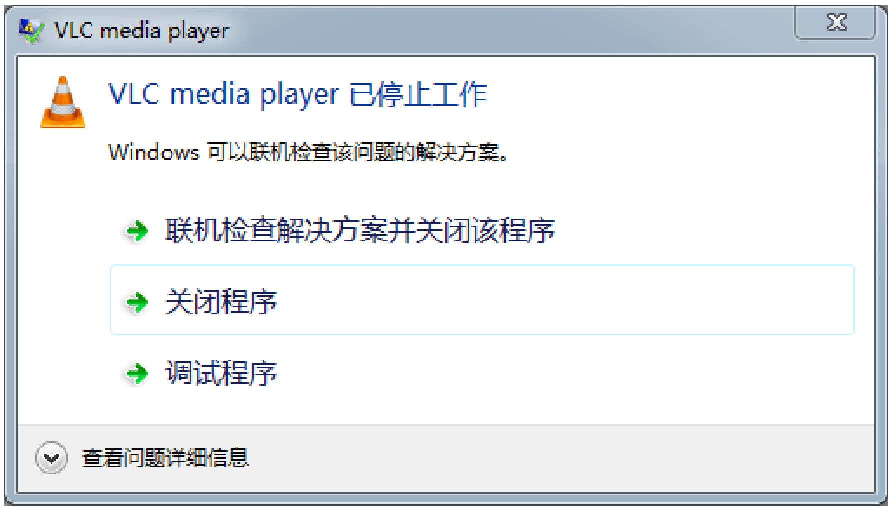
VLC windows 2.2.7 Heap Overflow causing Out-of-Bound Memory Write

Overview

I have found a vulnerability of VLC media player 2.2.7 for windows which is caused by out-of-bound heap memory write in calling memcpy(). The root cause is that function ffmpeg_CopyPicture() takes wrong size as the loop limit, which causes writing over the allocated heap memory. The vulnerability can cause Denial-of-Service and maybe further cause code execution by overwriting the next heap structure.

Exception


VLC media player

VLC media player 已停止工作

Windows 可以联机检查该问题的解决方案。

联机检查解决方案并关闭该程序

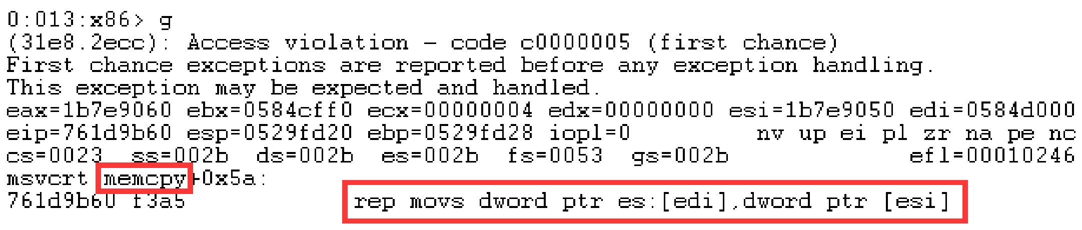
关闭程序

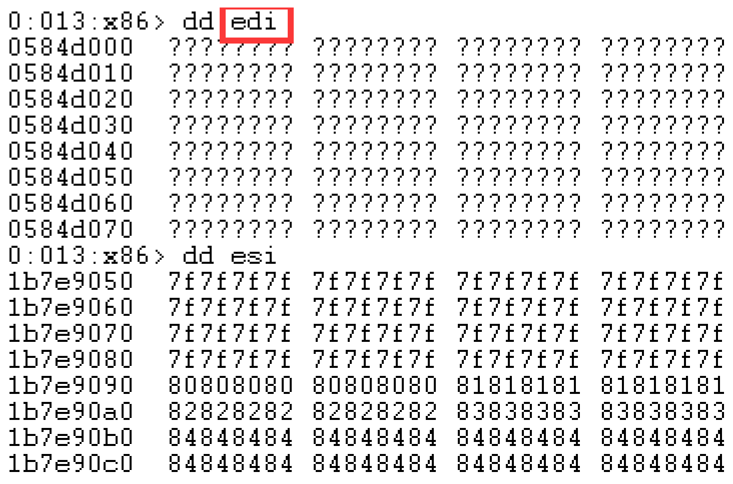
调试程序

查看问题详细信息

Analysis

Crash scene:

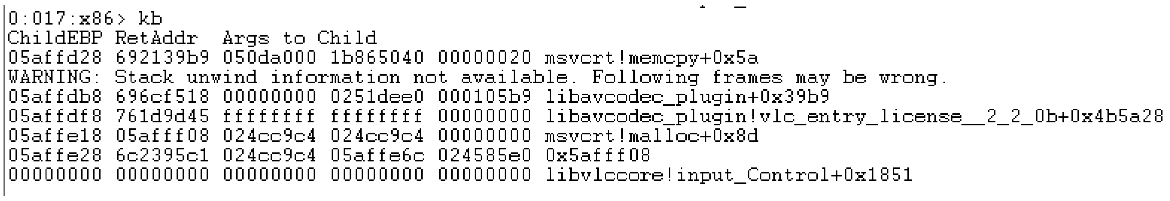

```
0:013:x86> g
(31e8.2ecc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=1b7e9060 ebx=0584cff0 ecx=00000004 edx=00000000 esi=1b7e9050 edi=0584d000
eip=761d9b60 esp=0529fd20 ebp=0529fd28 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
msvcrt!memcpy+0x5a:
761d9b60 f3a5            rep movs dword ptr es:[edi],dword ptr [esi]
```



```
0:013:x86> dd edi
0584d000  ???????? ???????? ???????? ????????
0584d010  ???????? ???????? ???????? ????????
0584d020  ???????? ???????? ???????? ????????
0584d030  ???????? ???????? ???????? ????????
0584d040  ???????? ???????? ???????? ????????
0584d050  ???????? ???????? ???????? ????????
0584d060  ???????? ???????? ???????? ????????
0584d070  ???????? ???????? ???????? ????????
0:013:x86> dd esi
1b7e9050  7f7f7f7f 7f7f7f7f 7f7f7f7f 7f7f7f7f
1b7e9060  7f7f7f7f 7f7f7f7f 7f7f7f7f 7f7f7f7f
1b7e9070  7f7f7f7f 7f7f7f7f 7f7f7f7f 7f7f7f7f
1b7e9080  7f7f7f7f 7f7f7f7f 7f7f7f7f 7f7f7f7f
1b7e9090  80808080 80808080 81818181 81818181
1b7e90a0  82828282 82828282 83838383 83838383
1b7e90b0  84848484 84848484 84848484 84848484
1b7e90c0  84848484 84848484 84848484 84848484
```

