

Random Acts of RE

Robert Graham

About me

Just a hacker (BlackICE, masscan, sidejacking)

I do only reverse engineering as a means to a goal.

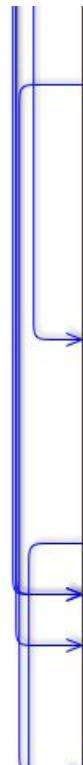
I have reverse engineered – I don't reverse engineer.

Each effort very different than the last

I don't do much

1. Get the thing (e.g. firmware update)
2. Extract the thing (e.g. find old version of Mac Stuffit to extract Cisco firmware)
3. Run tool to 'analyze' the thing
 - a. Massage the thing so the tool knows what to do
 - b. Massage some more
4. Document clues
 - a. Rename parms/variables
 - b. Document data structures
5. Produce disassembled code
 - a. Some like working with disassembly, I like working with decompiled

Example: your tools are often confused



```
0190eeb2        db    0x69 ; 'i'
0190eeb3        db    0x69 ; 'i'
0190eeb4        insb   byte [edi], dx
0190eeb6        jns    aEkcloudvoicewa+43
aEkcloudspeechr:
0190eeb8        db    "ek/cloud/SpeechRecognizer;", 0 ←
0190eed3        and    dword [ebx*2+0x6f], ecx
0190eed7        insd   dword [edi], dx
0190eed8        das
0190eed9        imul   esp, dword [esi+0x6c], 0x6b657479
0190eee0        das
0190eee1        arpl   word [edi+ebp*2+0x75], bp
0190eee5        das
0190eee7        push   ebx
aPeechutility:
0190eee8        db    "peechUtility;", 0 ←
0190eef6        and    dword [ebx*2+0x6f], ecx
0190effa        insd   dword [edi], dx
0190effb        das
0190effc        db    0x69 ; 'i'
0190effd        insb   byte [edi], dx
0190efff        jns    aEkcloudvoicewa+116
aEkcloudvoicewa:
0190ef01        db    "ek/cloud/VoiceWakeuper;", 0 ←
0190ef19        and    al, 0x4c
0190ef1b        arpl   word [edi+0x6d], bp
0190ef1e        das
0190ef1f        imul   esp, dword [esi+0x6c], 0x6b657479
0190ef26        das
0190ef27        arpl   word [edi+ebp*2+0x75], bp
0190ef2b        das
0190ef2d        push   edi
0190ef2e        popal
0190ef2f        imul   esp, dword [ebp+0x75], 0x70
0190ef33        ih    allreutilresource+2
; CODE XREF=aIzerresult+42
; CODE XREF=aIzerresult+39
```

this is in classes.dex

ASCII vs. x86

These constants are obviously lower-case letters in the range 0x61-0x7A.

There's only a 1-in-10,000 chance a 4-byte number will match 4 ASCII letters.

...and can we talk about ESP as a general purpose register??

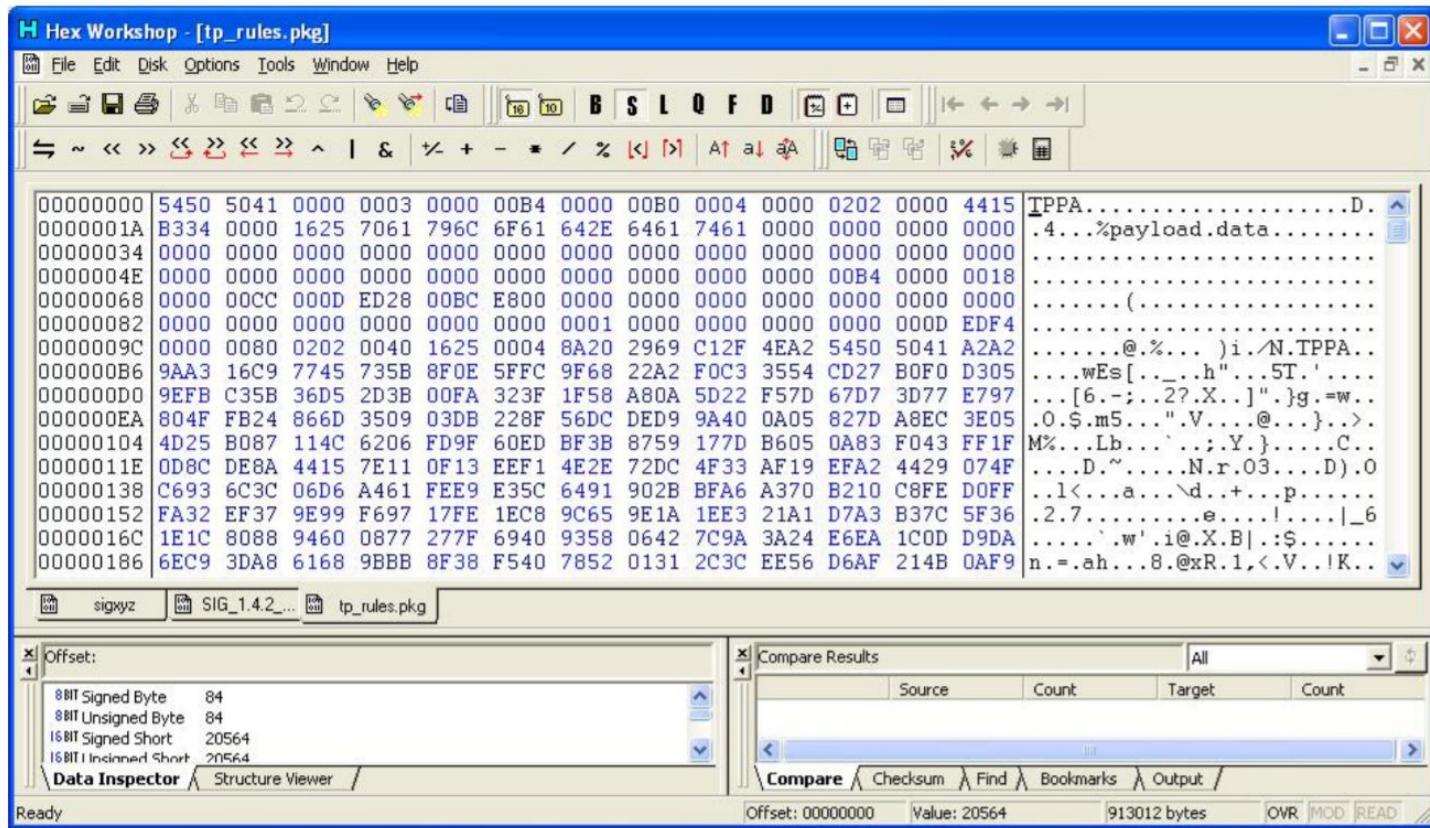
0190eeb8	db	"ek/cloud/SpeechRecognizer;", 0
0190eed3	and	dword [ebx*2+0x6f], ecx
0190eed7	insd	dword [edi], dx
0190eed8	das	
0190eed9	imul	esp, dword [esi+0x6c], 0x6b657479

Competitive Analysis, #1 - IPS signatures

```
<triggers>
  <trigger version="5500+" confidence="50">
    <payloads>
      <payload>
        <strmatch offset="0" depth="-1">|a0 01 00 00 00 00 00 [REDACTED] 0|
      </payload>
    </payloads>
  </trigger>
</triggers>

<tests>
  <test version="6024+" confidence="97">
    <protocol type="tcp">
      <ip>
        <saddr><eq><param refid="ip.saddr"/></eq></saddr>
        <daddr><eq><param refid="ip.daddr"/></eq></daddr>
      <tcp>
        <sport><eq><param refid="tcp.sport"/></eq></sport>
        <dport><eq><param refid="tcp.dport"/></eq></dport>
        <established/>
        <flags>A +</flags>
      </tcp>
    </ip>
  </protocol>
  <payloads>
    <payload>
      <strmatch offset="0" depth="-1">|a0 01 00 00 00 00 00 [REDACTED]
    </payload>
    <payload>
      <strmatch offset="0" depth="-1">|01 10 08 00 [REDACTED] |</strmatch>
    </payload>
  </payloads>
</test>
</tests>
```

