vxworks固件分析

前言

vxworks 的固件分析流程

```
1.用binwalk查看固件基本信息并解压固件
2.获取固件相关信息， cpu架构，大小端
3.确定固件的加载地址
4.用IDA加载固件，并修复符号表
5. 分析固件
```

实战分析

一道CTF题

分析固件

用到的例子

```
http://www.icsmaster.org/wp-content/uploads/2018/01/2018013004153995.zip
```

首先用 binwalk 扫描下固件的信息

```
$ binwalk ctf_vxworks.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
901           0x385           Zlib compressed data, default compression
```

发现就是 zlib 压缩的数据， 用 binwalk 直接解开， 然后用 binwalk 对解开后的文件扫描，发现 vxworks 关键信息， 于是拿到 vxworks 的固件。

```
$ binwalk 385

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
2054252       0x1F586C        EST flat binary
2088936       0x1FDFE8        HTML document header
2108532       0x202C74        HTML document footer
2110048       0x203260        HTML document header
2115564       0x2047EC        HTML document footer
2119528       0x205768        XML document, version: "1.0"
2119796       0x205874        XML document, version: "1.0"
2119912       0x2058E8        XML document, version: "1.0"
2192512       0x217480        Base64 standard index table
2192580       0x2174C4        Base64 standard index table
2211604       0x21BF14        VxWorks WIND kernel version "2.5"
2225264       0x21F470        Copyright string: "Copyright Wind River Systems, Inc., 1984-2000"
2321952       0x236E20        Copyright string: "copyright_wind_river"
3118988       0x2F978C        Copyright string: "Copyright, Real-Time Innovations, Inc., 1991.  All rights reserved."
3126628       0x2FB564        Copyright string: "Copyright 1984-1996 Wind River Systems, Inc."
3153524       0x301E74        VxWorks symbol table, big endian, first entry: [type: function, code address: 0x1FF058, symbol address: 0x27655C]
```