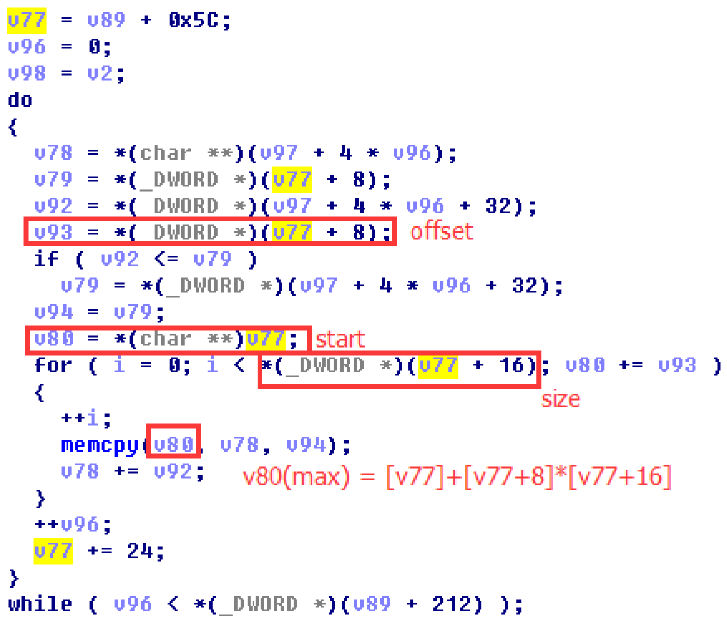
Crash happens in memcpy() where the dst_addr is inaccessible.

Crash Stack:



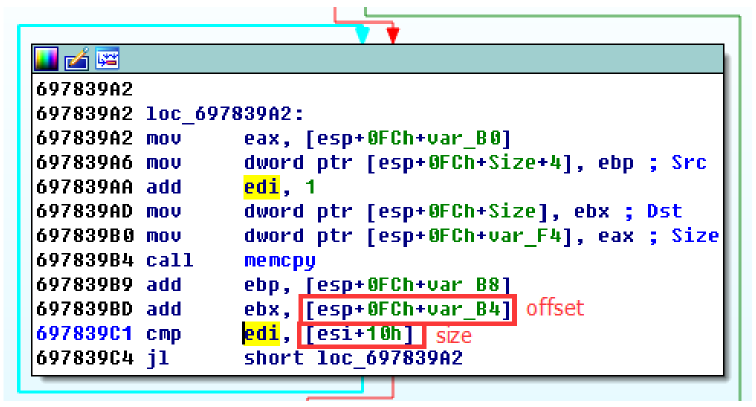
```
0:017:x86> kb
ChildEBP RetAddr  Args to Child
05affd28 692139b9 050da000 1b865040 00000020 msvcrt!memcpy+0x5a
WARNING: Stack unwind information not available. Following frames may be wrong.
05affdb8 696cf518 00000000 0251dee0 000105b9 libavcodec_plugin+0x39b9
05affdf8 761d9d45 ffffffff ffffffff 00000000 libavcodec_plugin!vlc_entry_license__2_2_0b+0x4b5a28
05affe18 05afff08 024cc9c4 024cc9c4 00000000 msvcrt!malloc+0x8d
05affe28 6c2395c1 024cc9c4 05affe6c 024585e0 0x5afff08
00000000 00000000 00000000 00000000 00000000 libvlccore!input_Control+0x1851
```

Look at the crash function using IDA:

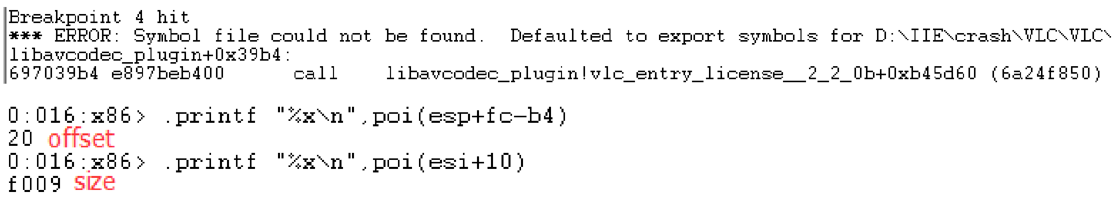


```
v77 = v89 + 0x5C;
v96 = 0;
v98 = v2;
do
{
  v78 = *(char **)(v97 + 4 * v96);
  v79 = *(_DWORD *)(v77 + 8);
  v92 = *(_DWORD *)(v97 + 4 * v96 + 32);
  v93 = *(_DWORD *)(v77 + 8); offset
  if ( v92 <= v79 )
    v79 = *(_DWORD *)(v97 + 4 * v96 + 32);
  v94 = v79;
  v80 = *(char **)v77; start
  for ( i = 0; i < *(_DWORD *)(v77 + 16); v80 += v93 )
  {                                    size
    ++i;
    memcpy(v80, v78, v94);
    v78 += v92;    v80(max) = [v77]+[v77+8]*[v77+16]
  }
  ++v96;
  v77 += 24;
}
while ( v96 < *(_DWORD *)(v89 + 212) );
```

There are two loops and in the inner loop, the dst of memcpy is v80, whose maximum value is [v77] + [v77+8]*[v77+16]. So, the cause may be that the dst addr is so large to extend the allocated space. We debug to prove this assumption.

