

UNDERSTANDING MICRO WAVE

Fergus Symon

Not this...





This



WHY?

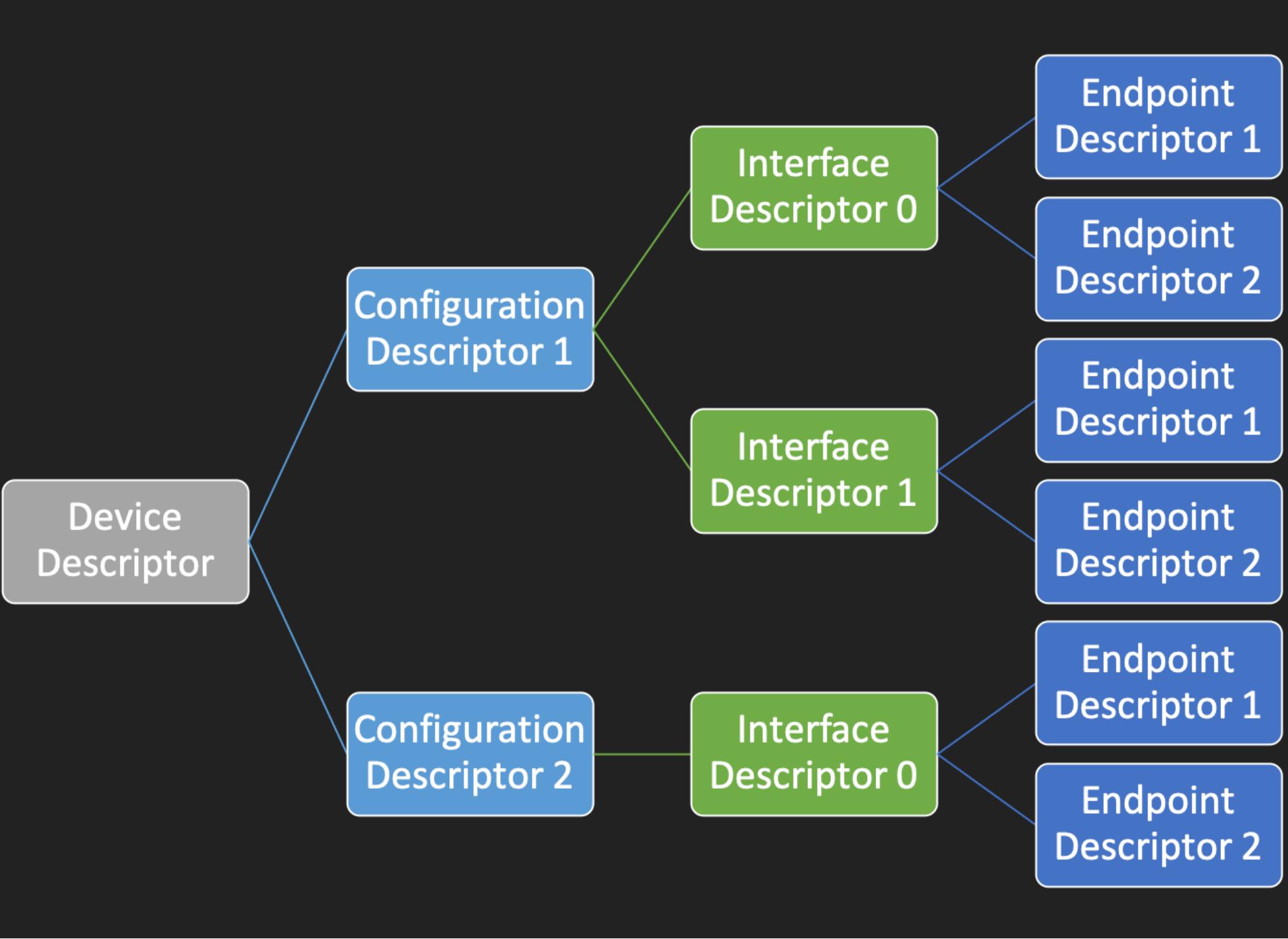
I wanted a system for home notifications

- 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



- 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

capb.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
97	2.355509	1.4.0	host	USB	36	URB_CONTROL out
98	2.355509	1.4.0	host	USB	28	SET CONFIGURATION Status
99	2.402307	1.4.1	host	USB	29	URB_INTERRUPT in
100	2.402307	host	1.4.0	USB	36	URB_CONTROL out
101	2.402307	1.4.0	host	USB	36	URB_CONTROL out
102	2.402307	1.4.0	host	USB	28	SET CONFIGURATION Status
103	2.449105	1.4.1	host	USB	29	URB_INTERRUPT in
104	2.449105	host	1.4.0	USB	36	URB_CONTROL out
105	2.449105	1.4.0	host	USB	36	URB CONTROL out

Device address: 4

Endpoint: 0x00, Direction: OUT

URB transfer type: URB_CONTROL (0x02)

Packet Data Length: 8

[Request in: 100]

[Time from request: 0.000000000 seconds]

Control transfer stage: Data (1)

[bInterfaceClass: Unknown (0xffff)]

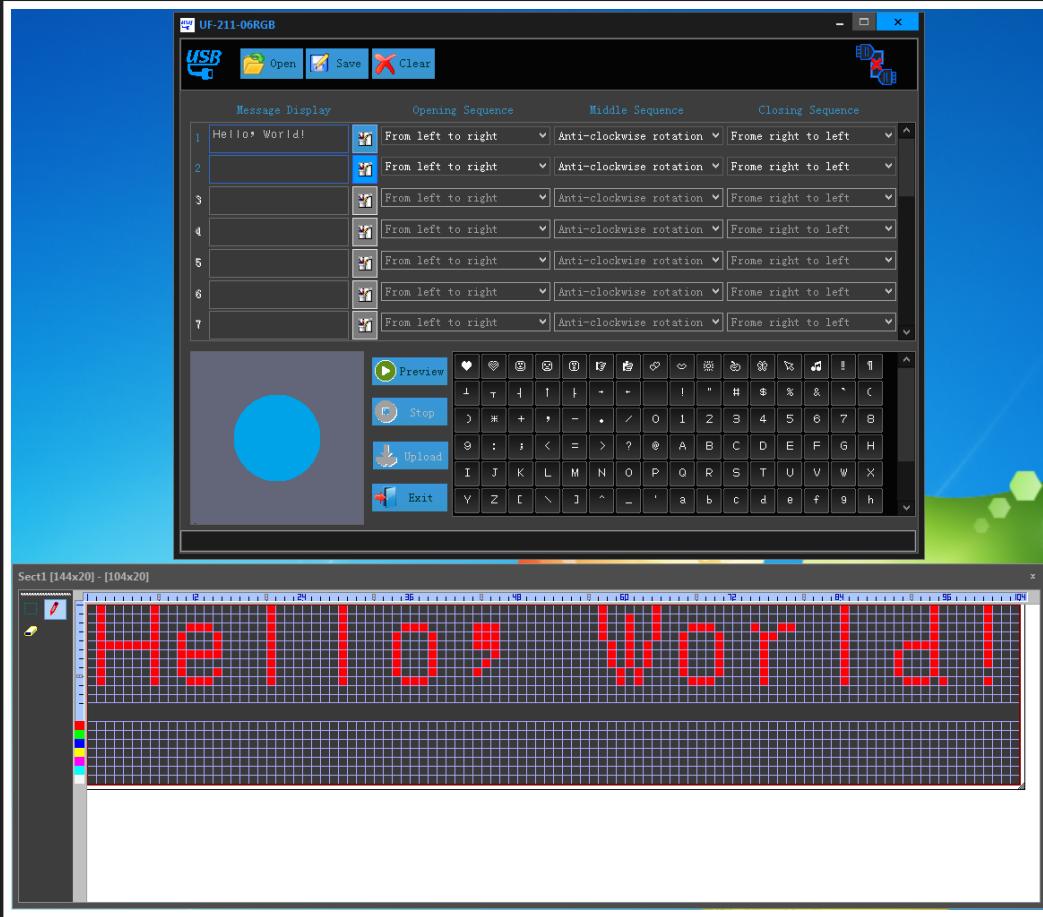
CONTROL response data: 402384a484958428

0000	1c 00 c8 29 76 86 ff ff ff ff 00 00 00 00 08 00	...)v....
0010	01 01 00 04 00 00 02 08 00 00 00 01 40 23 84 a4 @#..
0020	84 95 84 28	...()

