: 00402384A484A484 37
WriteUSB: 004023A484EC8380 7A
WriteUSB: 0040238380838084 ED
WriteUSB: 004023AC84A484A4 5F
WriteUSB: 00402383A5849484 27
WriteUSB: 0040239484948494 27
WriteUSB: 0040238494 83A584 27
WriteUSB: 004023A4A4A4A4A4 97
WriteUSB: 004023A4A4A4A4A4 97
Command:
Default
Running
WriteUSB: 004023A4A4A4A4A4 97
Time Wasted Debugging: 0:02:52:28

Single Red Dot

```
0040400122000000A3
004023A4239AA4A40C
004023A4A4A484A477
00402384A484A48437
004023A484A484A457
00402384A484A3A456
004023A4A4A4A4A497
```

Initial Packet

```
004040012200000A3
004023A4239AA4A40C
004023A4A4A484A477
00402384A484A48437
004023A484A484A457
00402384A484A3A456
004023A4A4A4A4A497
```

Checksum

```
0040400122000000A3
004023A4239AA4A40C
004023A4A4A484A477
00402384A484A48437
004023A484A484A457
00402384A484A3A456
004023A4A4A4A4A497
```

$$40 + 40 + 01 + 22 + 00 + 00 + 00 = A3$$ 👍

Packet Start

```
0040400122000000A3
004023A4239AA4A40C
004023A4A4A484A477
00402384A484A48437
004023A484A484A457
00402384A484A3A456
004023A4A4A4A4A497
```

Remove the subtraction

```
00810A0000
0000002000
2000200020
0020002000
2000200100
0000000000
```

Eight columns

```
00810A0000
0000002000
2000200020
0020002000
2000200100
0000000000
```

Single pixel on

```
00810A0000
0000002000
2000200020
0020002000
2000200100
0000000000
```

Two Red Dots

```
0040400122000000A3
004023A4239AA4A40C
004023A4A4A484A477
00402384A484A48437
004023A484A484A457
00402384A384A3A455
004023A4A4A4A4A497
```

Remove the subtraction

```
00810A0000
0000002000
2000200020
0020002000
2001200100
0000000000
```

Two pixels on

```
00810A0000
0000002000
2000200020
0020002000
2001200100
0000000000
```

Full Vertical Red

```
0040400122000000A3
004023A4239AA4A40C
004023A4A4A484A477
00402384A484A48437
004023A484A484A457
00402384A47DA5A451
004023A4A4A4A4A497
```

Eleven pixels on

```
00810A0000
0000002000
2000200020
0020002000
200027FF00
0000000000
```

Full Vertical Blue

```
0040400122000000A3
004023A4239AA4A40C
004023A4A4A484A477
00402384A484A48437
004023A484A484A457
00402384A45DA5A431
004023A4A4A4A4A497
```

Eleven pixels on: blue

```
00810A0000
0000002000
2000200020
0020002000
200047FF00
0000000000
```

Full Vertical Green

```
0040400122000000A3
004023A4239AA4A40C
004023A4A4A484A477
00402384A484A48437
004023A484A484A457
00402384A41DA5A4F1
004023A4A4A4A4A497
```

Eleven pixels on: green

```
00810A0000
0000002000
2000200020
0020002000
200087FF00
0000000000
```

But wait, the fan can do more!

- Multiple "pages" of messages
- As well as transition effects

But that's enough of the protocol analysis 😎

PROTOCOL REIMPLEMENTATION

With these results, reimplementation is very easy

- Python 3.6: `IntEnum`, f-strings, underscores in numeric literals
- `hidapi` package providing the abstraction for the "driverless" APIs

