POC3 analysis:

Vulnerability type: Out of bounds copy

interleaved-> TempSize is 16384, which is used to store the temporary memory of the decompressed bitmap.

```
if (interleaved)
{
    interleaved->TempSize = 64 * 64 * 4;
    interleaved->TempBuffer = _aligned_malloc(interleaved->TempSize, 16);

    if (!interleaved->TempBuffer)
    {
        free(interleaved);
        WLog_ERR(TAG, "_aligned_malloc failed!");
        return NULL;
    }

    interleaved->bts = Stream_New(NULL, interleaved->TempSize);
```

Out-of-bounds functions occur when RLEDECOMPRESS decompresses the bitmap

```
*/
static INLINE BOOL RLEDECOMPRESS(const BYTE* pbSrcBuffer, UINT32 cbSrcBuffer,
                                 BYTE* pbDestBuffer,
                                 UINT32 rowDelta, UINT32 width, UINT32 height)
{
    const BYTE* pbSrc = pbSrcBuffer;
    const BYTE* pbEnd = pbSrcBuffer + cbSrcBuffer;
    const BYTE* pbDestEnd = pbDestBuffer + rowDelta * height;
    BYTE* pbDest = pbDestBuffer;
    PIXEL temp;
    PIXEL fgPel = WHITE_PIXEL;
    BOOL fInsertFgPel = FALSE;
    BOOL fFirstLine = TRUE;
    BYTE bitmask;
    PIXEL pixelA, pixelB;
```

The main reason is that when judging the stored decompressed data again, due to the misjudgment of unsigned data, an excessive range of data was decompressed and copied.

The function RLEDECOMPRESS is calculated every time during the decompression

runLength = ExtractRunLength (code, pbSrc, & advance)

The key is to use the function ENSURE_CAPACITY to determine the boundary range of the data stored in memory.

if (! ENSURE_CAPACITY (pbDest, pbDestEnd, runLength))

return FALSE;

Then the function ENSURE_CAPACITY performs the operation

typedef unsigned __int64 size_t; is unsigned integer data, the result is available only positive integers, leading to the judgment is always true

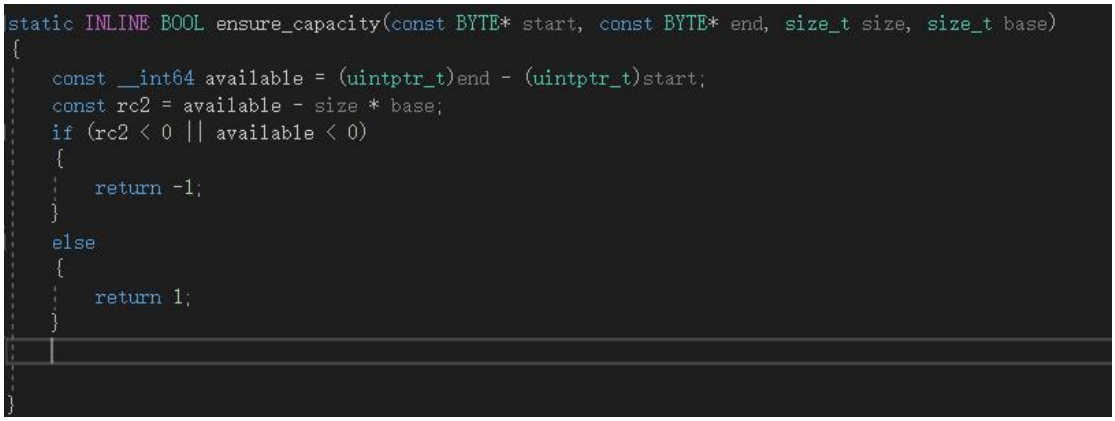
```
static INLINE BOOL ensure_capacity(const BYTE* start, const BYTE* end, size_t size, size_t base)
{
```

```
    const size_t available = (uintptr_t)end - (uintptr_t)start;
    const BOOL rc = available >= size * base;
    return rc;
}
```

When the next decompressed data copy is performed, the data size is greater than interleaved->TempSize and the overflow occurs.

After adding your own manual patch, the program runs normally and the repair is successful

The patch is as follows:



```
static INLINE BOOL ensure_capacity(const BYTE* start, const BYTE* end, size_t size, size_t base)
{
    const __int64 available = (uintptr_t)end - (uintptr_t)start;
    const rc2 = available - size * base;
    if (rc2 < 0 || available < 0)
    {
        return -1;
    }
    else
    {
        return 1;
    }

}
```