Competitive, #2



Side note

**DATA FILES NEED
REVERSING
TOO!!!**

Competitive, #3

HexWorkshop is a tool for helping you deal with raw unknown binaries. You can



A screenshot of a Windows Notepad window titled "pkg_hdr.hsl - Notepad". The window contains C code defining a struct named "pkg_hdr". The code includes a preprocessor directive "#pragma byteorder(big_endian)". The struct definition is as follows:

```
#pragma byteorder(big_endian)

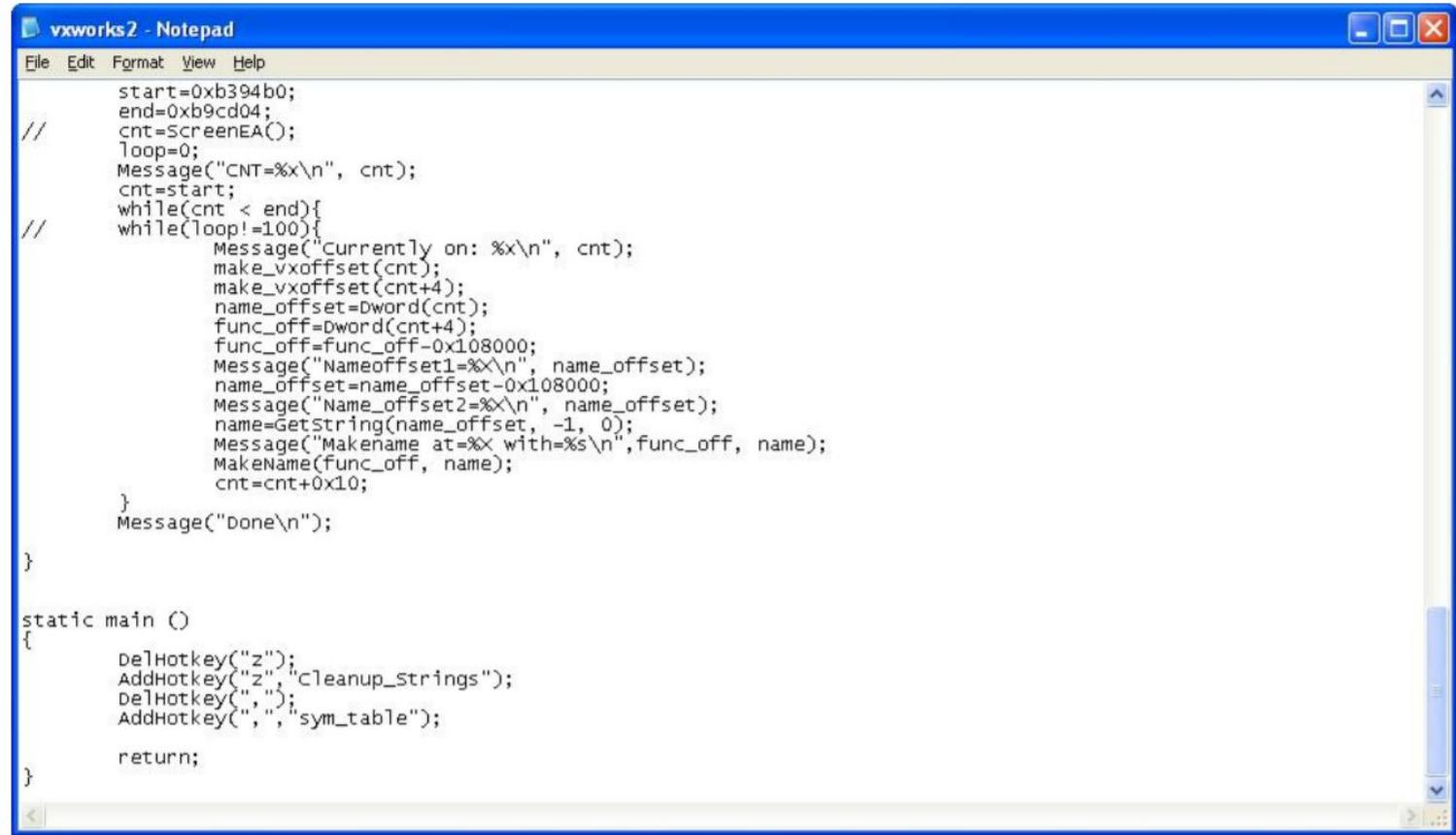
struct pkg_hdr
{
    DWORD Magic;
    DWORD Version;
    DWORD Size;
    DWORD unknw1;
    DWORD unknw2;
    DWORD unknw3;
    DWORD unknw4;
    DWORD unknw4_m;
    int name[16];
    DWORD key_offset;
    DWORD key_len;
    DWORD payload_data_start;
    DWORD unknw6;
    DWORD unknw7;
    DWORD unknw8;
    DWORD unknw9;
    DWORD unknw10;
    DWORD unknw11;
    DWORD unknw12;
    DWORD unknw13;
    DWORD unknw14;
    DWORD unknw15;
    DWORD unknw16;
    DWORD unknw17;
    DWORD unknw18;
    DWORD unknw19;
    DWORD unknw20;
    DWORD unknw21;
    DWORD unknw22;
    DWORD Magic_tail;
};
```

Competitive #4: analyzing the binary

The screenshot shows the IDA Pro debugger interface with the following details:

- Title Bar:** IDA - C:\Documents and Settings\dave\Desktop\TP-Audit\vxworks
- Menu Bar:** File, Edit, Jump, Search, View, Options, Windows, Help
- Toolbar:** Includes icons for file operations, search, and various analysis tools.
- Panels:**
 - Left Panel:** Shows assembly code for the .text section, including instructions like dec, outsb, jnb, popa, insb, add, jo, insd, popa, and db.
 - Right Panel:** Shows a memory dump pane displaying hex values (e.g., 70 49 6E 69 74 69 61 6C) and their corresponding ASCII representation (e.g., Rule._npInitia).
 - Bottom Panels:** Includes tabs for Hex View-A, Imports, Names, Functions, Strings, Structures, and Enums.
- Status Bar:** Displays the current file path (F:\0010DCCC), disk usage (Down: Disk: 22GB), and a message about the calling convention.

Competitive #4 fixups



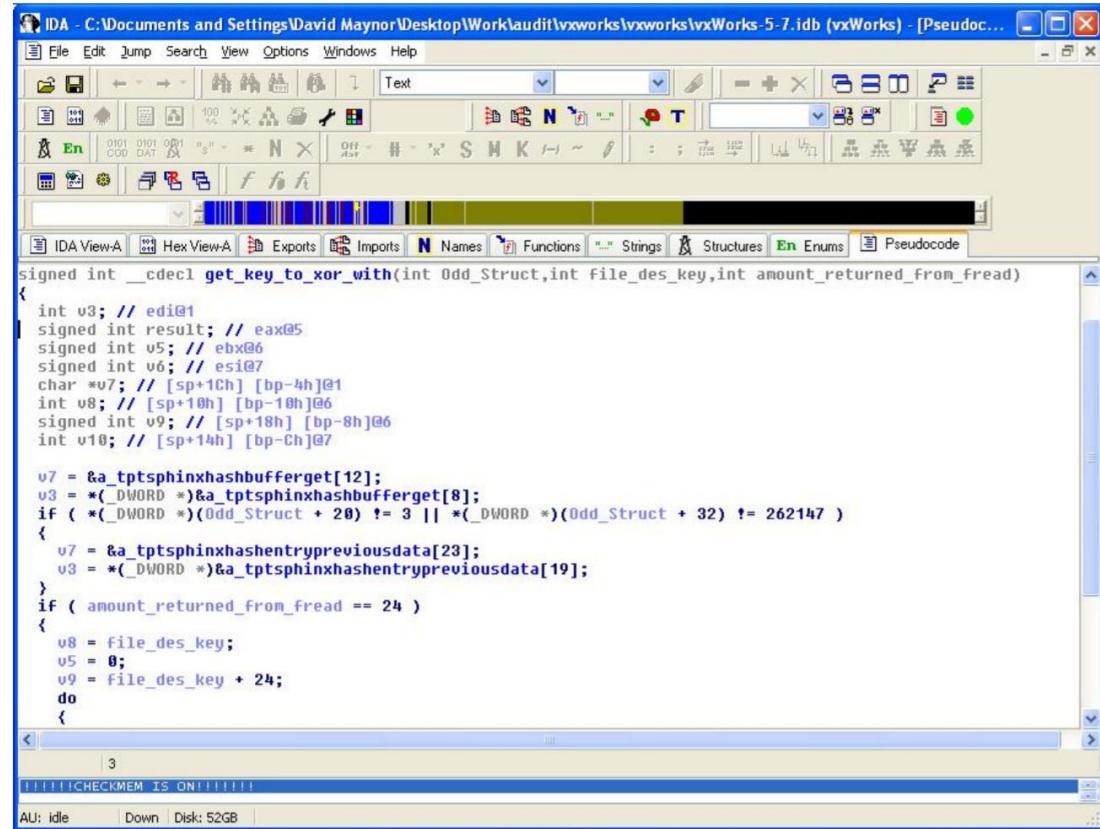
The screenshot shows a Windows-style Notepad window titled "vxworks2 - Notepad". The window contains the following C code:

```
vxworks2 - Notepad
File Edit Format View Help
start=0xb394b0;
end=0xb9cd04;
//cnt=ScreenEA();
loop=0;
Message("CNT=%x\n", cnt);
cnt=start;
while(cnt < end){
//    while(loop!=100){
        Message("Currently on: %x\n", cnt);
        make_vxoffset(cnt);
        make_vxoffset(cnt+4);
        name_offset=Dword(cnt);
        func_off=Dword(cnt+4);
        func_off=func_off-0x108000;
        Message("Nameoffset1=%x\n", name_offset);
        name_offset=name_offset-0x108000;
        Message("Name_offset2=%x\n", name_offset);
        name=GetString(name_offset, -1, 0);
        Message("Makename at=%x with=%s\n", func_off, name);
        MakeName(func_off, name);
        cnt=cnt+0x10;
    }
    Message("Done\n");
}

static main ()
{
    DelHotkey("z");
    AddHotkey("z", "cleanup_strings");
    DelHotkey(",");
    AddHotkey(",", "sym_table");

    return;
}
```