CONTROL response data (usb.control.Response), 8 bytes

Packets: 211 · Displayed: 211 (100.0%) · Load time: 0:0.0

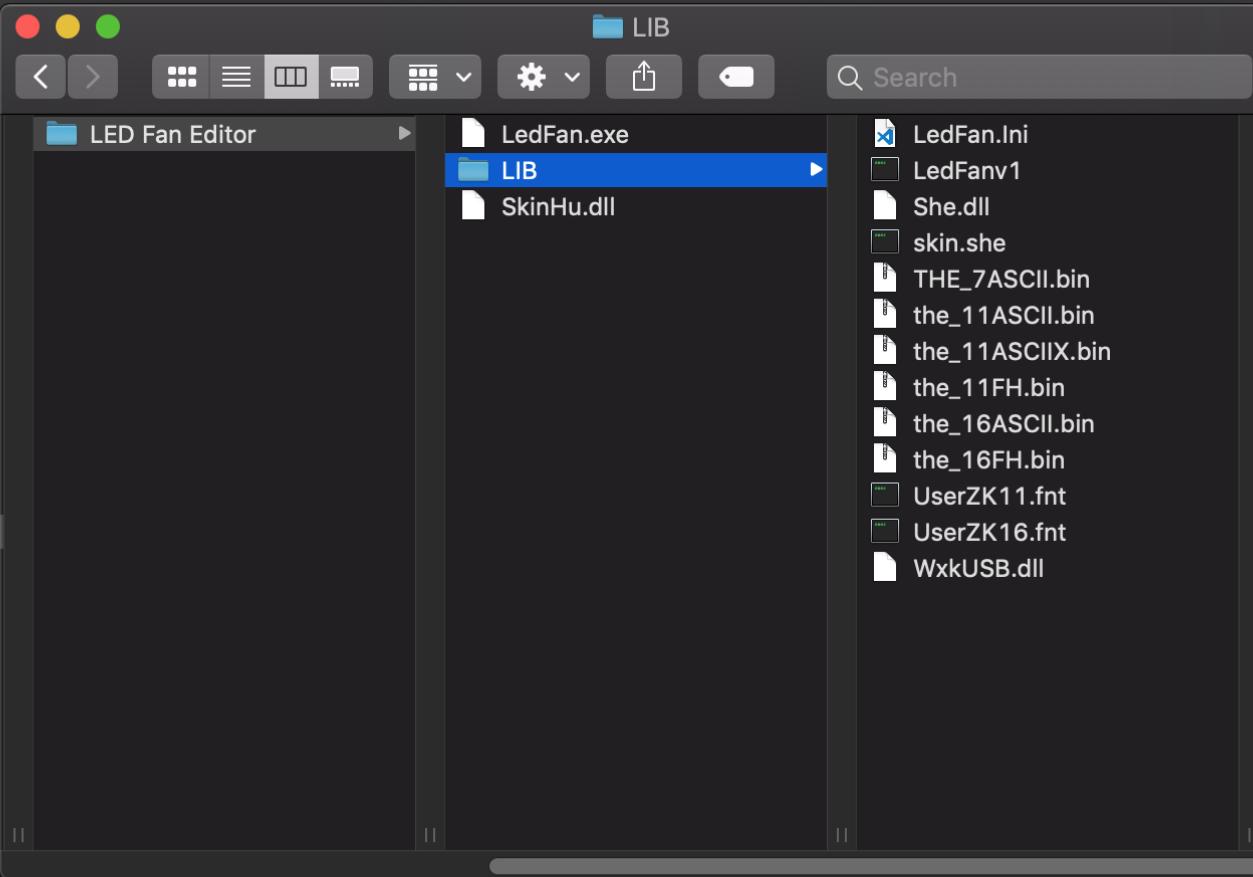
Profile: Default



Dynamic analysis alone is useless



What else can we find?



WxkUSB.dll

File Edit Jump Search View Debugger Options Windows Help

Library function Regular function Instruction Data Unexplored External symbol

Functions window 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]

Graph overview

Line 1 of 11

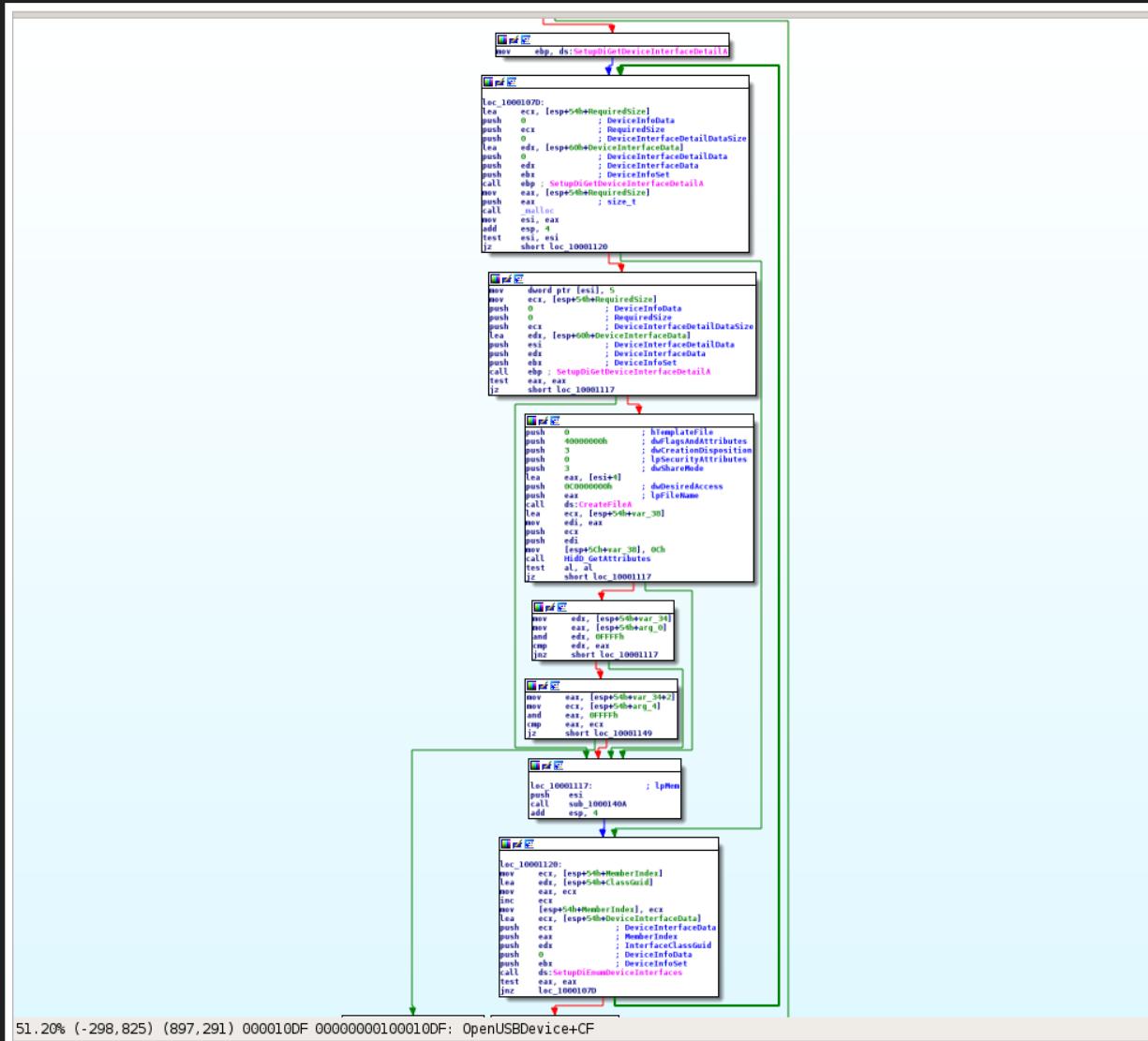
Output window

The initial autoanalysis has been finished.