同时还扫到了符号表的位置

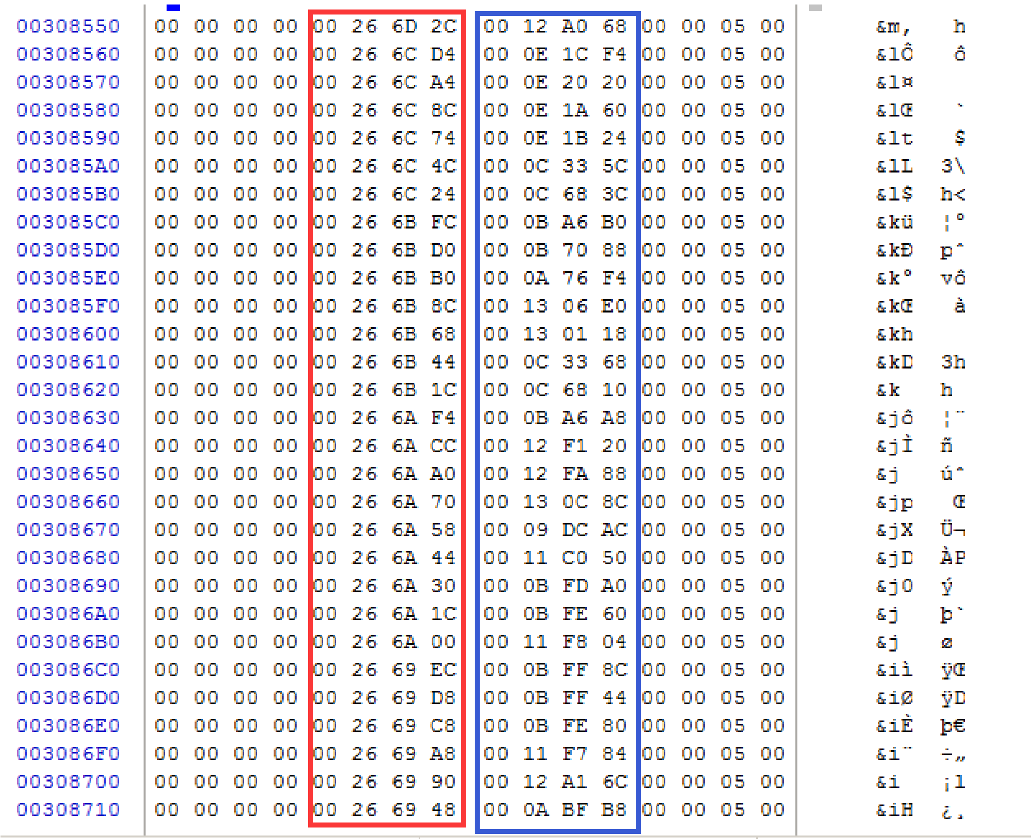
计算固件加载地址

固件加载地址的计算可以通过`vxworks`固件中的符号表计算得出的。

固件的符号表一般在固件的最后，有明显的规律。

以 **16个字节** 为一组数据:

- 4个字节 符号对应字符串的内存地址
- 4个字节 符号的内存地址
- 4个字节 特征数据
- 4个字节 `0x00`， 标识一组数据的结尾



```
00308550  00 00 00 00 00 26 6D 2C 00 12 A0 68 00 00 05 00      &m,    h
00308560  00 00 00 00 00 26 6C D4 00 0E 1C F4 00 00 05 00      &lÔ    ô
00308570  00 00 00 00 00 26 6C A4 00 0E 20 20 00 00 05 00      &l¤
00308580  00 00 00 00 00 26 6C 8C 00 0E 1A 60 00 00 05 00      &lŒ    `
00308590  00 00 00 00 00 26 6C 74 00 0E 1B 24 00 00 05 00      &lt    $
003085A0  00 00 00 00 00 26 6C 4C 00 0C 33 5C 00 00 05 00      &lL   3\
003085B0  00 00 00 00 00 26 6C 24 00 0C 68 3C 00 00 05 00      &l$   h<
003085C0  00 00 00 00 00 26 6B FC 00 0B A6 B0 00 00 05 00      &kü   ¦°
003085D0  00 00 00 00 00 26 6B D0 00 0B 70 88 00 00 05 00      &kÐ   pˆ
003085E0  00 00 00 00 00 26 6B B0 00 0A 76 F4 00 00 05 00      &k°   vô
003085F0  00 00 00 00 00 26 6B 8C 00 13 06 E0 00 00 05 00      &kŒ    à
00308600  00 00 00 00 00 26 6B 68 00 13 01 18 00 00 05 00      &kh
00308610  00 00 00 00 00 26 6B 44 00 0C 33 68 00 00 05 00      &kD   3h
00308620  00 00 00 00 00 26 6B 1C 00 0C 68 10 00 00 05 00      &k    h
00308630  00 00 00 00 00 26 6A F4 00 0B A6 A8 00 00 05 00      &jô   ¦¨
00308640  00 00 00 00 00 26 6A CC 00 12 F1 20 00 00 05 00      &jÌ   ñ
00308650  00 00 00 00 00 26 6A A0 00 12 FA 88 00 00 05 00      &j    úˆ
00308660  00 00 00 00 00 26 6A 70 00 13 0C 8C 00 00 05 00      &jp    Œ
00308670  00 00 00 00 00 26 6A 58 00 09 DC AC 00 00 05 00      &jX   Ü¬
00308680  00 00 00 00 00 26 6A 44 00 11 C0 50 00 00 05 00      &jD   ÀP
00308690  00 00 00 00 00 26 6A 30 00 0B FD A0 00 00 05 00      &j0   ý
003086A0  00 00 00 00 00 26 6A 1C 00 0B FE 60 00 00 05 00      &j    þ`
003086B0  00 00 00 00 00 26 6A 00 00 11 F8 04 00 00 05 00      &j    ø
003086C0  00 00 00 00 00 26 69 EC 00 0B FF 8C 00 00 05 00      &iì   ÿŒ
003086D0  00 00 00 00 00 26 69 D8 00 0B FF 44 00 00 05 00      &iØ   ÿD
003086E0  00 00 00 00 00 26 69 C8 00 0B FE 80 00 00 05 00      &iÈ   þ€
003086F0  00 00 00 00 00 26 69 A8 00 11 F7 84 00 00 05 00      &i¨   ÷„
00308700  00 00 00 00 00 26 69 90 00 12 A1 6C 00 00 05 00      &i    ¡l
00308710  00 00 00 00 00 26 69 48 00 0A BF B8 00 00 05 00      &iH   ¿¸
```