Competitive #5 zero in on the encryptions



signed int __cdecl get_key_to_xor_with(int Odd_Struct,int file_des_key,int amount_returned_from_fread){
 int v3; // edi@1
 signed int result; // eax@5
 signed int v5; // ebx@6
 signed int v6; // esi@7
 char *v7; // [sp+1Ch] [bp-4h]@1
 int v8; // [sp+10h] [bp-10h]@6
 signed int v9; // [sp+18h] [bp-8h]@6
 int v10; // [sp+14h] [bp-Ch]@7

 v7 = &a_tptspinhxhashbufferget[12];
 v3 = *(__DWORD *)&a_tptspinhxhashbufferget[8];
 if (*(__DWORD *)(Odd_Struct + 20) != 3 || *(__DWORD *)(Odd_Struct + 32) != 262147)
 {
 v7 = &a_tptspinhxhashentrypreviousdata[23];
 v3 = *(__DWORD *)&a_tptspinhxhashentrypreviousdata[19];
 }
 if (amount_returned_from_fread == 24)
 {
 v8 = file_des_key;
 v5 = 0;
 v9 = file_des_key + 24;
 do
 {
 3
 !!!!!CHECKMEM IS ON!!!!!!
AU: idle Down Disk: 52GB |

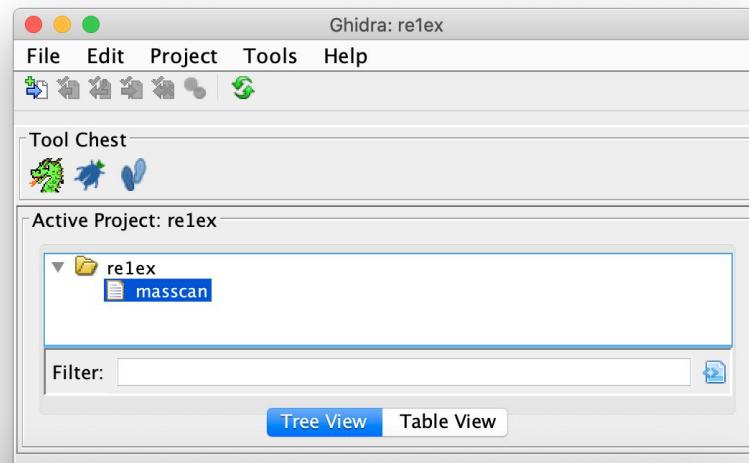
Competitive #6 - navigating the binary

- Log files
- Error messages
- File open/read
- Crypto routines

```
Default (108,22)
New Info Customize Close Execute Bookmarks
Default Default
15098 1142432042 0349999986 <DBG3> [UP] 0000 calling install with --sig_ .pkg--.
15099 1142432042 0399999984 <INFO> [UP ] 2004 Begin Package Install using /opt/tmp/download/i
mage.sig_ .pkg
15100 1142432042 0399999984 <DBG0> [UP ] 0000 Changing UpdateError to OK - No Error [1]
15101 1142432042 0399999984 <DBG0> [UP ] 0000 Changing UpdateState to Updating [2]
15102 1142432042 0416666650 <DBG0> [UP ] 0000 Changing UpdateStateQualifier to OK [1]
15103 1142432042 0433333316 <INFO> [UP ] 2015 DV update to .
15104 1142432042 0533333312 <INFO> [UP ] 2021 Extracting payload /opt/update/sec/ /p
ayload.data from pkg /opt/tmp/download/image/.pkg
15105 1142432048 0083333330 <INFO> [UP ] 4006 Replacing saved Digital Vaccine pkg with: /opt/
tmp/download/image/.pkg
15106 1142432048 0083333330 <DBG0> [UP ] 0000 upManifestRemoveFiles(/usr, /usr/dv-manifest)
15107 1142432048 0133333328 <DBG0> [UP ] 0000 upVaccinePkgRemove(dv)
15108 1142432049 0016666666 <DBG0> [UP ] 0000 upVaccineProcess(dv, /opt/update/sec/
, /payload.data ')
15109 1142432049 0099999996 <INFO> [UP ] 2801 Extracting /opt/update/sec/ /payload.d
ata to /usr
15110 1142432053 0733333304 <DBG0> [UP ] 2701 Validating dir: /usr, using manifest: /usr/mani
fest
15111 1142432054 0783333302 <INFO> [UP ] 2803 upTEaV: Successful extraction/validation of /op
t/update/sec/ /payload.data
```

Example#1 going backwards from strings

In this example, I'm going to work with a stripped version of 'masscan'. Masscan is fun because it has no dependencies

A screenshot of a terminal window titled "-zsh". The command entered is "ploppy@Roberts-Air masscan % bin/masscan 23.23.23.23 -p80". The output of the command is displayed:

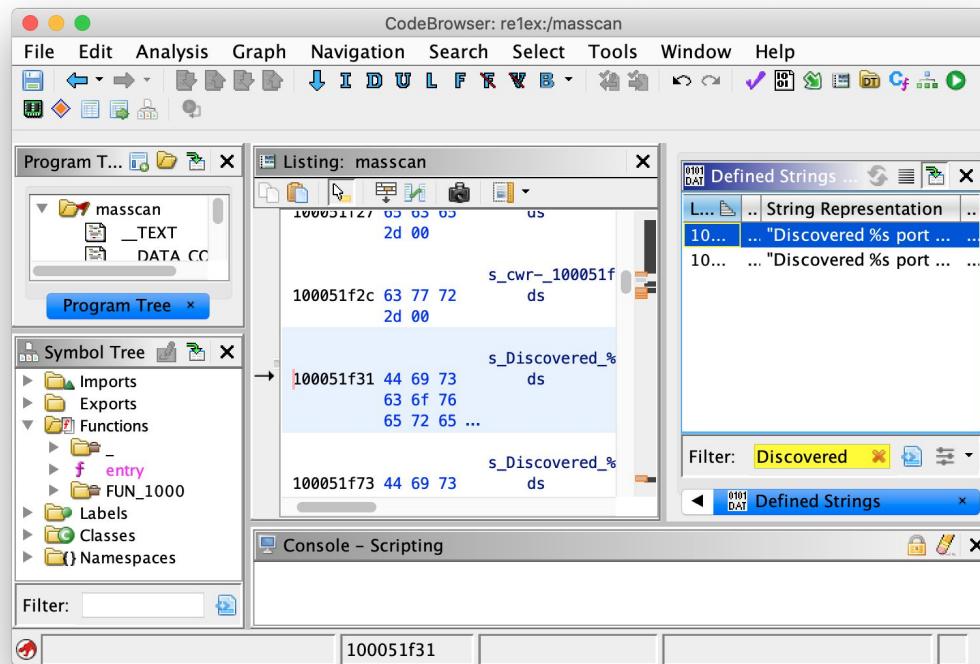
```
[ploppy@Roberts-Air masscan % bin/masscan 23.23.23.23 -p80
Starting masscan 1.3.2 (http://bit.ly/14GZzcT) at 2022-02-02 20:30:43 GMT
Initiating SYN Stealth Scan
Scanning 1 hosts [1 port/host]
Discovered open port 80/tcp on 23.23.23.23
ploppy@Roberts-Air masscan %
```

The word "Discovered" and the IP address "23.23.23.23" are highlighted in blue.