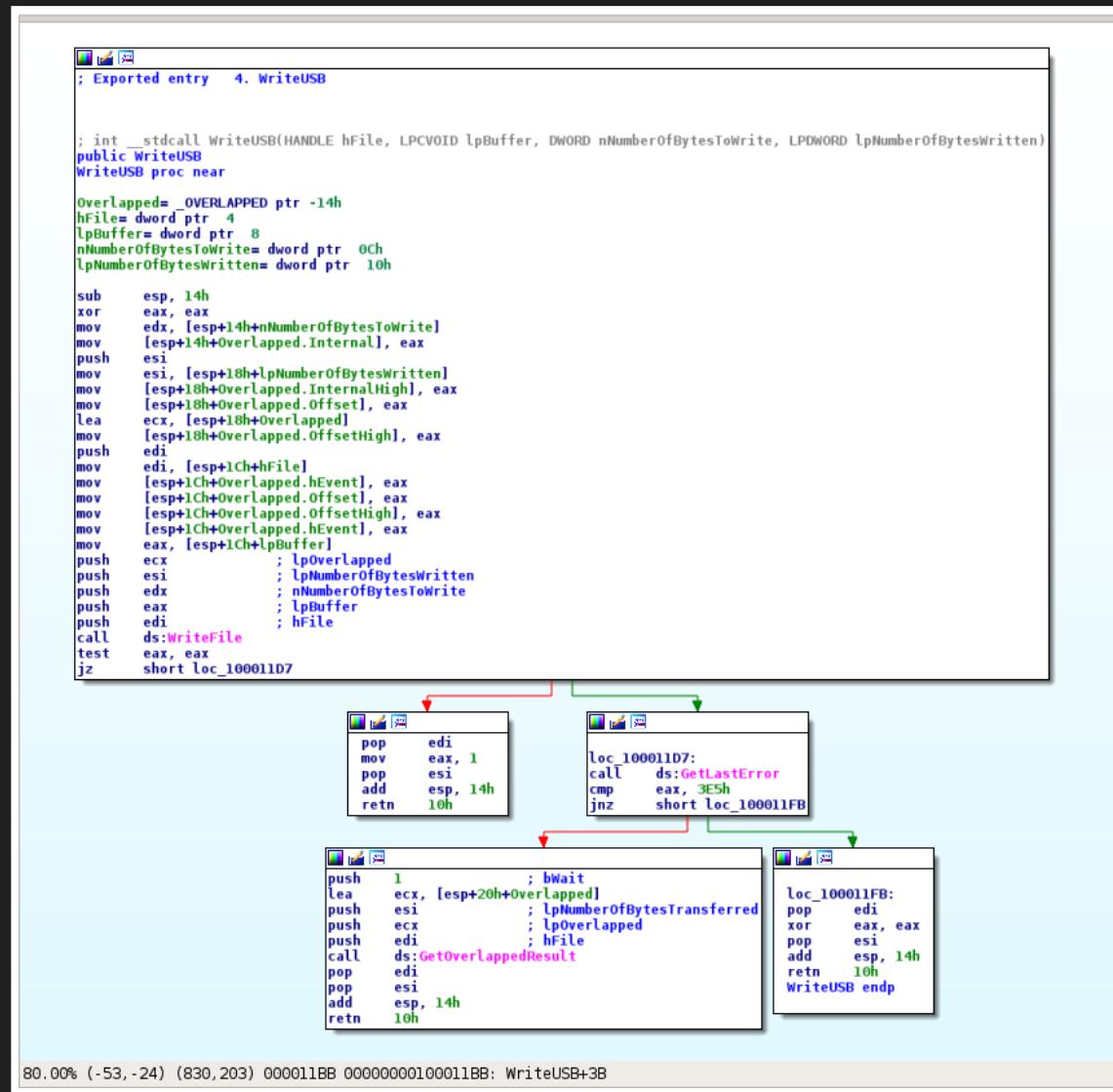
Python

AU: idle Down Disk: 11GB

OpenUSBDevice(WORD, WORD)



WriteUSB(HANDLE, LPCVOID, DWORD, LPDWORD)



ReadUSB(HANDLE, LPCVOID, DWORD, LPDWORD)

The diagram illustrates the assembly code for the `ReadUSB` function and its associated error handling and result retrieval logic.

Exported entry 3. ReadUSB:

```
; int __stdcall ReadUSB(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead)
public ReadUSB
ReadUSB proc near
```

The main function body starts by setting up parameters for `ReadFile`:

```
Overlapped= _OVERLAPPED ptr -14h
hFile= dword ptr 4
lpBuffer= dword ptr 8
nNumberOfBytesToRead= dword ptr 0Ch
lpNumberOfBytesRead= dword ptr 10h

sub esp, 14h
xor eax, eax
mov edx, [esp+14h+nNumberOfBytesToRead]
mov [esp+14h+Overlapped.Internal], eax
push esi
mov esi, [esp+18h+lpNumberOfBytesRead]
mov [esp+18h+Overlapped.InternalHigh], eax
mov [esp+18h+Overlapped.Offset], eax
lea ecx, [esp+18h+Overlapped]
mov [esp+18h+Overlapped.OffsetHigh], eax
push edi
mov edi, [esp+1Ch+hFile]
mov [esp+1Ch+Overlapped.hEvent], eax
mov [esp+1Ch+Overlapped.Offset], eax
mov [esp+1Ch+Overlapped.OffsetHigh], eax
mov [esp+1Ch+Overlapped.hEvent], eax
mov eax, [esp+1Ch+lpBuffer]
push ecx ; lpOverlapped
push esi ; lpNumberOfBytesRead
push edx ; nNumberOfBytesToRead
push eax ; lpBuffer
push edi ; hFile
call ds:ReadFile
test eax, eax
short loc_10001267
jz
```

Control then branches to handle errors or process results:

- loc_10001267:** Handles errors via `GetLastError`.
- loc_10001288:** Handles the result of the overlapped operation via `GetOverlappedResult`.
- loc_100012B0:** Returns the final status and exits the function.

Annotations at the bottom indicate the current instruction being executed:

80.00% (-29, -18) (721,606) 00001210 0000000010001210: ReadUSB

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... 🤔

The screenshot shows three assembly code snippets stacked vertically, each enclosed in its own window. A red arrow points from the bottom of the first window down to the top of the second, and another red arrow points from the bottom of the second window down to the top of the third.

```
loc_41C6A1:
mov    edx, [edi+0Ch]          ; ArgList
push   edx, [esi+0F0h]          ; 
lea    eax, [esi+0F0h]          ; 
push   offset aLibLedFanIni ; "%s\\LIB\\LedFan.ini"
push   eax, [esi+0E8h]          ; int
call   sub_409D90
add    esp, 0Ch
push   eax, [esp+0Ch]          ; 
call   sub_4016B0
mov    byte ptr [esp+18h+arg_89068], 2
mov    eax, [esp+18h+picce.dsSize]
push   eax, [esp+18h+picce.dsText]
push   eax, [esp+18h+picce.dsCaption]
call   ds:LoadLibraryW
mov    eax, [esi+0E8h], eax
test   eax, eax
jnz    short loc_41C70E

loc_41C70E:
mov    edx, [esp+18h+picce.dsSize]
push   edx, [edi+0Ch]          ; ArgList
lea    eax, [esp+18h+lpText]
push   offset aCan'tLoad ; "Can't Load %s"
push   eax, [esi+0E8h]          ; int
call   sub_409D90
mov    ebx, [esp+24h+lpText]
add    esp, 0Ch
push   0
push   offset Caption ; "Information"
push   eax, [esp+24h+lpText]
push   0, [eax+0Ch]            ; hIcon
call   ds:MessageBoxW

loc_41C7F:
mov    ecx, [esi+0E8h]
mov    edi, ds:GetProcAddress
push   offset aOpenUSBDevice ; "OpenUSBDevice"
push   ecx, [edi+0Ch]          ; hModule
call   edi, [esi+0Ch]
mov    edi, [esi+0E8h]
push   offset aCloseUSBDevice ; "CloseUSBDevice"
push   edx, [edi+0Ch]          ; hModule
mov    dword_5AF3A0, eax
call   edi, [esi+0Ch]
mov    edi, [esi+0Ch]
mov    eax, [esi+0E8h]
push   offset aWriteUSB ; "WriteUSB"
push   eax, [edi+0Ch]          ; hModule
call   edi, [esi+0Ch]
mov    edi, [esi+0E8h]
push   offset aReadUSB ; "ReadUSB"
push   ecx, [edi+0Ch]          ; hModule
mov    dword_5AF398, eax
call   edi, [esi+0Ch]
jmp   dword_5AF398, eax
mov    dword_5AF394, eax
jz    short loc_41C7F
```

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

Just quickly clean this up...