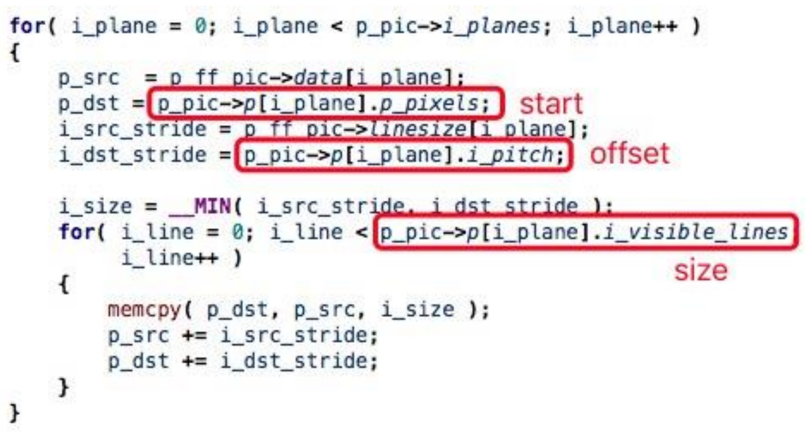


```
697839A2
697839A2 loc_697839A2:
697839A2 mov     eax, [esp+0FCh+var_B0]
697839A6 mov     dword ptr [esp+0FCh+Size+4], ebp ; Src
697839AA add     edi, 1
697839AD mov     dword ptr [esp+0FCh+Size], ebx ; Dst
697839B0 mov     dword ptr [esp+0FCh+var_F4], eax ; Size
697839B4 call    memcpy
697839B9 add     ebp, [esp+0FCh+var_B8]
697839BD add     ebx, [esp+0FCh+var_B4] offset
697839C1 cmp     edi, [esi+10h] size
697839C4 jl      short loc_697839A2
```



```
Breakpoint 4 hit
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for D:\IIE\crash\VLC\VLC\
libavcodec_plugin+0x39b4:
697039b4 e897beb400      call    libavcodec_plugin!vlc_entry_license__2_2_0b+0xb45d60 (6a24f850)

0:016:x86> .printf "%x\n",poi(esp+fc-b4)
20 offset
0:016:x86> .printf "%x\n",poi(esi+10)
f009 size
```

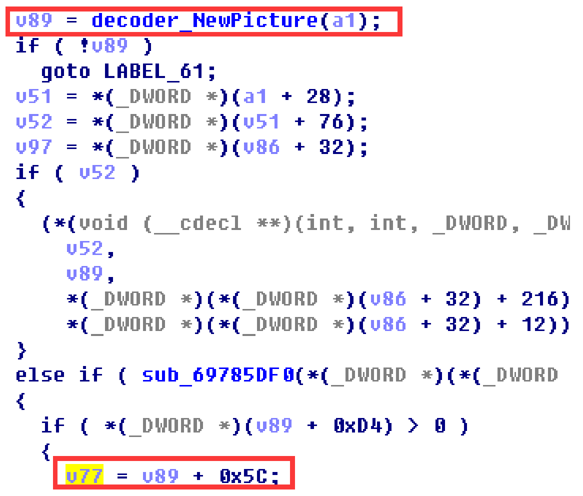
We can see that the ***size*** **(*(v77+16))** is relatively large. Have a look at the source code:



```
for( i_plane = 0; i_plane < p_pic->i_planes; i_plane++ )
{
    p_src  = p_ff_pic->data[i_plane];
    p_dst = p_pic->p[i_plane].p_pixels;  start
    i_src_stride = p_ff_pic->linesize[i_plane];
    i_dst_stride = p_pic->p[i_plane].i_pitch;  offset

    i_size = __MIN( i_src_stride, i_dst_stride );
    for( i_line = 0; i_line < p_pic->p[i_plane].i_visible_lines
         i_line++ )                                    size
    {
        memcpy( p_dst, p_src, i_size );
        p_src += i_src_stride;
        p_dst += i_dst_stride;
    }
}
```

The root cause should be the ***i_visible_lines*** attribute of p_pic->p[i].

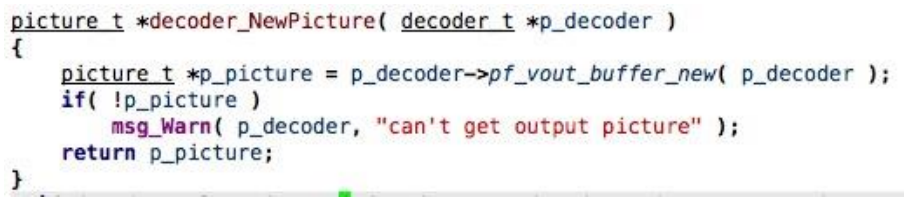
Now we need to figure out where *(v77+16) is set.



```
v89 = decoder_NewPicture(a1);
if ( !v89 )
  goto LABEL_61;
v51 = *(_DWORD *)(a1 + 28);
v52 = *(_DWORD *)(v51 + 76);
v97 = *(_DWORD *)(v86 + 32);
if ( v52 )
{
  (*(void (__cdecl **)(int, int, _DWORD, _DW
    v52,
    v89,
    *(_DWORD *)(*(_DWORD *)(v86 + 32) + 216)
    *(_DWORD *)(*(_DWORD *)(v86 + 32) + 12))
}
else if ( sub_69785DF0(*(_DWORD *)(*(_DWORD
{
  if ( *(_DWORD *)(v89 + 0xD4) > 0 )
  {
    v77 = v89 + 0x5C;
```

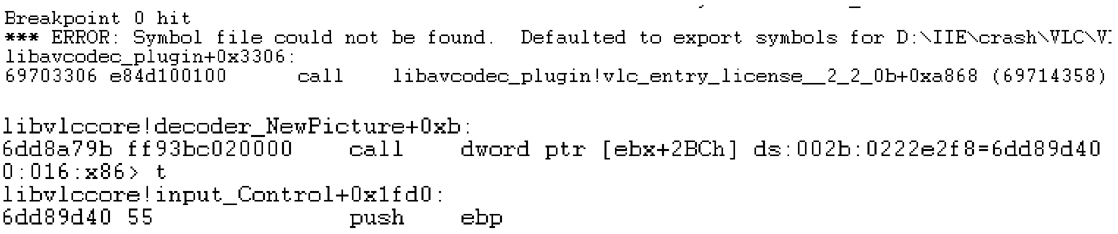
v77= v89+0x5c and v89 is from decoder_NewPicture().