Example #2

Stripped, so function table useless

But what do I have?
Strings. So start with strings.



Example #3

Right click, go to
“References”, switch to
Decompiler view, scroll
down to function

The screenshot shows the CodeBrowser application interface with several windows open:

- Program... X**: Assembly listing window showing assembly code for a function. The assembly code includes instructions like PUSH, CALL, ADD, MOV, and CMP.
- Symb... X**: Symbol browser window showing categories like Imports, Exports, Functions, Labels, Classes, and Namespaces.
- Listing... X**: Another assembly listing window, likely a secondary view or a different part of the assembly dump.
- Decompile: FUN_10001f2b0 - (masscan) X**: Decompiler window showing the C-like pseudocode for the function. The code includes conditional branches based on variable uVar3 and uses printf statements to interact with stdio.
- Console - Scripting X**: A text-based console window.

The decompiler window contains the following pseudocode:

```
if (uVar3 == 0xdca4ca) goto LAB_10001f6bd;
if (uVar3 != 0xe4956e) goto LAB_10001f94e;
pcVar9 = "[random]";
}
LAB_10001f6c4:
uVar3 = __stubs::__fprintf(*(FILE **)_got::__stdoutp,
"Discovered %s port %u/%s on
pcVar7,(ulong)param_5,pcVar8,
(ulong)param_11[1],(ulong)par
(ulong)param_11[4],(ulong)par
)
else {
uVar3 = __stubs::__fprintf(*(FILE **)_got::__stdoutp,
(ulong)param_5,pcVar8,local_6
)
if (uVar3 < 0x50) {
__stubs::__fprintf(*(FILE **)_got::__stdoutp,"%.*s",
"
```

Example #4

Rename the variables in this function

The screenshot shows the CodeBrowser interface with the following components:

- Listing View:** Shows assembly code for the function `FUN_10001f2b0`. The assembly code includes instructions like MOV, XOR, CALL, and JMP. A specific instruction at address `10001f3f5` is highlighted.
- Decompiler View:** Shows the decompiled C code for the same function. It includes printf statements and variable declarations. The variable `port` is highlighted in yellow.
- Symbols View:** Shows the symbol browser with sections for Imports, Exports, Functions, Labels, Classes, and Namespaces. The `n` section is expanded, showing various symbols.
- Console - Scripting:** An empty console window for scripting.

The status bar at the bottom displays the assembly address `10001f3f5`, the decompiled function name `FUN_10001f2b0`, and the current instruction `MOV ECX,R12D`.

```
rprintf(*(FILE **)_got::__stdout,
        "Discovered %s port %u/%s on %s (%02x:%02x:%02x:%02x:%02x:%02x:%02x:%02x)\\n",
        status,(ulong)port,tcpudp,ipaddrstring,(ulong)*macaddr,
        (ulong)macaddr[1],(ulong)macaddr[2],(ulong)macaddr[3],
        (ulong)macaddr[4],(ulong)macaddr[5],pcVar7,uVar8);

rprintf(*(FILE **)_got::__stdout,"Discovered %s port %u/%s on %s"
        (ulong)port,tcpudp,ipaddrstring);

FILE **)_got::__stdout,"%.*s", (ulong)(0x4f - uVar3),
        "
```

Example

Rename and highlight

CodeBrowser: re1ex:/masscan

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program... X Listing: masscan... X Decompile: print_status_message - (masscan)

Symb... X Imports Exports Function Labels Classes Names

1 void print_status_message
2 (long param_1,undefined8 param_2,int param_3,uint param_4,uint port,
3 undefined4 param_6,undefined8 param_7,undefined8 param_8,undefined8
4 undefined4 param_10,byte *macaddr)
5
6 {
7 byte bVar1;
8 undefined *puVar2;
9 uint uVar3;
10 time_t tVar4;
11 off_t oVar5;
12 long *plVar6;
13 char *status;
14 char *tcpudp;
15 char *pcVar7;
16 undefined8 uVar8;
17 FILE *local_78;

10001f2b0 55
10001f2b1 48 89
10001f2b4 41 57
10001f2b6 41 56
10001f2b8 41 55
10001f2ba 41 54
10001f2bc 53
10001f2bd 48 8:
88 0:

Decompile: print_statu... Defined Strings

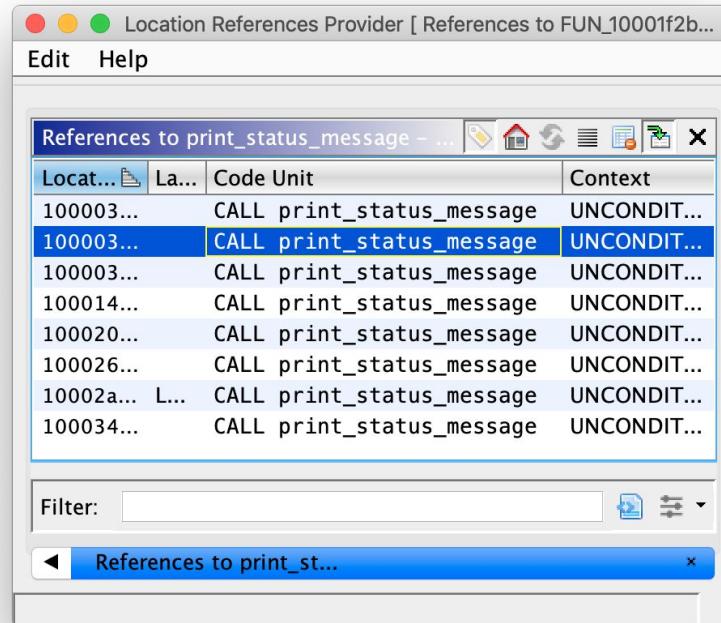
Console – Scripting

Filter:

10001f2b0 print_status_mes... PUSH RBP

Example

Now recursively find those callers of our “print_status_message()” function. Now we have many to choose from.



The screenshot shows a software window titled "Location References Provider [References to FUN_10001f2b...]". The window has "Edit" and "Help" menu options. Below the menu is a toolbar with icons for search, home, refresh, and others. The main area is a table titled "References to print_status_message - ...". The table has columns: "Locat...", "La...", "Code Unit", and "Context". There are eight rows, each showing a call to "print_status_message" from a different location. The second row is selected, highlighted with a blue border. The "Code Unit" column for the selected row shows "CALL print_status_message". The "Context" column for all rows shows "UNCONDIT...". At the bottom of the window is a "Filter:" input field and a "References to print_st..." button.

Locat...	La...	Code Unit	Context
100003...		CALL print_status_message	UNCONDIT...
100003...		CALL print_status_message	UNCONDIT...
100003...		CALL print_status_message	UNCONDIT...
100014...		CALL print_status_message	UNCONDIT...
100020...		CALL print_status_message	UNCONDIT...
100026...		CALL print_status_message	UNCONDIT...
10002a...	L...	CALL print_status_message	UNCONDIT...
100034...		CALL print_status_message	UNCONDIT...