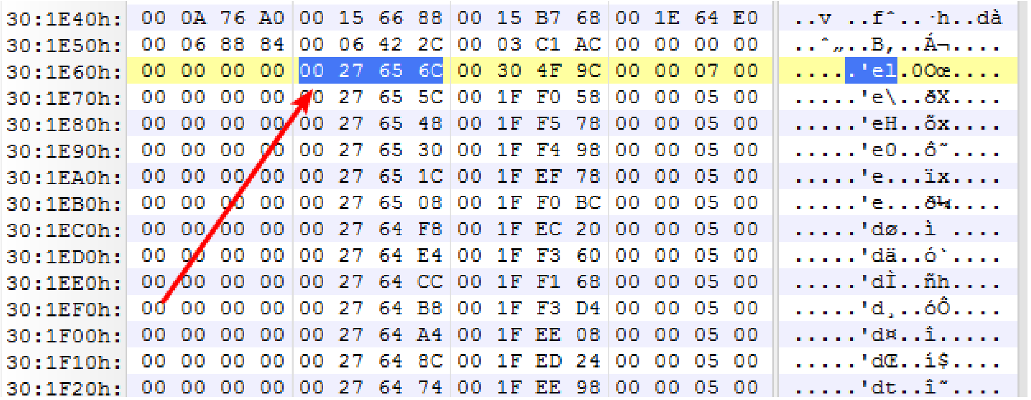
计算加载地址的方法就是

```
加载地址 = 符号表中字符串的地址 - 相应字符串在固件中的偏移
```