Next, I will use source code analysis and debug to check how decoder_NewPicture() set **v89**.



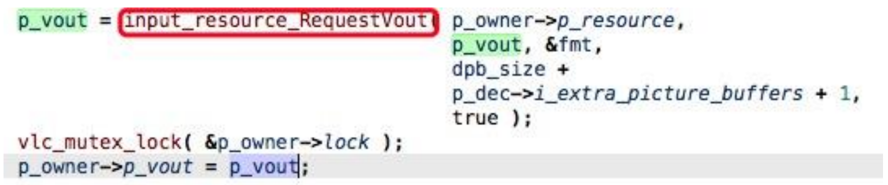
```
picture_t *decoder_NewPicture( decoder_t *p_decoder )
{
    picture_t *p_picture = p_decoder->pf_vout_buffer_new( p_decoder );
    if( !p_picture )
        msg_Warn( p_decoder, "can't get output picture" );
    return p_picture;
}
```

Set a breakpoint on *call **decoder_NewPicture()*** and step into this function, then we can come to libvlccore!input_Control+0x1fd0, which is p_decoder->pf_vout_buffer_new.

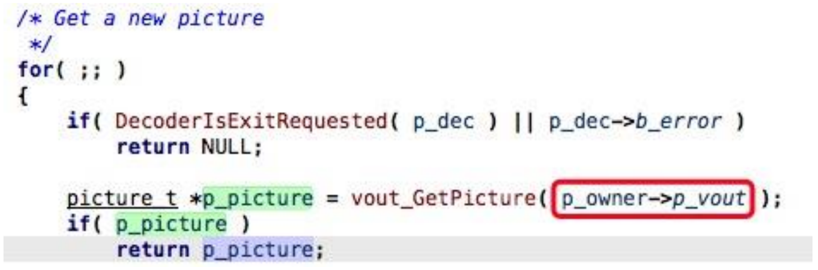


```
Breakpoint 0 hit
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for D:\IIE\crash\VLC\V
libavcodec_plugin+0x3306:
69703306 e84d100100      call    libavcodec_plugin!vlc_entry_license__2_2_0b+0xa868 (69714358)

libvlccore!decoder_NewPicture+0xb:
6dd8a79b ff93bc020000    call    dword ptr [ebx+2BCh] ds:002b:0222e2f8=6dd89d40
0:016:x86> t
libvlccore!input_Control+0x1fd0:
6dd89d40 55              push    ebp
```

Searching by strings used in this function (e.g. failed to create video output), we can know this function corresponds to ***vout_new_buffer()*** in source code.



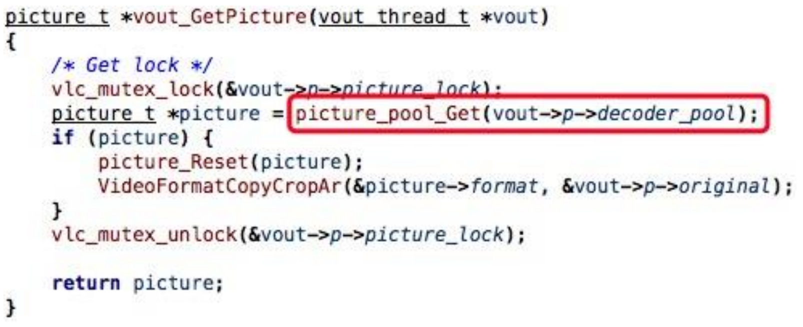
```
p_vout = input_resource_RequestVout( p_owner->p_resource,
                                     p_vout, &fmt,
                                     dpb_size +
                                     p_dec->i_extra_picture_buffers + 1,
                                     true );
vlc_mutex_lock( &p_owner->lock );
p_owner->p_vout = p_vout;
```



```
/* Get a new picture
 */
for( ;; )
{
    if( DecoderIsExitRequested( p_dec ) || p_dec->b_error )
        return NULL;

    picture_t *p_picture = vout_GetPicture( p_owner->p_vout );
    if( p_picture )
        return p_picture;
```

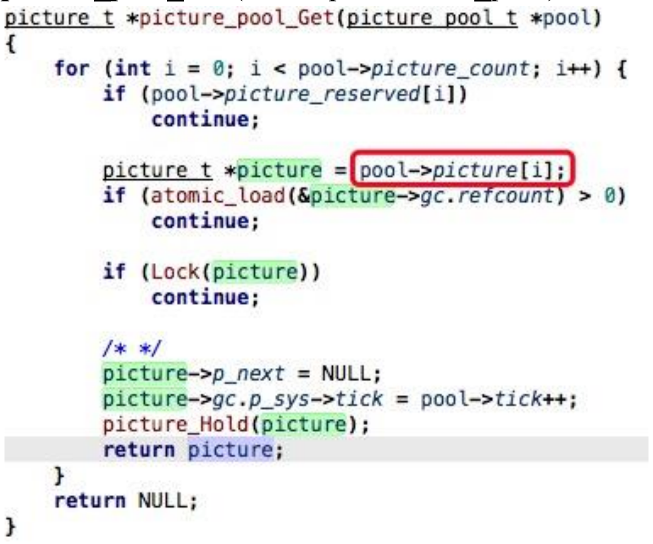
v89 = vout_GetPicture(p_owner->p_vout) and p_vout is set by input_resource_RequestVout().

