```
rax=000001efd30a4001 rbx=0000000000000000 rcx=000000000000001c
rdx=ffffffffffffffff rsi=00007ffbfcd1632b rdi=000000eeb8cfeea0
rip=00007ffbfcbdf96e rsp=000000eeb8cfee70 rbp=0000000000000000
 r8=0000000000001c00  r9=000000000000001c r10=000001efd30a7fa0
r11=000001efd30a7fd8 r12=0000000000000000 r13=0000000000000000
r14=0000000000000000 r15=0000000000000000
iopl=0         nv up ei ng nz na po cy
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010287
freerdp2!planar_skip_plane_rle+0x9e:
00007ffb`fcbdf96e 0fb600          movzx   eax,byte ptr [rax] ds:000001ef`d30a4001=??
```

Out-of-bounds access

locate the original location:

```
static INLINE INT32 planar_skip_plane_rle(const BYTE* pSrcData, UINT32 SrcSize,
        UINT32 nWidth, UINT32 nHeight)
{
    UINT32 x, y;
    BYTE controlByte;
    const BYTE* pRLE = pSrcData;
    const BYTE* pEnd = &pSrcData[SrcSize];

    for (y = 0; y < nHeight; y++)
    {
        for (x = 0; x < nWidth;)
        {
            int cRawBytes;
            int nRunLength;

            if (pRLE >= pEnd)
                return -1;

            controlByte = *pRLE++;
            nRunLength = PLANAR_CONTROL_BYTE_RUN_LENGTH(controlByte);
            cRawBytes = PLANAR_CONTROL_BYTE_RAW_BYTES(controlByte);
```

This is a bitmap-related function, pSrcData represents the cached data, and SrcSize represents the size. View the parent function that called the function:

```
rleSizes[0] = planar_skip_plane_rle(planes[0], SrcSize - (planes[0] - pSrcData),
                                    rawWidths[0], rawHeights[0]); /* RedPlane */
```

There is already a problem here, integer overflow. The second parameter is an unsigned integer, but it may be negative here, causing the parameter SrcSize in the subfunction to be very large. However, the cause of this exception is not here yet.

View the origin of planes [0]:

```
if (!bitmap->Decompress(context, bitmap,
                        cacheBitmapV2->bitmapDataStream,
                        cacheBitmapV2->bitmapWidth,
                        cacheBitmapV2->bitmapHeight,
                        cacheBitmapV2->bitmapBpp,
                        cacheBitmapV2->bitmapLength,
                        cacheBitmapV2->compressed
                        RDP_CODEC_ID_NONE))
```

Planes [0] is bitmapDataStream, and SrcSize is bitmapLength.

Look at the history of cacheBitmapV2::

```
static CACHE_BITMAP_V2_ORDER* update_read_cache_bitmap_v2_order(rdpUpdate* update, wStream* s,
        BOOL compressed, UINT16 flags)
```

This function initializes a variable of type CACHE_BITMAP_V2_ORDER。

Details of initialization::

```
if (!update_read_4byte_unsigned(s, &cache_bitmap_v2->bitmapLength)
```

Read the bitmapLength variable here:

```
cache_bitmap_v2->bitmapDataStream = malloc(cache_bitmap_v2->bitmapLength);
```

Then use this bitmapLength to malloc a heap block.

The problem is that when the read bitmapLength is 0, the pointer points to the end of the heap block, and then in the function that triggered the exception, the heap block is read, and the read is traversed, resulting in an out-of-bounds access.