Hold up, I know this game... 🤔



The screenshot shows three assembly code snippets in a debugger:

```
loc_41C6A1:
mov    edx, ledil
push   edx, [esi+0Ch] ; Arglist
lea    eax, [esi+0Ch] ; 
push   offset aSlibLedFanIni ; "LIB\LedFan.ini"
push   eax, [esi+0Ch] ; int
call   sub_409090
add   esp, 0Ch
push   eax, [esi+0Ch]
call   sub_4016B0
lea    eax, [esp+1Ch+piccel]
call   sub_4016B0
mov    byte ptr [esp+18h+arg_89068], 2
mov    eax, [esp+18h+picce.dsSize]
push   eax, [esi+0Ch] ; lpLibrileName
call   ds:LoadLibraryA
mov    [esi+0Ch], eax
test   eax, eax
jnz   short loc_41C70E
```

```
loc_41C70E:
mov    edx, [esp+18h+picce.dsSize]
push   edx, [esi+0Ch] ; Arglist
lea    eax, [esp+1Ch+lpText]
push   offset aCan'tLoad ; "Can't Load %s"
push   eax, [esi+0Ch] ; int
call   sub_409090
mov    ebx, [esp+24h+lpText]
add   esp, 0Ch
push   0
push   offset Caption ; uType
push   eax, [esi+0Ch] ; lpText
push   0, [esi+0Ch] ; hnd
call   ds:MessageBoxW
```

```
loc_41C70E:
mov    ecx, [esi+0Ch]
mov    edi, ds:GetProcAddress
push   offset aOpenUSBDevice ; "OpenUSBDevice"
push   ecx, [esi+0Ch] ; hModule
call   edi : GetProcAddress
mov    edx, [esi+0Ch]
push   offset aCloseUSBDevice ; "CloseUSBDevice"
push   edx, [esi+0Ch] ; hModule
mov    OpenUSBDevice, eax
call   edi : GetProcAddress
mov    CloseUSBDevice, eax
mov    eax, [esi+0Ch]
push   offset aWriteUSB ; "WriteUSB"
push   eax, [esi+0Ch] ; hModule
call   edi : GetProcAddress
mov    ecx, [esi+0Ch]
push   offset aReadUSB ; "ReadUSB"
push   ecx, [esi+0Ch] ; hModule
mov    WriteUSB, eax
call   edi : GetProcAddress
cmp    eax, 0
mov    ReadUSB, eax
jz    short loc_41C70F
```

At the bottom of the debugger window, it says: 80.00% (43,3067) (584,392) 0001BB62 00000000041C762: sub_41C4D0+292

Just quickly clean this up...

The image shows a debugger interface with three assembly code windows arranged vertically. Blue arrows indicate a flow from the top window down to the middle one, and then down again to the bottom one. The bottom window has a green border around its code area.

```
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+piccel]
call   sub_401680
mov    byte ptr [esp+18h+arg_89068], 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 aCan'tLoads ; "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!

File Edit Jump Search View Debugger Options Windows Help

Library function Regular function Instruction Data Unexplored External symbol

Functions window

Function name

- sub_401000
- sub_401010
- sub_401020
- nullsub_4
- sub_401040
- sub_401050
- sub_401080
- sub_4010A0
- sub_4010D0
- sub_401140
- sub_401180
- sub_401210
- sub_4012E0
- sub_401310

Line 3 of 8983

Graph overview

IDA View-A Hex View-1 Structures Enums Imports Exports

; Attributes: library function bp-based frame
; int _stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nShowCmd)
_wWinMain@16 proc near

hInstance= dword ptr 8
hPrevInstance= dword ptr 0Ch
lpCmdLine= dword ptr 10h
nShowCmd= dword ptr 14h

mov edi, edi
push ebp
mov ebp, esp
pop ebp
jmp loc_5419BC

loc_541992:
mov edi, edi
push ebp
mov ebp, esp
call ?AfxGetModuleState@@YGPAPVAFX_MODULE_STATE@X ; AfxGetModuleState(void)
mov cl, byte ptr [ebp+hInsta
mov [eax+14h], cl
xor eax, eax
inc eax
pop ebp
ret 8

loc_5419BC:
mov edi, edi
push ebp
mov ebp, esp
push ebx
push esi
push edi
or ebx, OFFFFFFFFh
call ?AfxGetThread@@YGPAVCWinThread@@X ; AfxGetThread(void)
mov esi, eax
call ?AfxGetModuleState@@YGPAPVAFX_MODULE_STATE@X ; AfxGetModuleState(void)
push [ebp+nShowCmd] ; int
mov edi, [eax+4]
push [ebp+lpCmdLine] ; wchar_t *
push [ebp+hPrevInstance] ; HINSTANCE
push [ebp+hInstance] ; HINSTANCE

100.00% (-46, -21) (28, 63) 00140D87 0000000000541987: wWinMain(x,x,x,x) (Synchronized with Hex View-1)

Output window

The initial autoanalysis has been finished.

Python

AU: idle Down Disk: 11GB

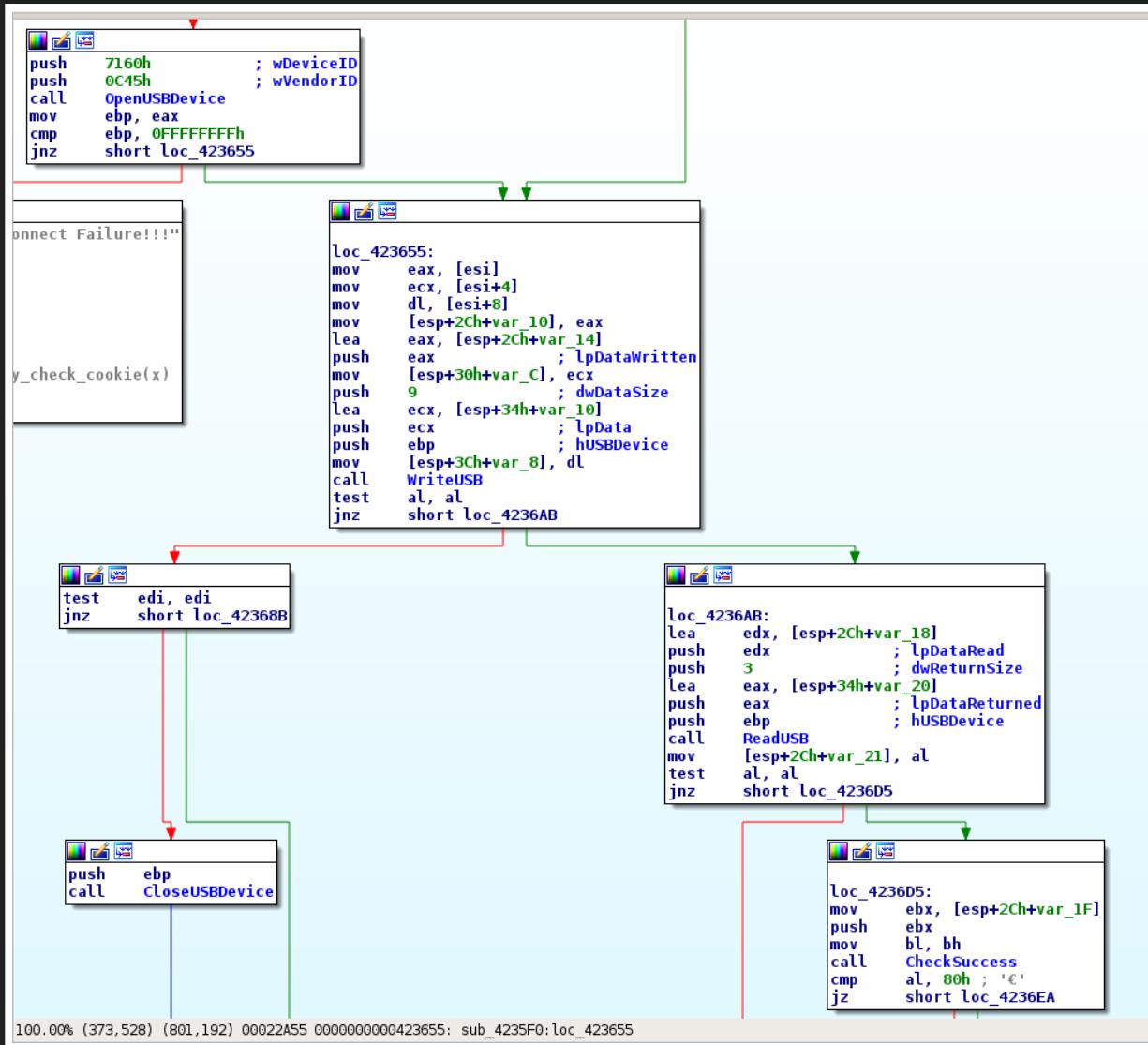
Straight to the interesting bit! 😊

Directio	Typ	Address	Text
↓	D...	r sub_4235F0+86	call WriteUSB
↓	D...	r sub_41C4D0+2A2	cmp WriteUSB, 0
↓	w	sub_41C4D0+284	mov WriteUSB, eax

Line 1 of 3

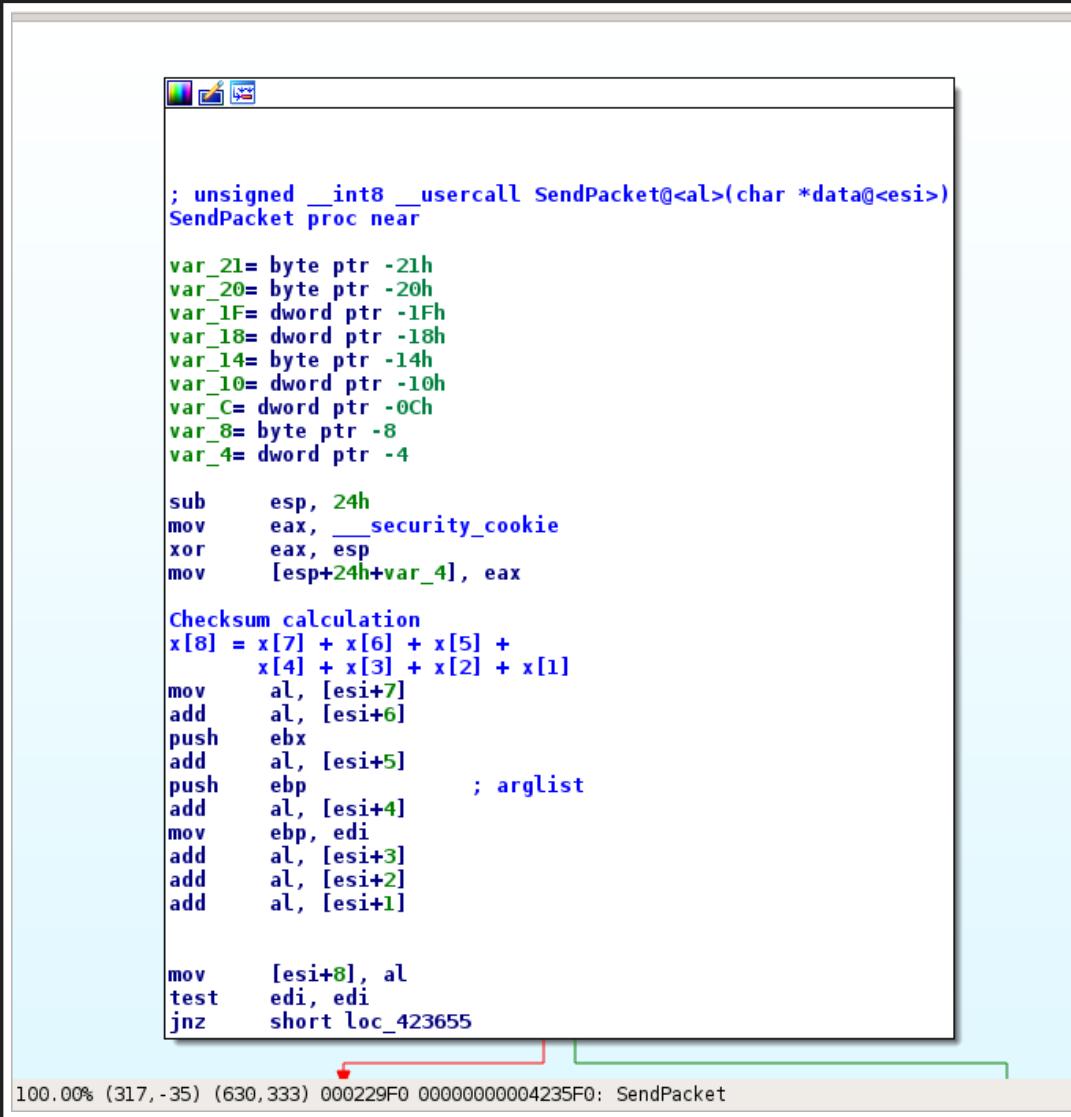
OK Cancel Search Help

One caller looks promising



Bingo 😊

Starting to learn about the protocol



The screenshot shows a debugger window displaying assembly code for a function named `SendPacket`. The code is written in Intel x86 assembly language. It includes variable declarations, memory allocations, and a checksum calculation loop. A red arrow points from the bottom status bar to the instruction `test edi, edi`, which is highlighted in green. The status bar at the bottom provides performance metrics and the current assembly address.