Example

Go up the call tree

The screenshot shows the CodeBrowser interface with the following windows:

- Program... X**: Shows the file structure of the program.
- Listing: masscan X**: Displays assembly code with memory addresses on the left and assembly instructions on the right. An arrow points from the assembly listing to the decompiled code window.
- Decompile: FUN_100003140 - (masscan) X**: Shows the decompiled C-like code corresponding to the assembly. The code handles parameters and local variables, including a print_status_message function call.
- Symb... X**: Shows the symbol browser with categories like Imports, Exports, Functions, Labels, Classes, and Namespaces.

```
24 bVar1 = param_3[8];
25 port = *(ushort *)(&param_3 + 9) << 8 | *(ushort *)(&param_3 + 9) >> 8;
26 bVar2 = param_3[0xb];
27 bVar3 = param_3[0xc];
28 if ((param_4 < 0x13) || (uVar6 != 0)) {
29     local_36 = 0;
30     local_3a = 0;
31     lVar4 = *(long *)(&param_1 + 0x228);
32 }
33 else {
34     local_36 = *(undefined2 *)(&param_3 + 0x11);
35     local_3a = *(undefined4 *)(&param_3 + 0xd);
36     lVar4 = *(long *)(&param_1 + 0x228);
37 }
38 if (lVar4 == 0) {
39     *(ulong *)(&param_1 + 0x228) = uVar7;
40 }
41 if ((param_5 == 0) ||
42     (((*(long *)(&param_5 + 0x78) == 0) || (iVar5 = FUN_10001af20(param_5), iVar5 != 0)) ||
43     (((*(long *)(&param_5 + 0x70) == 0) || (iVar5 = FUN_10001af50(param_5, port), iVar5 != 0)) ||
44     print_status_message(
45         (&param_1, (int)uVar7, param_2, bVar1, port, bVar2, CONCAT44(uStack84, uVar7,
46         CONCAT71(uStack71, 4), bVar3, &local_3a));
47 )
```

port = htons(*(short*)buf);

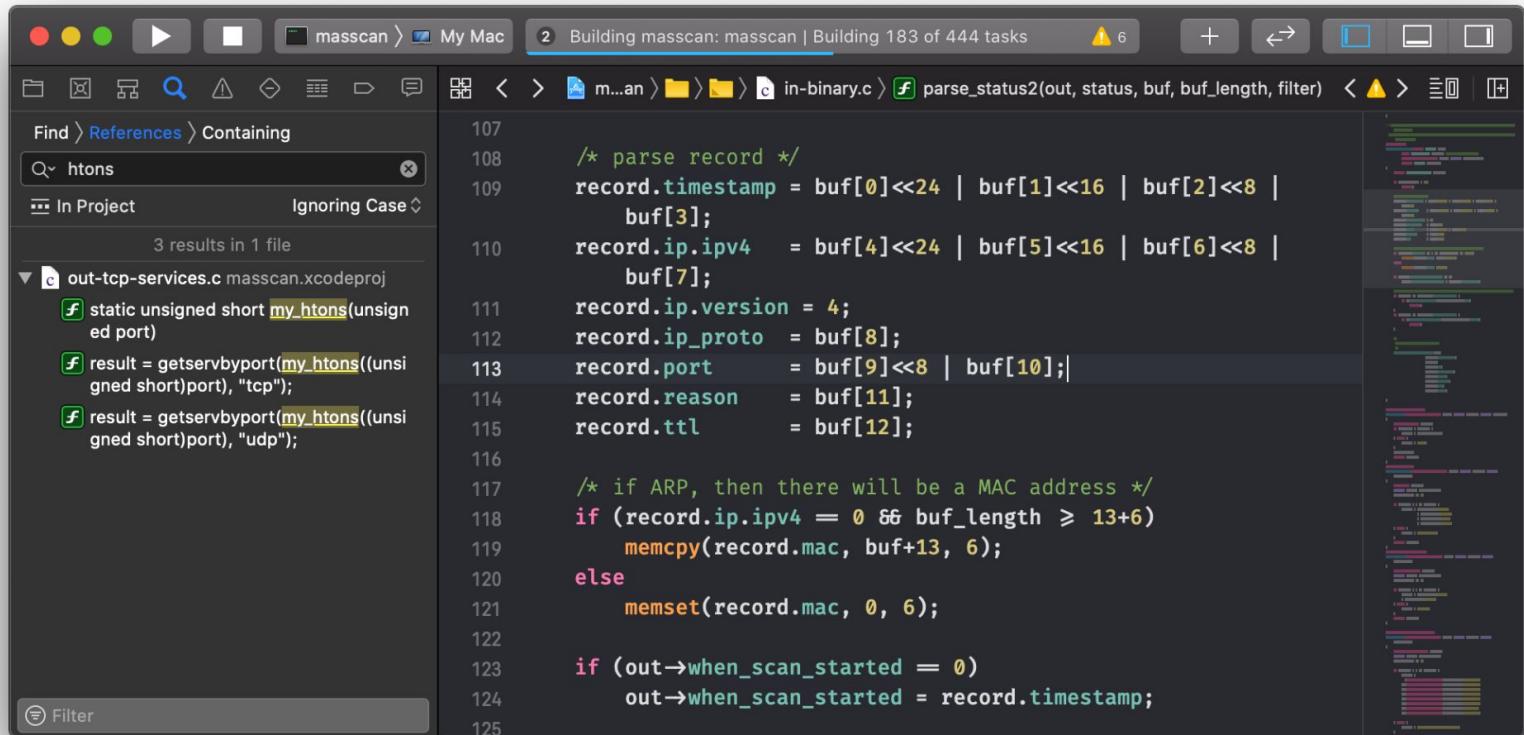
Reverse engineering
requires some
knowledge of
engineering –
recognizing patterns from
engineering, in this case,
htons()

The screenshot shows the re1ex debugger interface with the following windows:

- Program... X**: Shows the file structure of the program.
- Listing: masscan...**: Shows the assembly code listing. The assembly code includes instructions like `100003186 49 0f`, `100003189 44 85`, etc., followed by memory dump sections.
- Decompile: FUN_100003140 - (masscan)**: Shows the decompiled C code corresponding to the assembly. The code includes variable declarations like `uVar1`, `uVar2`, `uVar3`, and `lVar4`, and logic for handling `param_4` and `param_1`.
- Console - Scripting**: A text input field for scripting.
- Defined Strings X**: A list of defined strings.

```
CodeBrowser: re1ex:/masscan
File Edit Analysis Graph Navigation Search Select Tools Window Help
I D U L F R W B
Program... X
Listing: masscan...
Decompile: FUN_100003140 - (masscan)
100003186 49 0f
100003189 44 85
10000318d 8b 72
100003190 0f c6
100003192 89 75
100003195 c6 4f
100003199 8a 5e
10000319c 88 50
10000319f 0f b7
1000031a3 66 c3
1000031a7 66 89
1000031ab 8a 4f
1000031ae 88 4f
1000031b1 8a 4f
1000031b4 88 4f
1000031b7 48 8c
1000031bb 48 8c
1000031bf 72 1c
uVar1 = (ulong)buf[15] | (ulong)buf[12] << 8 | (ulong)buf[11] << 16 | (ulong)buf[8];
uVar6 = *(uint *)buf + 4;
uVar6 = uVar6 >> 0x18 | (uVar6 & 0xffff0000) >> 8 | (uVar6 & 0xff00) << 8 | uVar6;
uVar1 = buf[8];
port = *(ushort *)buf + 9 << 8 | *(ushort *)buf + 9 >> 8;
bVar2 = buf[0xb];
bVar3 = buf[0xc];
if ((param_4 < 0x13) || (uVar6 != 0)) {
    local_36 = 0;
    local_3a = 0;
    lVar4 = *(long *)param_1 + 0x228;
}
else {
    local_36 = *(undefined2 *)buf + 0x11;
    local_3a = *(undefined4 *)buf + 0xd;
    lVar4 = *(long *)param_1 + 0x228;
}
if (lVar4 == 0) {
    *(ulong *)param_1 + 0x7201 = 0x1a7.
}
Decompile: FUN_100003140 x 0101 DAT Defined Strings x
Console - Scripting
1000031a3 FUN_100003140 ROL BX,0x8
```

DISTRACTION: don't use htons()



The screenshot shows the Xcode IDE interface. The top bar displays the project name "masscan" and the build status "Building masscan: masscan | Building 183 of 444 tasks". A warning icon indicates 6 errors.