这里还有一个小 `tips`

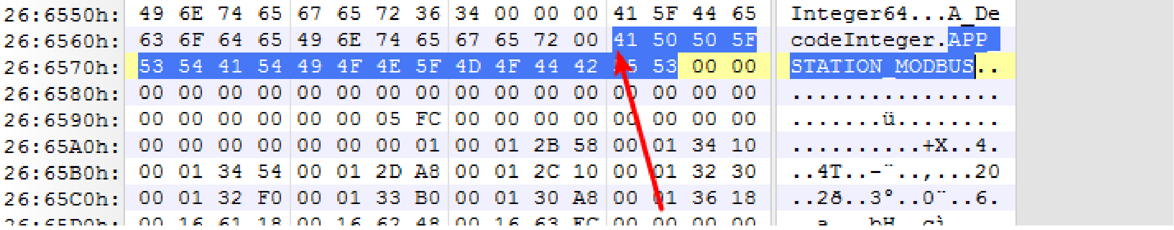
```
字符串表里面的最后一个字符串 在 符号表的第一项被引用
```

首先根据特征找到符号表, 根据 `binwalk -A` 可知固件为大端，所以 第一个符号表项对应字符串在内存中的地址为 `0x27656c`



```
30:1E40h: 00 0A 76 A0 00 15 66 88 00 15 B7 68 00 1E 64 E0  ..v ..fˆ..·h..dà
30:1E50h: 00 06 88 84 00 06 42 2C 00 03 C1 AC 00 00 00 00  ..ˆ„..B,..Á¬....
30:1E60h: 00 00 00 00 00 27 65 6C 00 30 4F 9C 00 00 07 00  .....'el.0Oœ....
30:1E70h: 00 00 00 00 00 27 65 5C 00 1F F0 58 00 00 05 00  .....'e\..ðX....
30:1E80h: 00 00 00 00 00 27 65 48 00 1F F5 78 00 00 05 00  .....'eH..õx....
30:1E90h: 00 00 00 00 00 27 65 30 00 1F F4 98 00 00 05 00  .....'e0..ô˜....
30:1EA0h: 00 00 00 00 00 27 65 1C 00 1F EF 78 00 00 05 00  .....'e...ïx....
30:1EB0h: 00 00 00 00 00 27 65 08 00 1F F0 BC 00 00 05 00  .....'e...ð¼....
30:1EC0h: 00 00 00 00 00 27 64 F8 00 1F EC 20 00 00 05 00  .....'dø..ì ....
30:1ED0h: 00 00 00 00 00 27 64 E4 00 1F F3 60 00 00 05 00  .....'dä..ó`....
30:1EE0h: 00 00 00 00 00 27 64 CC 00 1F F1 68 00 00 05 00  .....'dÌ..ñh....
30:1EF0h: 00 00 00 00 00 27 64 B8 00 1F F3 D4 00 00 05 00  .....'d¸..óÔ....
30:1F00h: 00 00 00 00 00 27 64 A4 00 1F EE 08 00 00 05 00  .....'d¤..î.....
30:1F10h: 00 00 00 00 00 27 64 8C 00 1F ED 24 00 00 05 00  .....'dŒ..í$....
30:1F20h: 00 00 00 00 00 27 64 74 00 1F EE 98 00 00 05 00  .....'dt..î˜....
```

然后找到字符串表中最后一个字符串所在偏移为 `0x26656c`



```
26:6550h: 49 6E 74 65 67 65 72 36 34 00 00 00 41 5F 44 65  Integer64...A_De
26:6560h: 63 6F 64 65 49 6E 74 65 67 65 72 00 41 50 50 5F  codeInteger.APP_
26:6570h: 53 54 41 54 49 4F 4E 5F 4D 4F 44 42 45 53 00 00  STATION_MODBUS..
26:6580h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
26:6590h: 00 00 00 00 00 00 05 FC 00 00 00 00 00 00 00 00  .......ü........
26:65A0h: 00 00 00 00 00 00 00 01 00 01 2B 58 00 01 34 10  ..........+X..4.
26:65B0h: 00 01 34 54 00 01 2D A8 00 01 2C 10 00 01 32 30  ..4T..-¨..,...20
26:65C0h: 00 01 32 F0 00 01 33 B0 00 01 30 A8 00 01 36 18  ..2ð..3°..0¨..6.
26:65D0h: 00 16 61 18 00 16 62 48 00 16 63 FC 00 00 00 00  ..a...bH..cü....
```

所以加载地址为

```
$ python -c "print hex(0x27656c - 0x26656c)"
0x10000
```

恢复符号表

拿到加载地址后，把固件用 `IDA` 加载起来，然后用 `idapython` 的脚本恢复即可、

```
from idaapi import *
from idc import *

loadaddress = 0x10000
eaStart = 0x301e64 + loadaddress
eaEnd = 0x3293a4 + loadaddress

ea = eaStart
eaEnd = eaEnd
while ea < eaEnd:
    create_strlit(Dword(ea), BADADDR)
    sName = get_strlit_contents(Dword(ea))
    print sName
    if sName:
        eaFunc = Dword(ea + 4)
        MakeName(eaFunc, sName)
        MakeCode(eaFunc)
        MakeFunction(eaFunc, BADADDR)
    ea = ea + 16
```