```
; 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

100.00% (317, -35) (630, 333) 000229F0 0000000004235F0: SendPacket
```

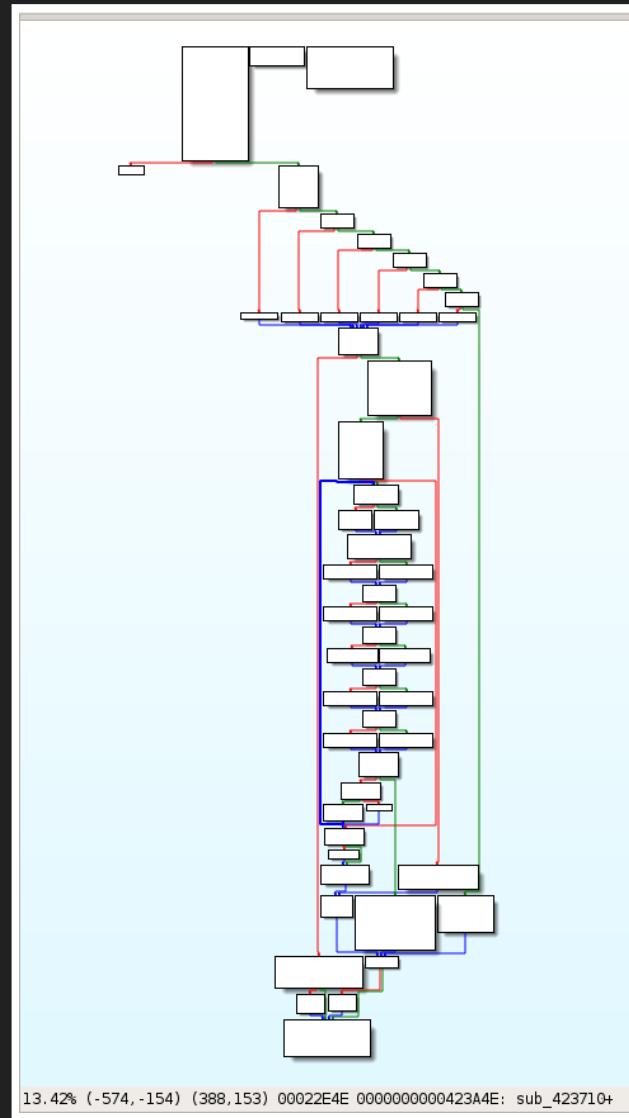
Only one function that calls into it 😊

Directio	Typ	Address	Text
...	p	sub_423710+18B	call SendPacket
...	p	sub_423710+33E	call SendPacket

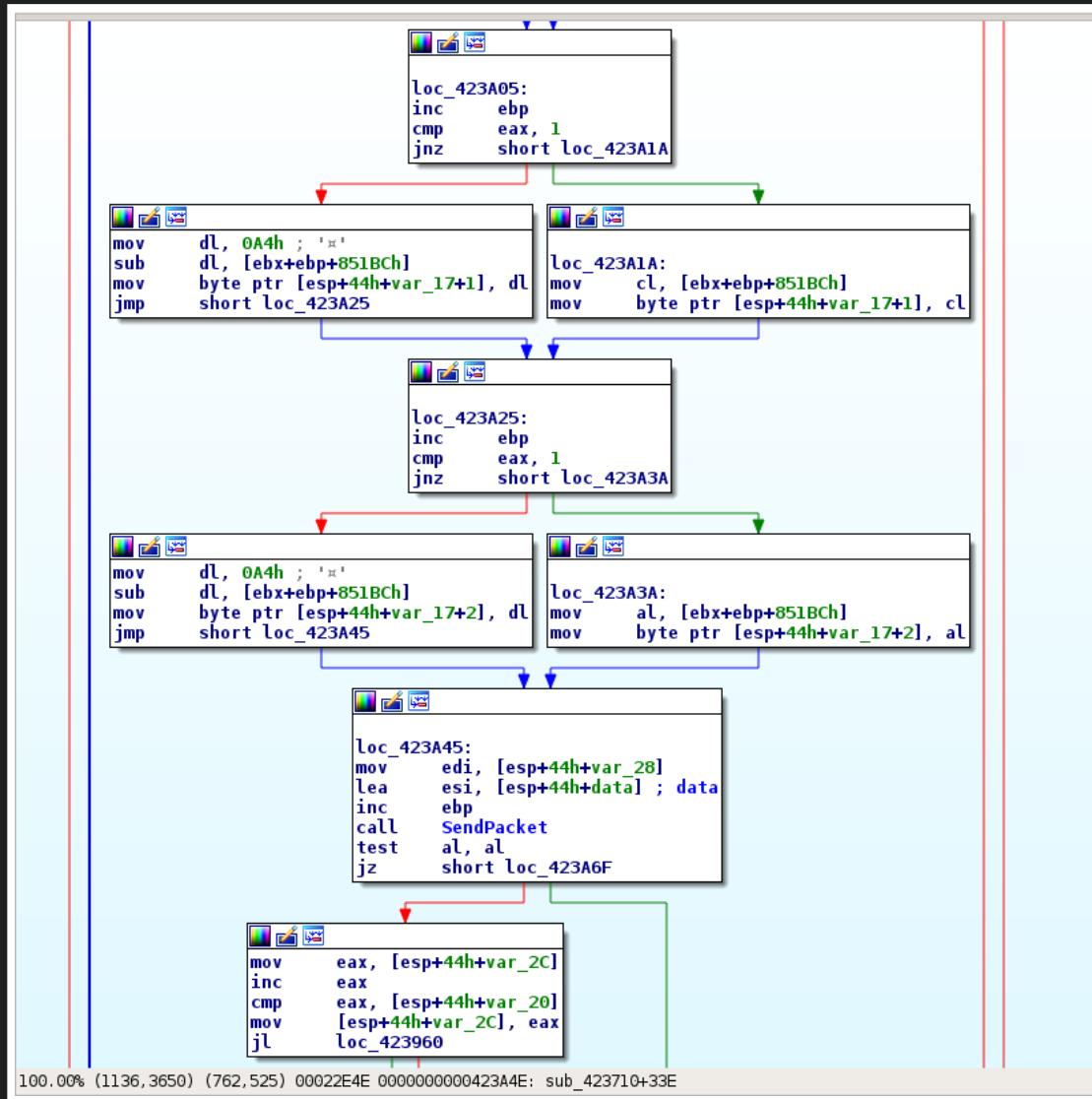
Line 2 of 2

OK Cancel Search Help

Big, but still learning!



Protocol seems to have an optional subtraction?



We also have the initial packet

The screenshot shows the Immunity Debugger interface with the assembly window open. The assembly code is as follows:

```
jnz    loc_42385E
loc_42385E:
Zero the initial packet

loc_42385E:
xor    eax, eax
mov    dword ptr [esp+44h+data+1], eax
mov    dword ptr [esp+44h+data+5], eax

data[0] = '\0'
mov    [esp+44h+data], al

data[3] = sizeish??? // Weird size log_2 thing
mov    al, [esp+44h+sizeish??]
mov    [esp+44h+data+3], al

data[4] = (unsigned char)size??? // Looks like this is the size
mov    eax, [esp+44h+size??]
mov    [esp+44h+data+4], al
shr    eax, 8           ; size??? >> 8
lea    esi, [esp+44h+data] ; data

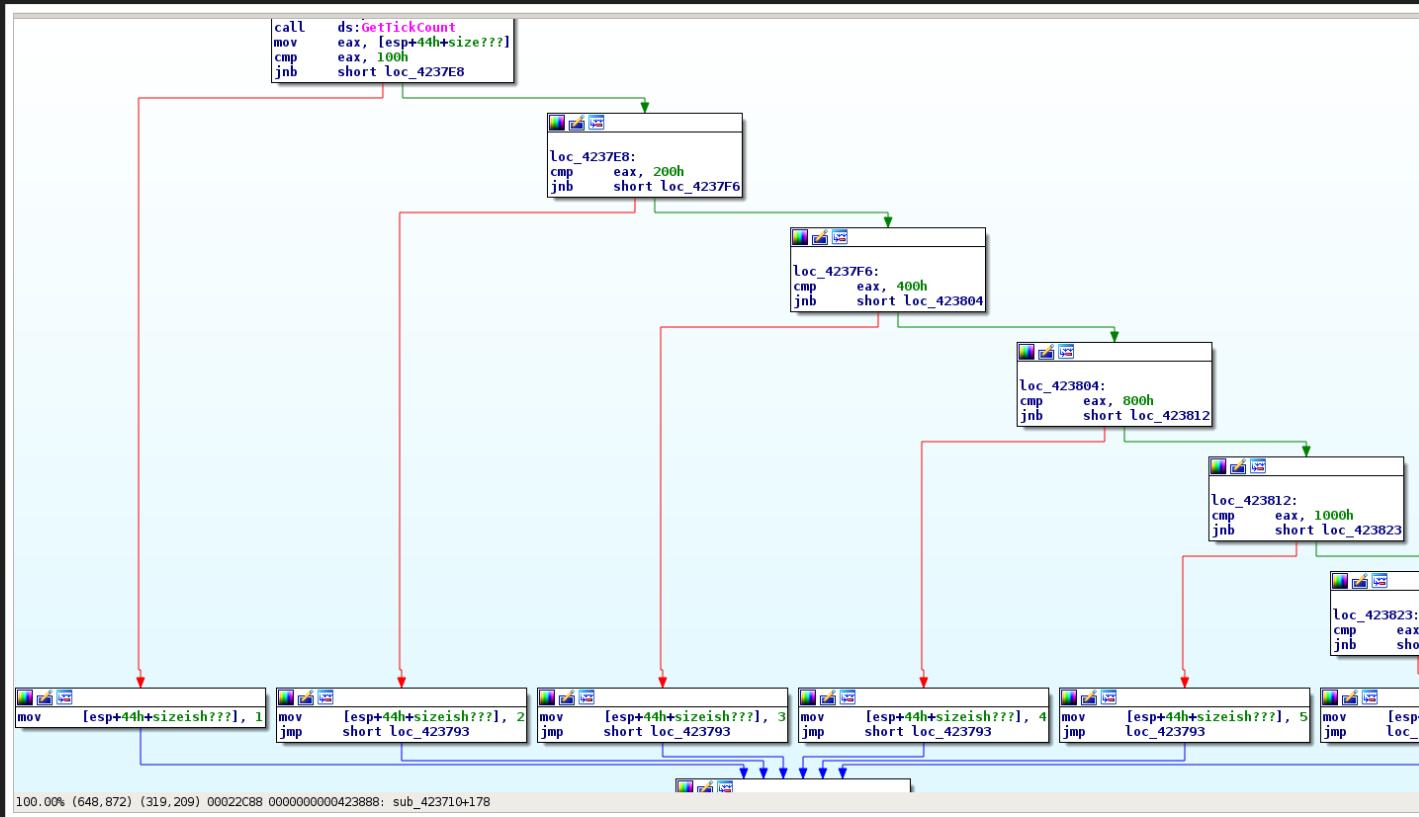
data[1] = '@'
mov    [esp+44h+data+1], 40h ; '@'
data[2] = '@'
mov    [esp+44h+data+2], 40h ; '@'
data[6] = '\0'
mov    [esp+44h+data+6], 0
data[7] = '\0'
mov    [esp+44h+data+7], 0

data[5] = (unsigned char)(size??? >> 8)
mov    [esp+44h+data+5], al

data = [\x00][\x40][\x40][sizeish??][size???.size??][\x00][\x00]
          ^^^little endian^^
call    SendPacket
test   al, al
jnz    short loc_4238C5