First follow in ***vout_GetPicture()***,



```
picture_t *vout_GetPicture(vout_thread_t *vout)
{
    /* Get lock */
    vlc_mutex_lock(&vout->p->picture_lock);
    picture_t *picture = picture_pool_Get(vout->p->decoder_pool);
    if (picture) {
        picture_Reset(picture);
        VideoFormatCopyCropAr(&picture->format, &vout->p->original);
    }
    vlc_mutex_unlock(&vout->p->picture_lock);

    return picture;
}
```

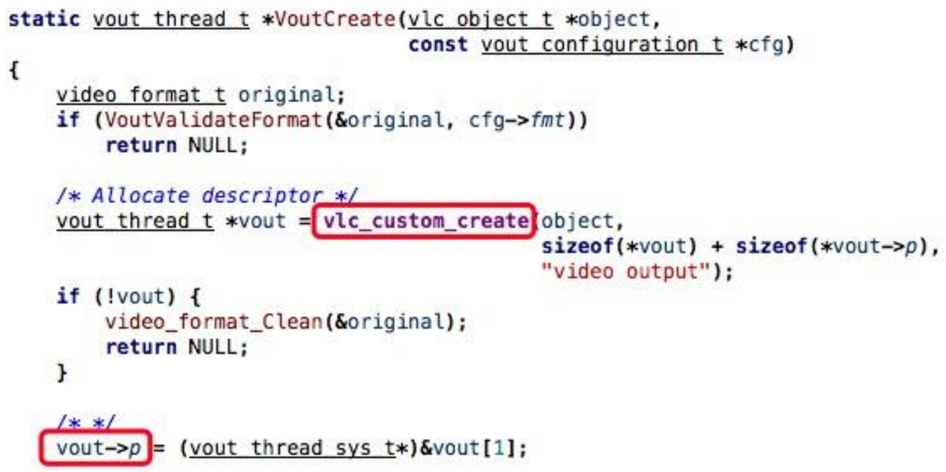
picture_pool_Get(vout->p->decoder_pool):



```
picture_t *picture_pool_Get(picture_pool_t *pool)
{
    for (int i = 0; i < pool->picture_count; i++) {
        if (pool->picture_reserved[i])
            continue;

        picture_t *picture = pool->picture[i];
        if (atomic_load(&picture->gc.refcount) > 0)
            continue;

        if (Lock(picture))
            continue;

        /* */
        picture->p_next = NULL;
        picture->gc.p_sys->tick = pool->tick++;
        picture_Hold(picture);
        return picture;
    }
    return NULL;
}
```

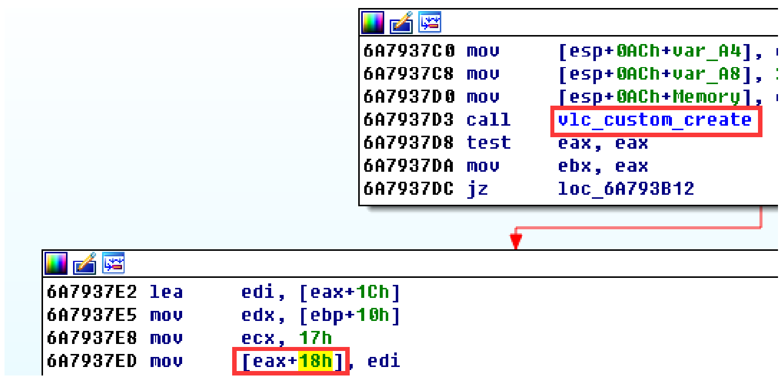
From above we know **v89 = p_vout->p->decoder_pool->picture[i]** where ***i*** satisfies some constraints.

Then need to know where and how p_vout is set.
Follow in ***input_resource_RequestVout()***:
input_resource_RequestVout() -> RequestVout() -> vout_Request() -> VoutCreate():



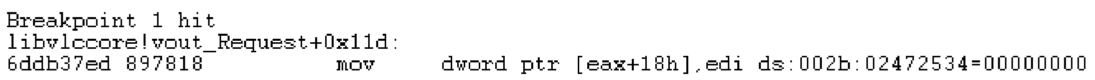
```
static vout_thread_t *VoutCreate(vlc_object_t *object,
                                 const vout_configuration_t *cfg)
{
    video_format_t original;
    if (VoutValidateFormat(&original, cfg->fmt))
        return NULL;

    /* Allocate descriptor */
    vout_thread_t *vout = vlc_custom_create(object,
                                            sizeof(*vout) + sizeof(*vout->p),
                                            "video output");
    if (!vout) {
        video_format_Clean(&original);
        return NULL;
    }

    /* */
    vout->p = (vout_thread_sys_t*)&vout[1];
```

In VoutCreate(), vout->p is set. We set a breakpoint here to check vout->p.



```
6A7937C0 mov     [esp+0ACh+var_A4],
6A7937C8 mov     [esp+0ACh+var_A8],
6A7937D0 mov     [esp+0ACh+Memory],
6A7937D3 call    vlc_custom_create
6A7937D8 test    eax, eax
6A7937DA mov     ebx, eax
6A7937DC jz      loc_6A793B12
```

```
6A7937E2 lea     edi, [eax+1Ch]
6A7937E5 mov     edx, [ebp+10h]
6A7937E8 mov     ecx, 17h
6A7937ED mov     [eax+18h], edi
```



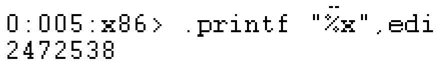
```
Breakpoint 1 hit
libvlccore!vout_Request+0x11d:
6ddb37ed 897818          mov     dword ptr [eax+18h],edi ds:002b:02472534=00000000
```

Here:

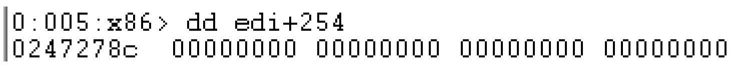
vout->p = *(eax+0x18) = edi

vout->p->decoder_pool = *(*(eax+0x18)+0x254)

vout->p->decoder_pool->picture = *(*(*(eax+0x18)+0x254)+0x14)



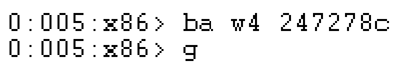
```
0:005:x86> .printf "%x",edi
2472538
```



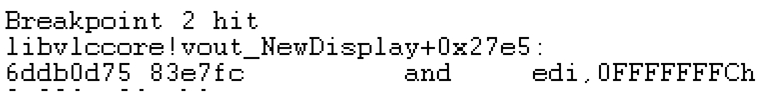
```
0:005:x86> dd edi+254
0247278c  00000000 00000000 00000000 00000000
```

So, after vout->p=&vout[1],   vout->p->decoder_pool = 0

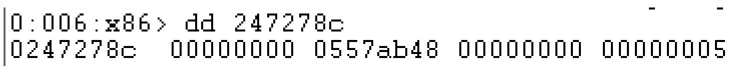
We set a write breakpoint on 0x247278c to check where vout->p->decoder_pool is set:



```
0:005:x86> ba w4 247278c
0:005:x86> g
```



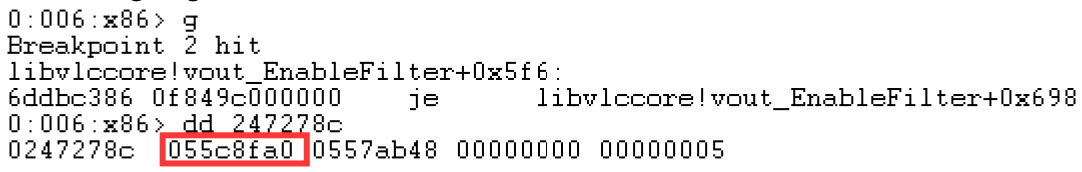
```
Breakpoint 2 hit
libvlccore!vout_NewDisplay+0x27e5:
6ddb0d75 83e7fc          and     edi,0FFFFFFFCh
```



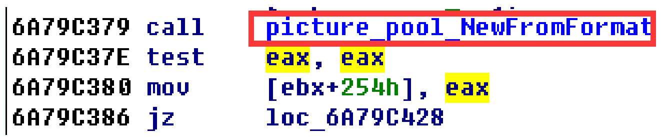
```
0:006:x86> dd 247278c
0247278c  00000000 0557ab48 00000000 00000005
```

After first hit the write breakpoint, it is still zero.

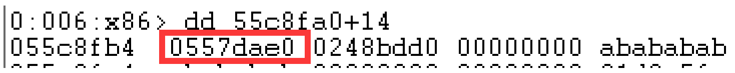
Continue going until it is not zero:



```
0:006:x86> g
Breakpoint 2 hit
libvlccore!vout_EnableFilter+0x5f6:
6ddbc386 0f849c000000    je      libvlccore!vout_EnableFilter+0x698
0:006:x86> dd 247278c
0247278c  055c8fa0 0557ab48 00000000 00000005
```



```
6A79C379 call    picture_pool_NewFromFormat
6A79C37E test    eax, eax
6A79C380 mov     [ebx+254h], eax
6A79C386 jz      loc_6A79C428
```



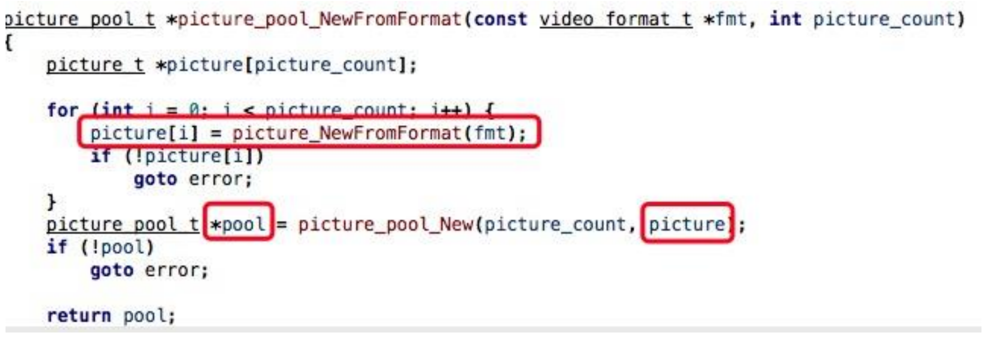
```
0:006:x86> dd 55c8fa0+14
055c8fb4  0557dae0 0248bdd0 00000000 abababab
```

Now:

vout->p->decoder_pool = picture_pool_NewFromFormat() = 0x55c8fa0

vout->p->decoder_pool->picture = *(0x55c8fa0+0x14) = 0x557dae0

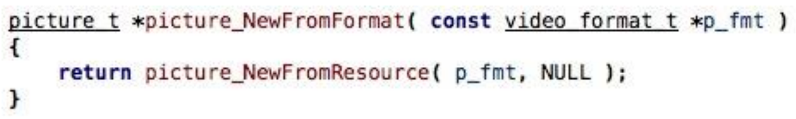
But we need to know how vout->p->decoder_pool is set, so let's follow in ***picture_pool_NewFromFormat()***:



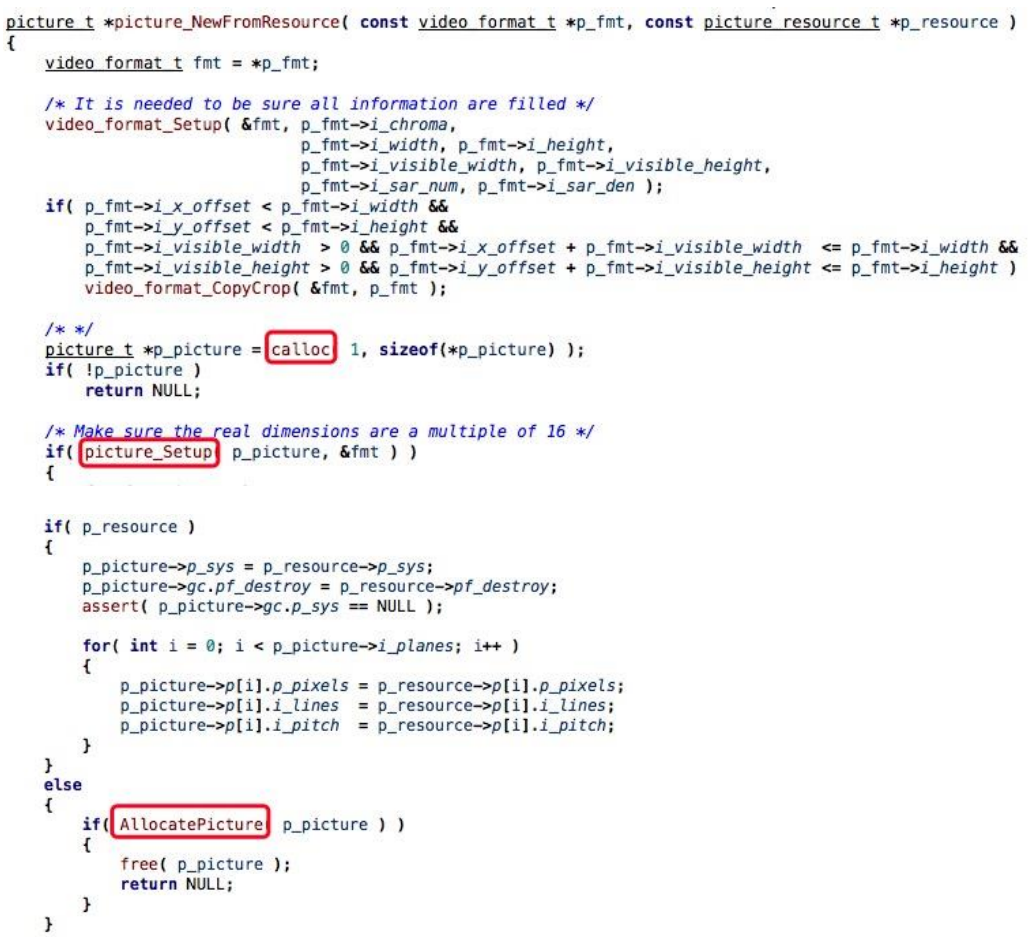
```
picture_pool_t *picture_pool_NewFromFormat(const video_format_t *fmt, int picture_count)
{
    picture_t *picture[picture_count];

    for (int i = 0; i < picture_count; i++) {
        picture[i] = picture_NewFromFormat(fmt);
        if (!picture[i])
            goto error;
    }
    picture_pool_t *pool = picture_pool_New(picture_count, picture);
    if (!pool)
        goto error;

    return pool;
```

For each picture, call picture_NewFromFormat() to initialize and connect picture with pool->picture.
Don't forget out target value:
v89 = p_vout->p->decoder_pool->picture[i] = pic
size = *(v77+16) = *((v89+0x5c)+0x10) = pic->p[0].i_visible_lines

So we need to see how each picture is set (determining how v89 is set):

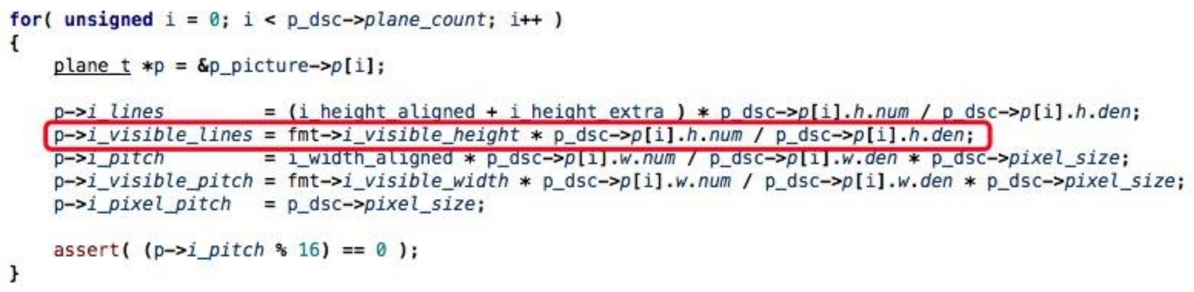


```
picture_t *picture_NewFromFormat( const video_format_t *p_fmt )
{
    return picture_NewFromResource( p_fmt, NULL );
}
```