就是遍历符号表项，在指定位置命名 + 设置函数。

参考

http://www.icsmaster.org/archives/ics/784

http://www.freebuf.com/vuls/177036.html

TP-Link wr886v6 分析

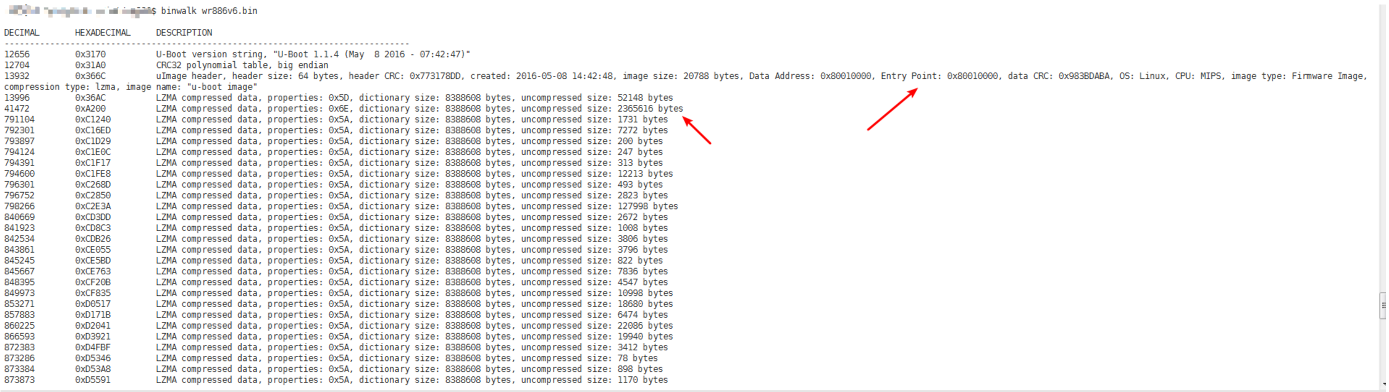
下载固件

网上可以搜到

http://www.drvsky.com/TP-Link/TL-WR886N.htm

分析固件

首先 `binwalk` 看看信息



```
[redacted]$ binwalk wr886v6.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
12656         0x3170          U-Boot version string, "U-Boot 1.1.4 (May  8 2016 - 07:42:47)"
12704         0x31A0          CRC32 polynomial table, big endian
13932         0x366C          uImage header, header size: 64 bytes, header CRC: 0x773178DD, created: 2016-05-08 14:42:48, image size: 20788 bytes, Data Address: 0x80010000, Entry Point: 0x80010000, data CRC: 0x983BDABA, OS: Linux, CPU: MIPS, image type: Firmware Image,
compression type: lzma, image name: "u-boot image"
13996         0x36AC          LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 52148 bytes
41472         0xA200          LZMA compressed data, properties: 0x6E, dictionary size: 8388608 bytes, uncompressed size: 2365616 bytes
791104        0xC1240         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 1731 bytes
792301        0xC16ED         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 7272 bytes
793897        0xC1D29         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 200 bytes
794124        0xC1E0C         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 247 bytes
794391        0xC1F17         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 313 bytes
794600        0xC1FE8         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 12213 bytes
796301        0xC268D         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 493 bytes
796752        0xC2850         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 2823 bytes
798266        0xC2E3A         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 127998 bytes
840669        0xCD3DD         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 2672 bytes
841923        0xCD8C3         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 1008 bytes
842534        0xCDB26         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 3806 bytes
843861        0xCE055         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 3796 bytes
845245        0xCE5BD         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 822 bytes
845667        0xCE763         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 7836 bytes
848395        0xCF20B         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 4547 bytes
849973        0xCF835         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 10998 bytes
853271        0xD0517         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 18680 bytes
857883        0xD171B         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 6474 bytes
860225        0xD2041         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 22086 bytes
866593        0xD3921         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 19940 bytes
872383        0xD4FBF         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 3412 bytes
873286        0xD5346         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 78 bytes
873384        0xD53A8         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 898 bytes
873873        0xD5591         LZMA compressed data, properties: 0x5A, dictionary size: 8388608 bytes, uncompressed size: 1170 bytes
```

这里 `binwalk` 只识别到了 `u-boot` 镜像，和一大堆 `lzma` 压缩的数据。通过 `u-boot` 的信息可以知道固件的加载地址为 `0x80010000`

发现其中有个 `lzma` 压缩的数据大小和其他的不在一个数量级， 把它拿出来解析。

```
41472         0xA200          LZMA compressed data, properties: 0x6E, dictionary size: 8388608 bytes, uncompressed size: 2365616 bytes
```