The left sidebar shows a search results panel for "htons". It lists 3 results found in 1 file, specifically in the file "out-tcp-services.c". The results are:

- static unsigned short my_htons(unsigned port)
- result = getservbyport(my_htons((unsigned short)port), "tcp");
- result = getservbyport(my_htons((unsigned short)port), "udp");

The main editor area displays the source code for the function `parse_status2`. The code uses `my_htons` instead of the standard `htons` function. The code snippet is as follows:

```
107     /* parse record */
108     record.timestamp = buf[0]<<24 | buf[1]<<16 | buf[2]<<8 |
109         buf[3];
110     record.ip.ipv4   = buf[4]<<24 | buf[5]<<16 | buf[6]<<8 |
111         buf[7];
112     record.ip.version = 4;
113     record.ip.proto   = buf[8];
114     record.port       = buf[9]<<8 | buf[10];
115     record.reason     = buf[11];
116     record.ttl        = buf[12];
117
118     /* if ARP, then there will be a MAC address */
119     if (record.ip.ipv4 == 0 && buf_length >= 13+6)
120         memcpy(record.mac, buf+13, 6);
121     else
122         memset(record.mac, 0, 6);
123
124     if (out->when_scan_started == 0)
125         out->when_scan_started = record.timestamp;
```

DISTRACTION (continued)

CodeBrowser: re1ex:/masscan

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program... X Listing: masscan Decompile... X

10000317e 48 09 f7 OR param_1,param_2
100003181 44 0f b6 MOVZX param_5,byte ptr [buf + 0x3]
42 03
100003186 49 09 f8 OR param_5,param_1
100003189 44 89 45 a8 MOV dword ptr [RBP + local_60],p
10000318d 8b 72 04 MOV param_2,dword ptr [buf + 0x4]
100003190 0f ce BSWAP param_2
100003192 89 75 b0 MOV dword ptr [RBP + local_58],p
100003195 c6 45 c0 04 MOV byte ptr [RBP + local_48],0x
100003199 8a 5a 08 MOV BL,byte ptr [buf + 0x8]
10000319c 88 5d c8 MOV byte ptr [RBP + local_40],BL
10000319f 0f b7 5a 09 MOVZX EBX,word ptr [buf + 0x9]
1000031a3 66 c1 c3 08 ROL port,0x8
1000031a7 66 89 5d ca MOV word ptr [RBP + local_3e],po
1000031ab 8a 42 0b MOV AL,byte ptr [buf + 0xb]
1000031ae 88 45 cc MOV byte ptr [RBP + local_3c],AL
1000031b1 8a 42 0c MOV AL,byte ptr [buf + 0xc]
1000031b4 88 45 cd MOV byte ptr [RBP + local_3b],AL
1000031b7 48 8d 7d ce LEA param_4=>local_3a,[RBP + -0x
1000031bb 48 83 f9 13 CMP param_4,0x13
1000031bf 72 1d JC LAB_1000031de
1000031c1 85 f6 TEST param_2,param_2
1000031c3 75 19 JNZ LAB_1000031de
1000031c5 0f b7 42 11 MOVZX EAX,word ptr [buf + 0x11]
1000031c9 66 89 47 04 MOV word ptr [RDI + local_1],AX
1000031cd 8b 42 0d MOV EAX,dword ptr [buf + 0xd]
1000031d0 89 07 MOV dword ptr [param_1]=>local_3

12 ulong uVar7;
13 undefined4 uStack84;
14 undefined8 local_50;
15 undefined7 uStack71;
16 undefined4 local_3a;
17 undefined2 local_36;
18
19 if (0x < param_4) {
20 uVar7 = (ulong)buf[3] | (ulong)
21 uVar6 = *(uint *) (buf + 4);
22 uVar5 = uVar6 >> 0x18 | (uVar
23 bVar1 = buf[8];
24 bVar2 = buf[0xb];
25 bVar3 = buf[0xc];
26 if ((param_4 < 0x13) || (uVar
27 local_36 = 0;
28 local_3a = 0;
29 lVar4 = *(long *) (param_1 +
30)
31 }
32 else {
33 local_36 = *(undefined2 *) (l
34 local_3a = *(undefined4 *) (l
35 lVar4 = *(long *) (param_1 +
36)
37
Decompile: FUN_100003140 x ►

1000031a3 100003140 ROL BX,0x8

Example

CodeBrowser: re1ex:/masscan

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program... X Listing: masscan... X Decompile: FUN_100002470 - (masscan)

Symb... X

Imports Exports Functor Labels Classes Namespaces

Console – Scripting

Filter:

100002470 55
100002471 48 89
100002474 41 51
100002476 41 56
100002478 41 56
10000247a 41 56
10000247c 53
10000247d 48 89
98 0f

lVar17 = (long)param_2;
do {
 pcVar1 = *(char **)(param_4 + lVar17 * 8);
 local_40 = (FILE *)0x0;
 buf = (ulong *)FUN_10004a9a0(0x100000);
 iVar4 = FUN_100045050(&local_40,pcVar1,"rb");
 if ((iVar4 == 0) && (local_40 != (FILE *)0x0)) &&
 (sVar6 = __stubs::__fread(buf,1,99,local_40), 0x62 < sVar6)) {
 if ((*((ulong *)((long)buf + 3) ^ 0x312e312f6e616373 | *buf ^ 0x2f6e61637373
 __stubs::__fprintf(*((FILE **)__got::__stderr,
 "ss: unknown file format (expeced \\"masscan/1.1\\")\n",
 goto joined_r0x000100002cc1);
 }
 if (((*(byte *)((long)buf + 0xb) == 0x2e) &&
 (uVar7 = __stubs::__strtoul((char *)((long)buf + 0xc),(char **)0x0,0), 1
 for (uVar7 = 1;
 (bVar12 = *((byte *)((long)buf + (uVar7 - 1))), bVar12 != 0 && (bVar12
 uVar7 = uVar7 + 4) {

Defined Strings x

100002470 FUN_100002470 PUSH RBP

The screenshot shows the CodeBrowser application window with several panes. The top menu bar includes File, Edit, Analysis, Graph, Navigation, Search, Select, Tools, Window, and Help. Below the menu is a toolbar with various icons. The left sidebar contains 'Program' and 'Symb...' tabs, along with lists for Imports, Exports, Functor, Labels, Classes, and Namespaces. The main area has three tabs: 'Listing: masscan...', 'Decompile: FUN_100002470 - (masscan)', and 'Console – Scripting'. The 'Decompile' tab displays assembly code with line numbers 34 to 51. The assembly code involves reading from a file, performing a check on the buffer contents, and then printing an error message if the condition fails. The 'Console' tab is currently empty. At the bottom, there's a status bar with the address 100002470, the function name FUN_100002470, and the instruction PUSH RBP.

Q: Where to start?

A: Anywhere.

You don't have what you want, so start with what you have.

Each of my projects is different. I like to start with 'strings' because no matter how much things have been stripped, there always strings to start with.

Blaster.c – reverse engineering a worm

Reverse engineer the code all the way back to the comments – find bugs and anomalies that can help us find open-source

```
1464     /* Major bug here: the worm writer incorrectly calculates the
1465      * IP checksum over the entire packet. This is incorrect --
1466      * the IP checksum is just for the IP header itself, not for
1467      * the TCP header or data. However, Windows fixes the checksum
1468      * anyway, so the bug doesn't appear in the actual packets
1469      * themselves.
1470 */
1471     ip.checksum = blaster_checksum(buf, sizeof(ip) + sizeof(tcp));
```

Godbolt forward engineering

The screenshot shows the Compiler Explorer interface on godbolt.org. On the left, the C++ source code for `blaster_checksum` is displayed:

```
1 int blaster_checksum(const void *bufv, int len)
2 {
3     const unsigned short *buf = (const unsigned short *)bufv;
4     unsigned long result = 0;
5
6     while (length > 1)
7     {
8         result += *(buf++);
9         length -= sizeof(*buf);
10    }
11    if (length)
12        result += *(unsigned char *)buf;
13    result = (result >> 16) + (result & 0xFFFF);
14    result += (result >> 16);
15    result = (~result) & 0xFFFF;
16
17    return (int)result;
18 }
```

On the right, the generated assembly code for the x86 msvc v19.0 compiler is shown:

```
5 int blaster_checksum(void const *,int) PROC
6     push    ebp
7     mov     ebp, esp
8     sub     esp, 8
9     mov     eax, DWORD PTR _bufv$[ebp]
10    mov    DWORD PTR _buf$[ebp], eax
11    mov    DWORD PTR _result$[ebp], 0
12 $LN2@blaster_ch:
13    cmp    DWORD PTR _length$[ebp], 1
14    jle    SHORT $LN3@blaster_ch
15    mov    ecx, DWORD PTR _buf$[ebp]
16    movzx  edx, WORD PTR [ecx]
17    add    edx, DWORD PTR _result$[ebp]
18    mov    DWORD PTR _result$[ebp], edx
19    mov    eax, DWORD PTR _buf$[ebp]
```

The assembly code includes labels like `$LN2@blaster_ch` and `$LN3@blaster_ch`, and uses registers `eax`, `ecx`, `edx`, and `ebp`.