 https://git.io/fxawk

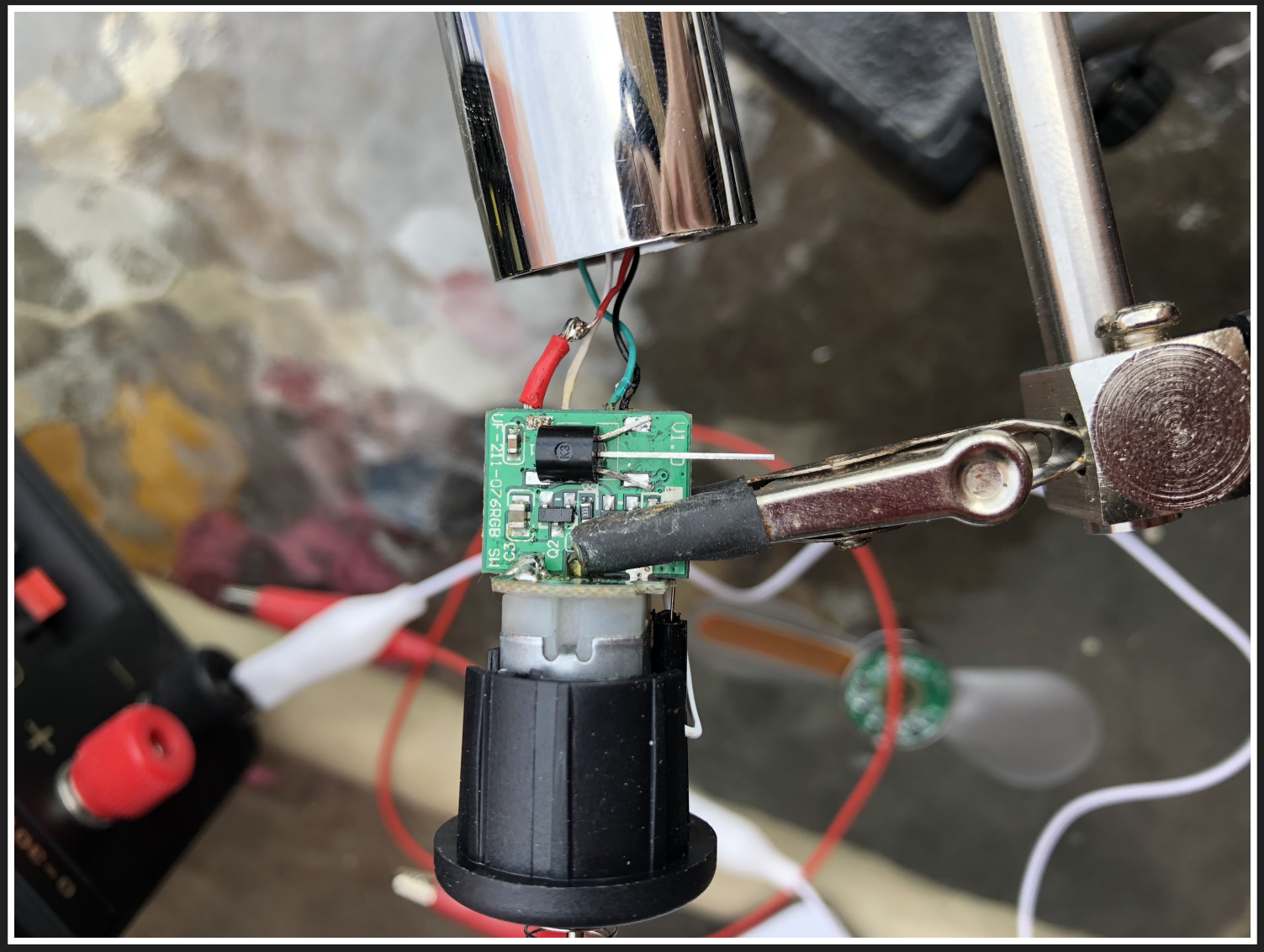
DRAWBACKS

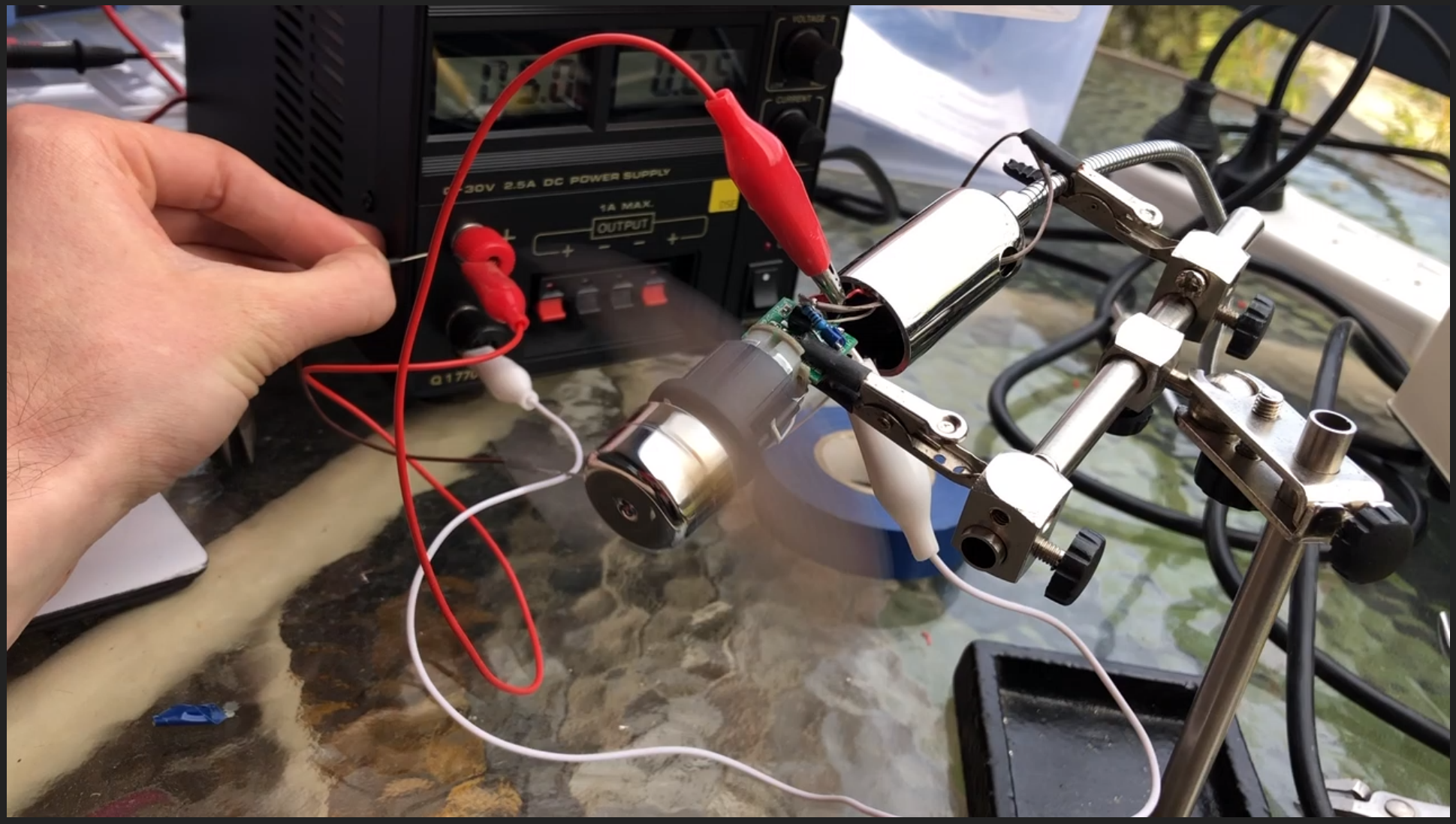
My main aim for the fan was to have notifications for home events

The fan's programming can only happen when it isn't spinning

There's no programmatic way of turning the motor off

Swap the button for a BJT 🤓




30V 2.5A DC POWER SUPPLY
1A MAX.
OUTPUT

But it shorted when I put the case back on 😥

I have a new fan now, it's looking much better 🙂

QUESTIONS