然后用 `dd` 把它 拿出来

```
dd if=wr886v6.bin of=large.lzma bs=1 skip=41472 count=749632
```

然后用 `binwalk` 解出来

```
$ binwalk _large.lzma.extracted/0
0     0.7z
$ binwalk _large.lzma.extracted/0

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
1846464       0x1C2CC0        Certificate in DER format (x509 v3), header length: 4, sequence length: 4
1853752       0x1C4938        Certificate in DER format (x509 v3), header length: 4, sequence length: 4
1899120       0x1CFA70        VxWorks operating system version "5.5.1" , compiled: "Oct 20 2017, 16:17:22"
1968188       0x1E083C        Copyright string: "Copyright(C) 2001-2011 by TP-LINK TECHNOLOGIES CO., LTD."
1997876       0x1E7C34        VxWorks WIND kernel version "2.6"
2042936       0x1F2C38        HTML document header
2043001       0x1F2C79        HTML document footer
2062828       0x1F79EC        PEM certificate
2062884       0x1F7A24        PEM RSA private key
2072188       0x1F9E7C        Base64 standard index table
2107248       0x202770        CRC32 polynomial table, big endian
2108272       0x202B70        CRC32 polynomial table, big endian
2109296       0x202F70        CRC32 polynomial table, big endian
2110320       0x203370        CRC32 polynomial table, big endian
2130920       0x2083E8        XML document, version: "1.0"
2150332       0x20CFBC        SHA256 hash constants, big endian
2248421       0x224EE5        StuffIt Deluxe Segment (data): f
2248452       0x224F04        StuffIt Deluxe Segment (data): fError
2248533       0x224F55        StuffIt Deluxe Segment (data): f
```

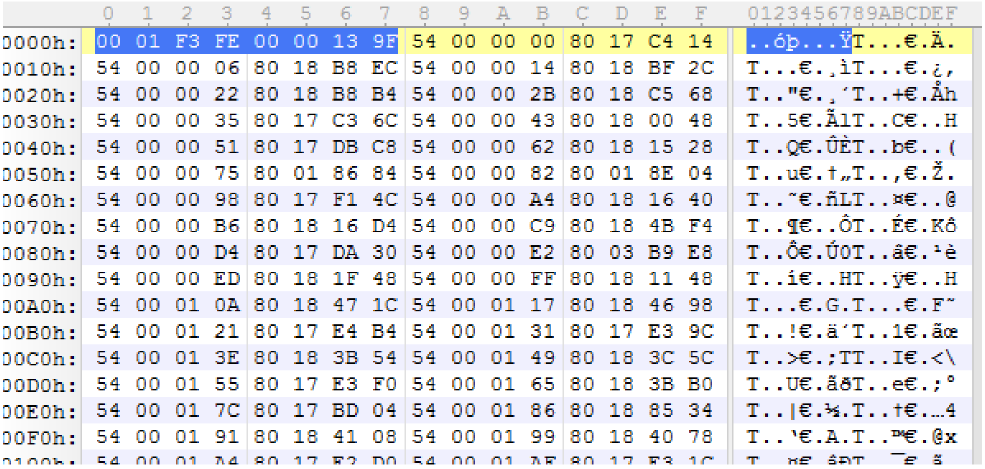
可以看到有 `vxworks` 的字符串，前面已经拿到了固件的基地址，用 `ida` 加载。

恢复符号表

使用 `grep` 来找符号表

```
$ binwalk  -Me wr886v6.bin
........
$ cd _wr886v6.bin.extracted/
$ grep -r bzero
Binary file C2E3A matches
```