100.00% (1320,1786) (672,375) 00022C88 0000000000423888: sub_423710+178
```

The assembly code is annotated with comments explaining its purpose. It initializes memory at [esp+44h+data] with zeros, sets data[3] to sizeish??? (a weird size log_2 thing), calculates data[4] as the size (unsigned char) of size???, and then fills the buffer with '@' characters and zeros. It then calls the SendPacket function. The assembly window has a title bar "loc_42385E" and a status bar at the bottom showing CPU usage and memory addresses.



$$\begin{aligned}
sizeish &= \log_2 \left(\frac{\text{size}}{64} \right) \\
&= \log_2(\text{size}) - 6
\end{aligned}$$

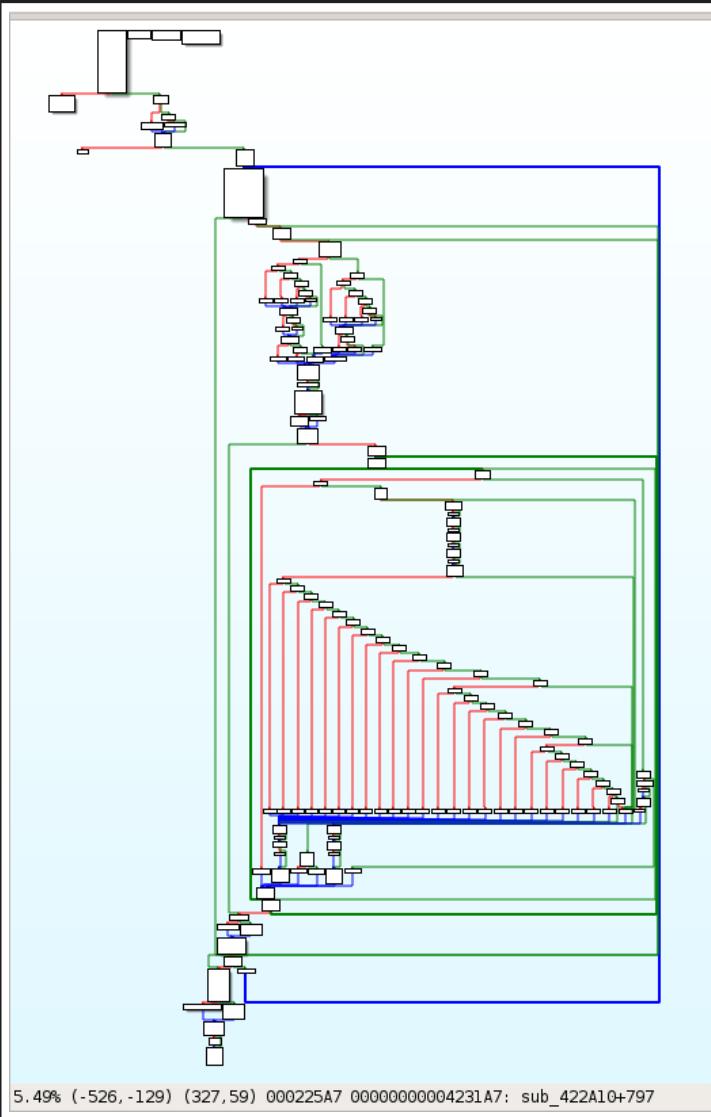
Also only one caller!

Directio	Typ	Address	Text
Up	p	sub_422A10+797	call SendMessage

Line 1 of 1

OK Cancel Search Help

But it's bigger, and uglier



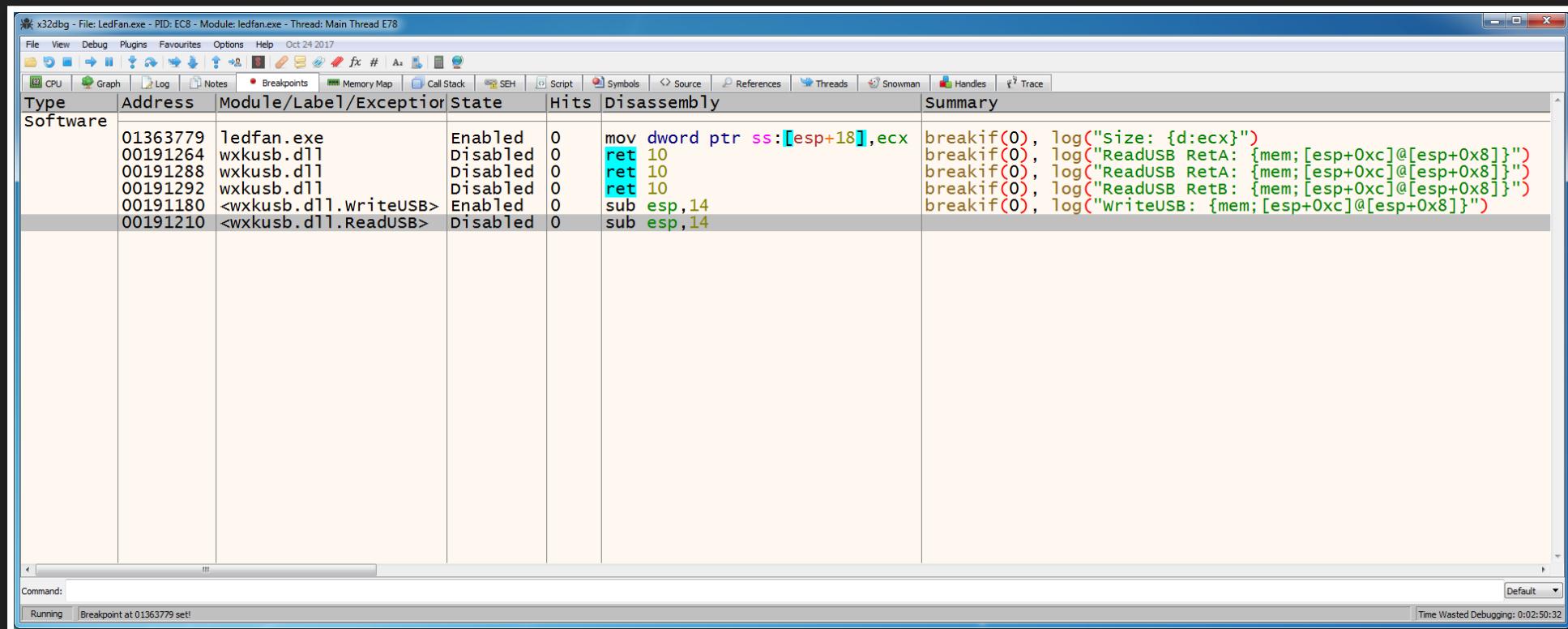
STEP BACK

- 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?

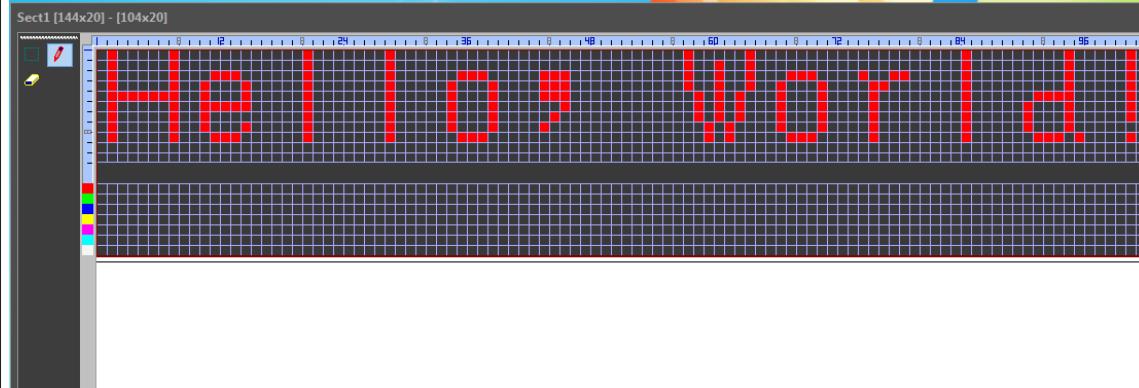
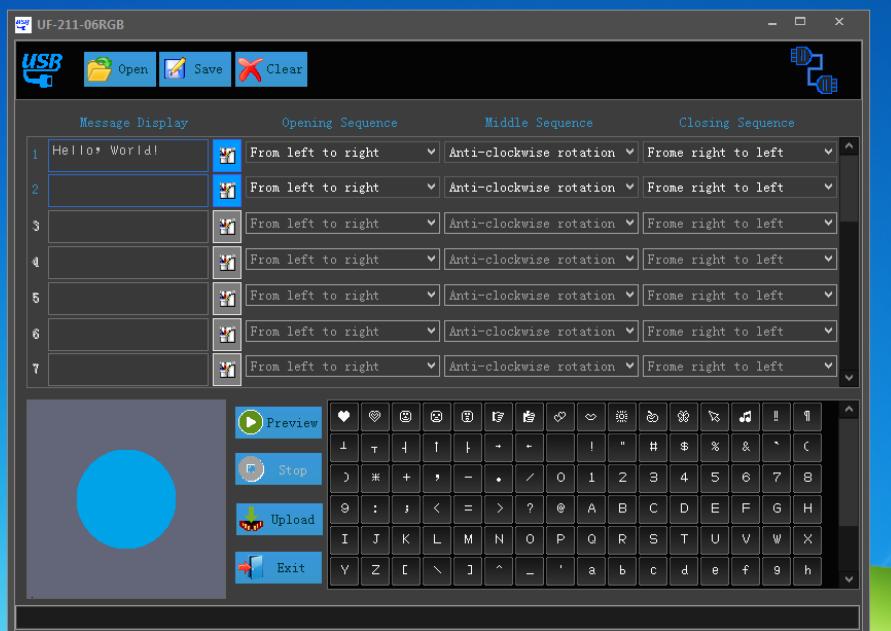
Lets validate some assumptions and dump the messages!



The screenshot shows the x32dbg debugger interface with a list of breakpoints. The table has columns for Type, Address, Module/Label/Exception, State, Hits, Disassembly, and Summary.

Type	Address	Module/Label/Exception	State	Hits	Disassembly	Summary
Software	01363779	ledfan.exe	Enabled	0	mov dword ptr ss:[esp+18],ecx	breakif(0), log("Size: {d:ecx}")
	00191264	wxkusb.dll	Disabled	0	ret 10	breakif(0), log("ReadUSB RetA: {mem; [esp+0xc]@[esp+0x8]}")
	00191288	wxkusb.dll	Disabled	0	ret 10	breakif(0), log("ReadUSB RetA: {mem; [esp+0xc]@[esp+0x8]}")
	00191292	wxkusb.dll	Disabled	0	ret 10	breakif(0), log("ReadUSB RetB: {mem; [esp+0xc]@[esp+0x8]}")
	00191180	<wxkusb.dll.WriteUSB>	Enabled	0	sub esp,14	breakif(0), log("WriteUSB: {mem; [esp+0xc]@[esp+0x8]}")
	00191210	<wxkusb.dll.ReadUSB>	Disabled	0	sub esp,14	

At the bottom, a command window shows "Running" and "Breakpoint at 01363779 set!", and a status bar indicates "Time Wasted Debugging: 0:02:50:32".