Obsessive reverse engineering

I spend a lot of time obsessively documenting and reversing exactly what's going on. The difference between -1 and 0xFFFFFFFF is important. All my friends are lazy about it

ADD eax,-1

ADD eax,0xFFFFFFFF

BlackICE #1 - reversing Microsoft

- One of the first personal firewalls
- No APIs existed to intercept packets
- So we had to reverse engineer the kernel
 - Find any function that processes incoming packets
 - Figure out how it works, how to mess with it safely
 - Put our code in some slack area of memory
 - Insert JMP instruction, to our code, then we jump back
 - On operating system boot, hunt for the function, do our patch
 - On shutdown, remove our patch

BlackICE #2 patent

How to patch live

- Multi-CPU responding to interrupts
- Impossible to make a multibyte change without a crash
- Find one byte JMP change, to JMP back a few bytes
- Catch CPU in a infinite loop
- Put multibyte change after it to jump to our code
- Then break open the loop with one byte change
-



US008006243B2

(10) Patent No.: **US 8,006,243 B2**
(45) Date of Patent: **Aug. 23, 2011**



US007181486B1

(10) Patent No.: **US 7,181,486 B1**
(45) Date of Patent: **Feb. 20, 2007**

BlackICE #3 - WinXP SP2

They radically changed the network stack we couldn't find the function

We are on list of “must work software”

So they reverse engineered what we were doing

They stuck the pattern we looked for in the code.

Their packet-receive function jumped to, and after, jumped right back.

We found it and patched it

So we continued to work

Legal #1 reverse engineering

So yea, they caught us reverse engineering their stack.

Their lawyers made noises about it.

Their techies told 'em to shut up.

FYI: to prove we reverse engineered them, they had to reverse engineer us

So, counter-suit?

5. **LIMITATION ON REVERSE ENGINEERING, DECOMPILATION, AND DISASSEMBLY.** You may not reverse engineer, decompile, or disassemble the Product, except and only to the extent that it is expressly permitted by applicable law notwithstanding this limitation.
6. **TERMINATION.** Without prejudice to any other rights, Microsoft may cancel this EULA if you do not abide by the terms and conditions of this EULA, in which case you must destroy all copies of the Product and all of its component parts.

Legal #2 - Reverse engineer is VERY LEGAL, except...

Fundamentally, reverse engineering is legal and encouraged.

But software licenses always forbid it, so in practice, software engineering is effectively illegal.

There are various ways to get around it, such as clean room specifications and such.

But in practice, most of what we do, people don't care, as in the BlackICE example above.

Don't believe a lawyer "you can't do it", but find a good one that explains the risks if you do it.

Legal #3 - DMCA and exceptions

DMCA adds a whole new twist on it

Can't reverse for circumventing digital rights protections

But so many exceptions

Every 3 years Library of Congress issues exceptions.

One exception for cybersecurity research is regularly granted.

New exceptions for right-to-repair (e.g. John Deere) are coming

Legal #4 - user vs. agreeer

Example: Airtight, WiFi intrusion prevention system

So we threw WiFi packets at it in order to test how it responded

So they made legal threats

But we weren't the customer who agreed to the EULA. We were an outsider.

DLL/PRELOAD injection

Interact live while it's running.

Write code to walk through data structures, log out a socket/file-handle

Deterministically, parse the stack

Instead of deterministically, grep through looking for the things you want

- I.e. send packet “robertgraham” to process, have your code look for that pattern in memory, then again search elsewhere in memory for that 4 byte pointer

ROT3 (Caeser cipher) in Mike Lindell's Absolute Proof

```
public: String^ GetData(String^ val)
{
    String^ rnx = "!a-bn1poe4-kjb=87rger7rge3$##$#4675!@#$34(^%!@#$225(&%*";
    String^ key = rnx->Remove(17, 5);
    String^ decrypt = "";
    int ValIndex = Convert::ToInt32(rnx->ToCharArray()[25].ToString());
    for (int iChar = 0; iChar < val->Length; iChar++)
    {
        decrypt += (Char)(val[iChar] - ValIndex);
    }
    return decrypt;
}
```

The best reverse engineering tool

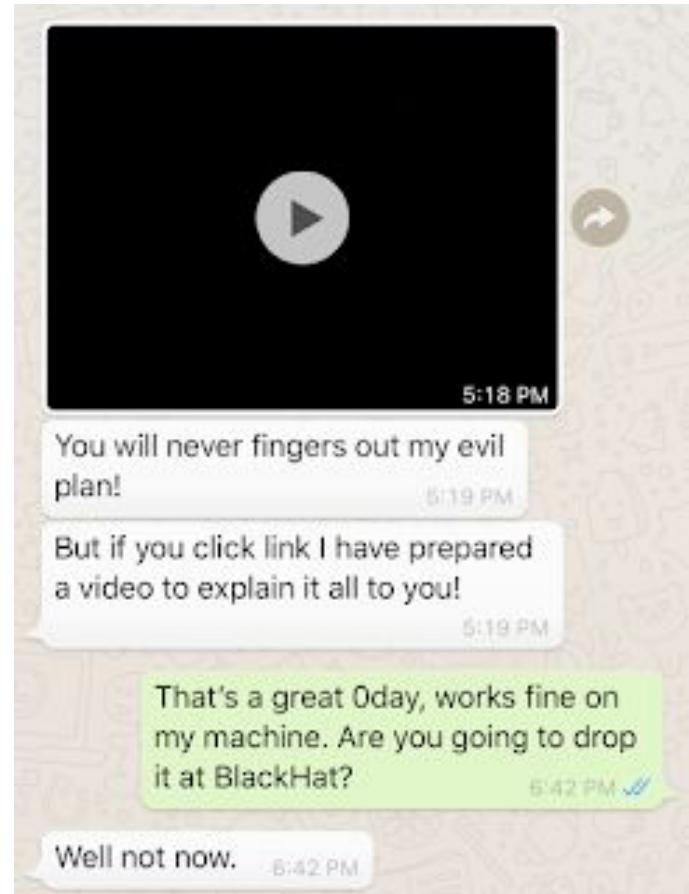
DEBUGGER

WhatsApp

Claim: Jeff Bezos security team claims the Saudi Prince hacked his phone by sending a video, encrypted by WhatsApp, containing an 0day

Analysis: how WhatsApp encrypts videos

<https://blog.erratasec.com/2020/01/how-to-decrypt-whatsapp-end-to-end.html>



WhatsApp app uses SQLite backend

The screenshot shows a terminal window and a DB Browser for SQLite interface. The terminal window displays a command to calculate the SHA1 hash of a SQLite database file, resulting in the hash: 7c7fba66680ef796b916b067077cc246adacf01d.

The DB Browser for SQLite window shows the WhatsApp database. The main table view displays two rows of data:

	ZVCARDSTRING	ZMEDIAKEY
1 Os-lq6MPSAd7lZXSU-x4BTKm.enc	video/mp4	BLOB
2 /TRMo3NCmzMpesUIYyFmEZ0IR.enc	video/mp4	BLOB

An 'Edit Database Cell' dialog is open over the second row, specifically for the ZMEDIAKEY column of the second record. The mode is set to 'Binary'. The binary data is displayed as a series of hex bytes:

```
0000 0a 20 4c a8 0d 66 c6 84 02 fb 53 cc d1 20 7c  
0010 9d e5 40 1d 9a 70 4d 51 c2 6d 37 b9 b1 30 ak  
0020 00 fc 12 20 75 27 54 f6 e8 c8 3b 7a cd 98 00  
0030 52 cf 66 81 d3 0e 88 7e 62 31 b1 bd 6d e2 79  
0040 ab 03 8a e5 18 b0 a0 b8 f1 05 20 00
```

The status bar at the bottom indicates "Type of data currently in cell: Binary" and "76 byte(s)".