然后打开 C2E3A 看看



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 00 01 F3 FE 00 00 13 9F 54 00 00 00 80 17 C4 14  ..óþ...ŸT...€.Ä.
0010h: 54 00 00 06 80 18 B8 EC 54 00 00 14 80 18 BF 2C  T...€.¸ìT...€.¿,
0020h: 54 00 00 22 80 18 B8 B4 54 00 00 2B 80 18 C5 68  T.."€.¸´T..+€.Åh
0030h: 54 00 00 35 80 17 C3 6C 54 00 00 43 80 18 00 48  T..5€.ÃlT..C€..H
0040h: 54 00 00 51 80 17 DB C8 54 00 00 62 80 18 15 28  T..Q€.ÛÈT..b€..(
0050h: 54 00 00 75 80 01 86 84 54 00 00 82 80 01 8E 04  T..u€.†„T..‚€.Ž.
0060h: 54 00 00 98 80 17 F1 4C 54 00 00 A4 80 18 16 40  T..˜€.ñLT..¤€..@
0070h: 54 00 00 B6 80 18 16 D4 54 00 00 C9 80 18 4B F4  T..¶€..ÔT..É€.Kô
0080h: 54 00 00 D4 80 17 DA 30 54 00 00 E2 80 03 B9 E8  T..Ô€.Ú0T..â€.¹è
0090h: 54 00 00 ED 80 18 1F 48 54 00 00 FF 80 18 11 48  T..í€..HT..ÿ€..H
00A0h: 54 00 01 0A 80 18 47 1C 54 00 01 17 80 18 46 98  T...€.G.T...€.F˜
00B0h: 54 00 01 21 80 17 E4 B4 54 00 01 31 80 17 E3 9C  T..!€.ä´T..1€.ãœ
00C0h: 54 00 01 3E 80 18 3B 54 54 00 01 49 80 18 3C 5C  T..>€.;TT..I€.<\
00D0h: 54 00 01 55 80 17 E3 F0 54 00 01 65 80 18 3B B0  T..U€.ãðT..e€.;°
00E0h: 54 00 01 7C 80 17 BD 04 54 00 01 86 80 18 85 34  T..|€.½.T..†€.…4
00F0h: 54 00 01 91 80 18 41 08 54 00 01 99 80 18 40 78  T..‘€.A.T..™€.@x
0100h: 54 00 01 A4 80 17 F2 D0 54 00 01 AF 80 17 F3 1C  T..¤€.òÐT..¯€.ó.
```

开头 8 字节表示文件大小和符号表大小， 然后就是符号表了， 0x9d00 为字符串表的位置。

然后用脚本恢复

```
# coding=utf-8
import idc
import idaapi
import idautils
sym_file = open(″PATH OF SYM FILE″, ’rb’).read()
table_data = sym_file[0x08:0x9f80]
print(table_data[-8:].encode(’hex’))
string_table = sym_file[0x9f80:]
def get_string(offset):
    string = ″″
    while True:
        if string_table[offset] != ’\x00’:
            string += string_table[offset]
            offset += 1
        else:
            break
    return string
def get_sym_data():
    sym_data = []
    for offset in range(0, len(table_data), 8):
        table = table_data[offset: offset + 8]
        flag = table[0]
        # print(’flag: %s’ % flag)
        string_offset = int(table[1:4].encode(’hex’), 16)
        # print(’string_offset: %s’ % string_offset)
        string = get_string(string_offset)
        # print(’string: %s’ % string)
        target_address = int(table[-4:].encode(’hex’), 16)
        # print(’target_address: %s’ % hex(target_address))
        sym_data.append([flag, string, target_address])
    return sym_data
def fix_idb(sym_data):
    for sym in sym_data:
        flag, string, target_address = sym
        idc.MakeName(target_address, string)
        if flag == ’\x54’:
            print(″Start fix Function %s at %s″ % (string, hex(target_address)))
            idc.MakeCode(target_address)
            idc.MakeFunction(target_address, idc.BADADDR)
            # print(’flag: %s’ % flag)
            # print(’string: %s’ % string)
            # print(’target_address: %s’ % hex(target_address))
if __name__ == ’__main__’:
    sym_data = get_sym_data()
    fix_idb(sym_data)
```

参考

https://www.secpulse.com/archives/75635.html

来源： https://www.cnblogs.com/hac425/p/9706815.html