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: 00404001E200000063
WriteUSB: 004023A4233AA4A4AC
WriteUSB: 004023A4A4A484A477
WriteUSB: 00402384A484A48336
WriteUSB: 0040232584A484A4D8
WriteUSB: 00402384A484A48336
WriteUSB: 004023A484A5839447
WriteUSB: 004023839483948415
WriteUSB: 004023C484A484A477
WriteUSB: 00402384A484A48437
WriteUSB: 004023A483A584A457
WriteUSB: 00402384A484A48437
WriteUSB: 004023A484A484A053
WriteUSB: 00402384A0849C832A
WriteUSB: 004023AC84A084A45B
WriteUSB: 00402384A484A48437
WriteUSB: 004023AC83A083A055
WriteUSB: 00402383A084AC843A
WriteUSB: 004023A484A4849548
WriteUSB: 004023843483248446
WriteUSB: 0040232683248434E8
WriteUSB: 004023849584A48428
WriteUSB: 004023A484A484A457
WriteUSB: 00402384A484A48437
WriteUSB: 004023A484A484A457
WriteUSB: 00402384A484A48437
WriteUSB: 004023688448840823
WriteUSB: 00402384A484A48437
WriteUSB: 004023A484A484AC5F
WriteUSB: 00402383A083A0832C
WriteUSB: 004023A084AC84A45B
WriteUSB: 00402384A484A48437
WriteUSB: 004023A484A483A557
WriteUSB: 00402384A484A48437
WriteUSB: 004023A484A484A457
WriteUSB: 00402384A484A48336
WriteUSB: 004023A584A484A458
WriteUSB: 00402384A484A48437
WriteUSB: 004023A484EC83807A
WriteUSB: 0040238380838084ED
WriteUSB: 004023AC84A484A45F
WriteUSB: 00402383A584948427
WriteUSB: 004023948494849427
WriteUSB: 004023849483A58427
WriteUSB: 004023A4A4A4A4A497
WriteUSB: 004023A4A4A4A4A497

Command: Default

Running WriteUSB: 004023A4A4A4A4A497 Time Wasted Debugging: 0:02:52:28

Single Red Dot

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

Initial Packet

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

Checksum

0040400122000000**A3**

004023A4239AA4A4**0C**

004023A4A4A484A4**77**

00402384A484A484**37**

004023A484A484A4**57**

00402384A484A3A4**56**

004023A4A4A4A4**97**

$$40 + 40 + 01 + 22 + 00 + 00 + 00 = \text{A3} \quad \text{👍}$$

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 😎

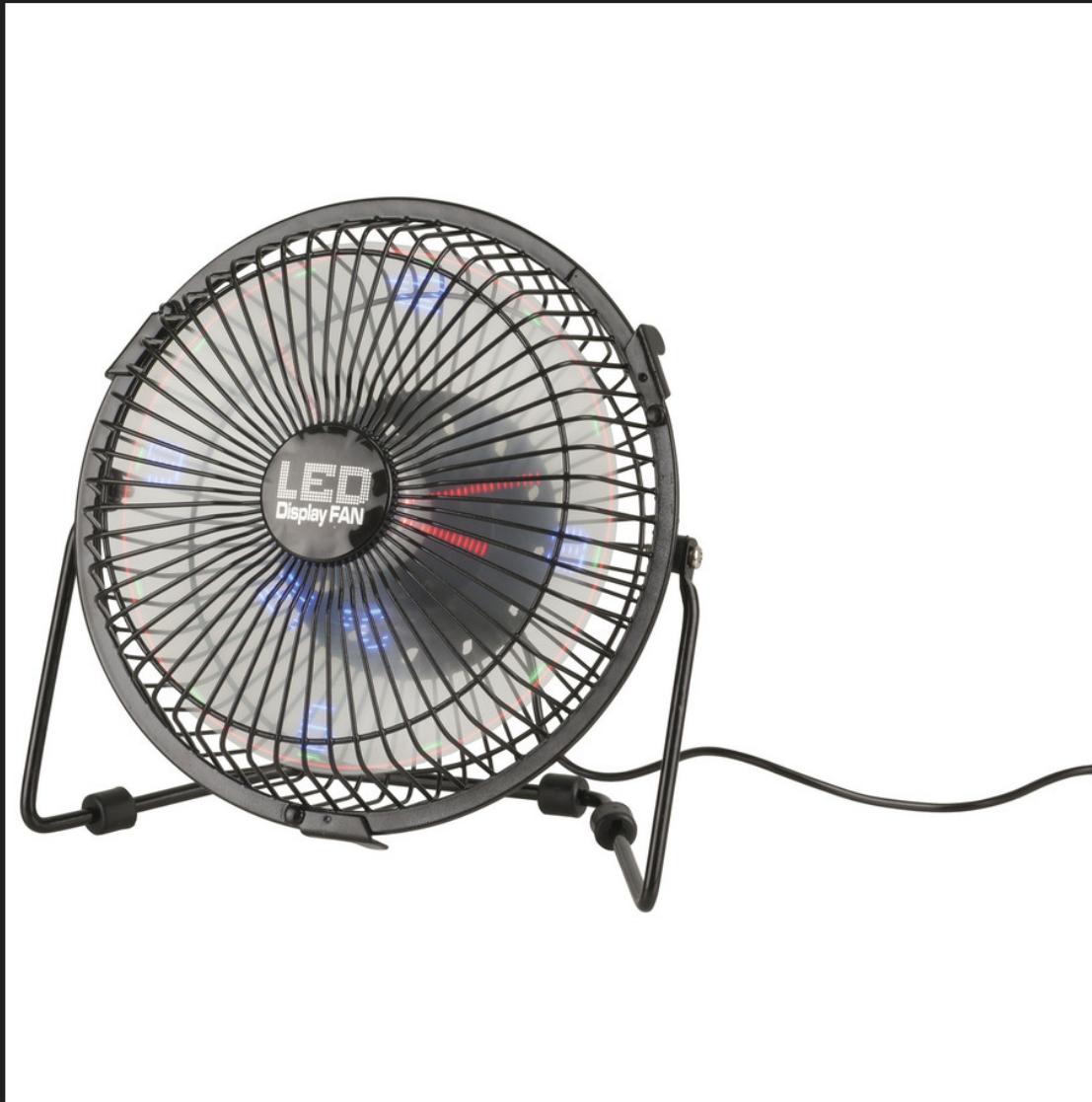






But it shorted when I put the case back on 😢

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



QUESTIONS