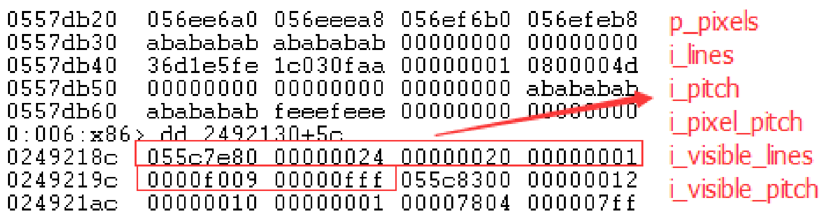
```
picture_t *picture_NewFromResource( const video_format_t *p_fmt, const picture_resource_t *p_resource )
{
    video_format_t fmt = *p_fmt;

    /* It is needed to be sure all information are filled */
    video_format_Setup( &fmt, p_fmt->i_chroma,
                              p_fmt->i_width, p_fmt->i_height,
                              p_fmt->i_visible_width, p_fmt->i_visible_height,
                              p_fmt->i_sar_num, p_fmt->i_sar_den );
    if( p_fmt->i_x_offset < p_fmt->i_width &&
        p_fmt->i_y_offset < p_fmt->i_height &&
        p_fmt->i_visible_width  > 0 && p_fmt->i_x_offset + p_fmt->i_visible_width  <= p_fmt->i_width &&
        p_fmt->i_visible_height > 0 && p_fmt->i_y_offset + p_fmt->i_visible_height <= p_fmt->i_height )
        video_format_CopyCrop( &fmt, p_fmt );

    /* */
    picture_t *p_picture = calloc( 1, sizeof(*p_picture) );
    if( !p_picture )
        return NULL;

    /* Make sure the real dimensions are a multiple of 16 */
    if( picture_Setup( p_picture, &fmt ) )
    {

    if( p_resource )
    {
        p_picture->p_sys = p_resource->p_sys;
        p_picture->gc.pf_destroy = p_resource->pf_destroy;
        assert( p_picture->gc.p_sys == NULL );

        for( int i = 0; i < p_picture->i_planes; i++ )
        {
            p_picture->p[i].p_pixels = p_resource->p[i].p_pixels;
            p_picture->p[i].i_lines  = p_resource->p[i].i_lines;
            p_picture->p[i].i_pitch  = p_resource->p[i].i_pitch;
        }
    }
    else
    {
        if( AllocatePicture( p_picture ) )
        {
            free( p_picture );
            return NULL;
        }
    }
```

first call calloc() to allocate space for each picture to store struct picture_t;
then call ***picture_Setup()*** to initialize attributes of struct picture_t;



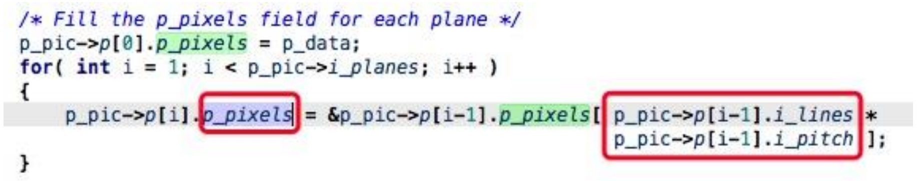
```
for( unsigned i = 0; i < p_dsc->plane_count; i++ )
{
    plane_t *p = &p_picture->p[i];

    p->i_lines         = (i_height_aligned + i_height_extra ) * p_dsc->p[i].h.num / p_dsc->p[i].h.den;
    p->i_visible_lines = fmt->i_visible_height * p_dsc->p[i].h.num / p_dsc->p[i].h.den;
    p->i_pitch         = i_width_aligned * p_dsc->p[i].w.num / p_dsc->p[i].w.den * p_dsc->pixel_size;
    p->i_visible_pitch = fmt->i_visible_width * p_dsc->p[i].w.num / p_dsc->p[i].w.den * p_dsc->pixel_size;
    p->i_pixel_pitch   = p_dsc->pixel_size;

    assert( (p->i_pitch % 16) == 0 );
}
```

Set all the attributes the same value for each plane in each picture, and here i_visible_lines=0xf009.



```
0557db20  056ee6a0 056eeea8 056ef6b0 056efeb8
0557db30  abababab abababab 00000000 00000000
0557db40  36d1e5fe 1c030faa 00000001 0800004d
0557db50  00000000 00000000 00000000 abababab
0557db60  abababab feeefeee 00000000 00000000
0:006:x86> dd 2492130+5c
0249218c  055c7e80 00000024 00000020 00000001
0249219c  0000f009 00000fff 055c8300 00000012
024921ac  00000010 00000001 00007804 000007ff
```

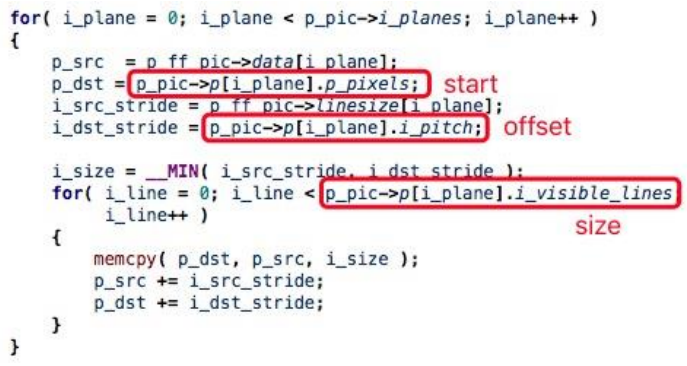
finally call ***AllocatePicture()*** to allocate space to store pixel data in picture.



```
/* Fill the p_pixels field for each plane */
p_pic->p[0].p_pixels = p_data;
for( int i = 1; i < p_pic->i_planes; i++ )
{
    p_pic->p[i].p_pixels = &p_pic->p[i-1].p_pixels[ p_pic->p[i-1].i_lines *
                                                    p_pic->p[i-1].i_pitch ];
}
```

Set p_pixels points to allocated data space and each plane takes space of i_lines*i_pitch.

So, back to the crash point where copying data to p_pixels with offset=i_pitch and size=i_visible_lines. Now it is obvious where the bug is and how to fix it. The size to copy should be i_lines(=0x24) not i_visible_lines(=0xf009), which is so large as to extend the allocated heap memory.



```
for( i_plane = 0; i_plane < p_pic->i_planes; i_plane++ )
{
    p_src  = p_ff_pic->data[i_plane];
    p_dst = p_pic->p[i_plane].p_pixels;
    i_src_stride = p_ff_pic->linesize[i_plane];
    i_dst_stride = p_pic->p[i_plane].i_pitch;

    i_size = __MIN( i_src_stride, i_dst_stride );
    for( i_line = 0; i_line < p_pic->p[i_plane].i_visible_lines
         i_line++ )
    {
        memcpy( p_dst, p_src, i_size );
        p_src += i_src_stride;
        p_dst += i_dst_stride;
    }
}
```

Author

Name: Jiaqi Peng,   Bingchang Liu of VARAS@IIE
Organization: IIE (http://iie.ac.cn/)