Phone app to website hacking

Find with strings: <https://example.com/foo/bar>

Find in code: <https://example.com/foo/bar?id=107>

Their code:

```
"SELECT * FROM Users WHERE UserId = " + $id
```

My query: <https://example.com/foo/bar?id=107;+DROP+TABLE+customers;>

Their code:

```
"SELECT * FROM Users WHERE UserId = " + “107; DROP TABLE customers;”
```

```
"SELECT * FROM Users WHERE UserId = 107; DROP TABLE customers;"
```

Always debug malware in a virtual machine

A lot of malware does weird things on startup

Among these is unpacking

Among these is obfuscation

You'll need to run the code partway through to unpack, decrypt, and unobfuscate things, to get a memory image to the point where you can then start reading the code.

If you don't do this in a virtual machine, you'll eventually make a mistake and infect yourself and the network you are attached to.

Superfish #1

<https://blog.erratasec.com/2015/02/extracting-superfish-certificate.html>

Use ‘strings’ to grab all strings from executable, which also gets embedded certificates

```
4844 pc2g&h
4845 IhvcD
4846 Unknown error
4847 equence
4848 operation
4849 SocketAsync
4850 -----BEGIN ENCRYPTED PRIVATE KEY-----
4851 MIICxjBABgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIDHHhyAEZQoICAQgA
4852 MBQGCCqGSIB3DQMHBAlHEg+MCYQ30ASCAoDevGvFRHvtWOb5Rc0f3lbVKqeUvWSz
4853 xQn+rZELhnwb6baolmbFcsi6XkacVzL/EF7Ll4de/CSQ6pZZCCvfDzov0mPOuGve
4854 SAe7hbAcol7+JWVfzbnVTblPf0i7mwSvK61cKq7YfcKJ2os/uJGpeX9zraywlyFx
4855 f+EdTr348d0ez8uHkURyY1cvSHsIdITALkCh0onAYT68SVighTeB6xOCwfmsHx+X
4856 3Qbhom2YCIxfJiaAoz2/LndCpDaEfOrVrxXFOKXrIbmeDEyjDQj16AVni9uuaj7l
4857 NiO3zrrqxsfdVINPaAYRKQnS102jXqkH01z72c/MpMMC6dwZswF5V3R7RSXngyBn
```

Superfish #2 PEMcrack

<https://github.com/robertdavidgraham/pemcrack>

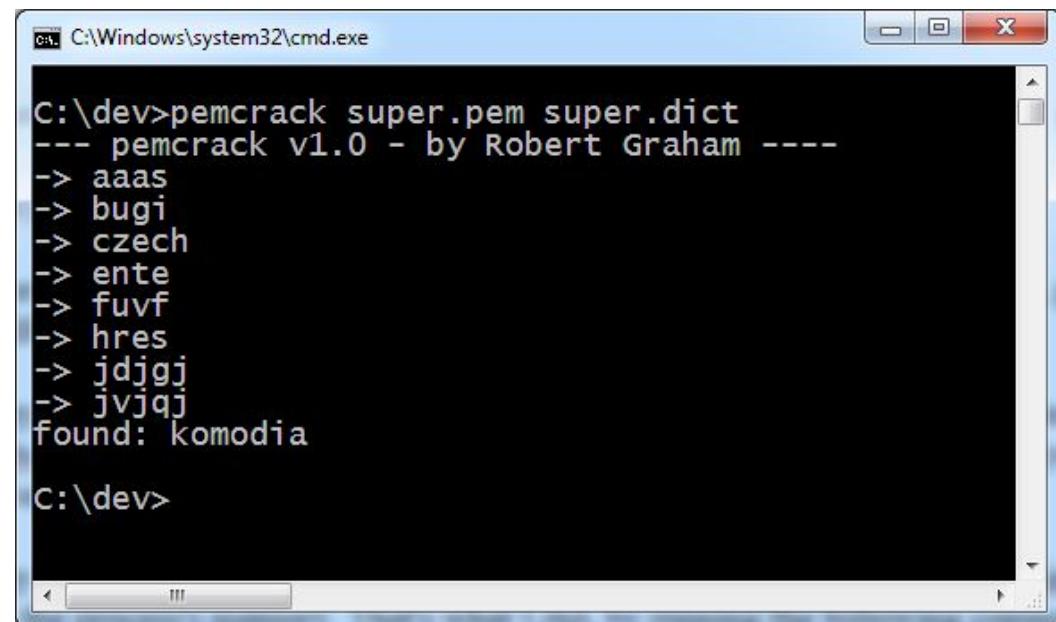
Quick and dirty tool

```
50
51     /* Gxempt to read the file using the current password */
52     p = PEM_read_PrivateKey(fp,&pkey, NULL, password);
53     if (p) {
54         /* FOUND!!! print and exit the program */
55         printf("found: %s\n", password);
56         exit(0);
57 }
```

Superfish #2

```
strings super.dmp > super.txt
```

```
grep "^[a-z]*$" super.txt | sort | uniq > super.dict
```



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:

```
C:\dev>pemcrack super.pem super.dict
--- pemcrack v1.0 - by Robert Graham ---
-> aaas
-> bugi
-> czech
-> ente
-> fuvf
-> hres
-> jdjgj
-> jvjqj
found: komodia

C:\dev>
```

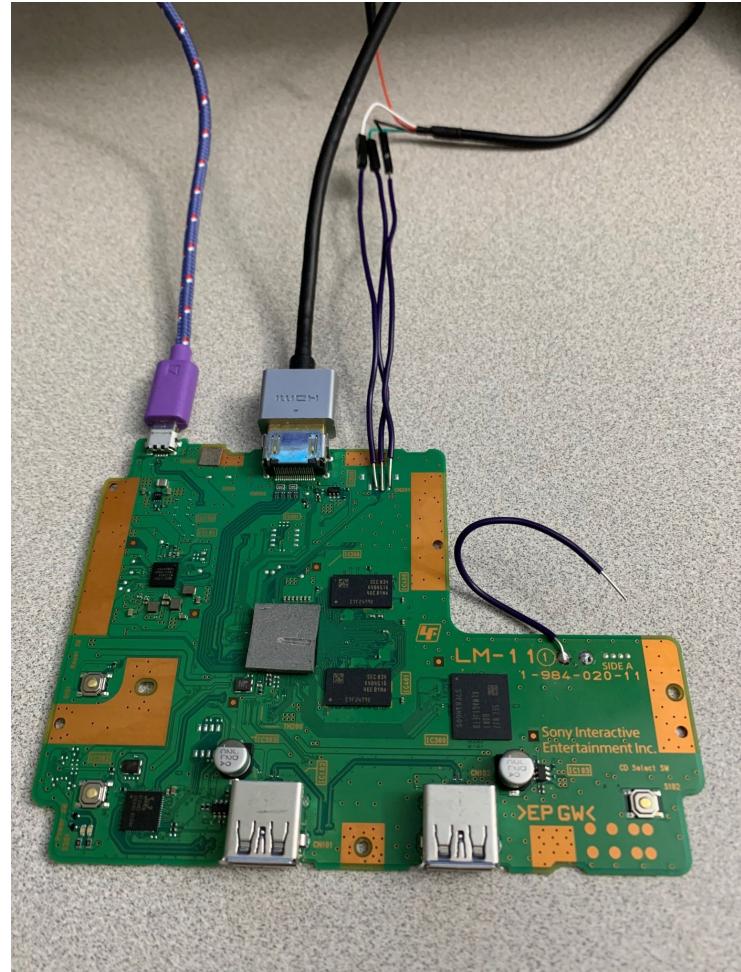
BTW: Don't be afraid to solder wires

There's a lot of fun to be had with devices.

This shows soldering on a serial port to a Playstation Classic.

Or, you'll be soldering onto JTAG.

These are overwhelmingly standard ports.



Conclusion

Every reverse engineering task started with wondering “where do I start?”

Just jump in anywhere.

The key is simply self confidence, know that the impossible will start to become possible as you build up clues.

All hacking is the foolish belief that you'll succeed at the impossible.