Integer overflow 1:

There is an integer overflow at Freerdp Surface

```
 *
 * @return 0 on success, otherwise a Win32 error code
 */
static UINT gdi_CreateSurface(RdpgfxClientContext* context,
                              const RDPGFX_CREATE_SURFACE_PDU* createSurface)
{
    UINT rc = ERROR_INTERNAL_ERROR;
    gdiGfxSurface* surface;
    rdpGdi* gdi = (rdpGdi*)context->custom;
    EnterCriticalSection(&context->mux);
    surface = (gdiGfxSurface*)calloc(1, sizeof(gdiGfxSurface));

    if (!surface)
        goto fail;

    surface->codecs = gdi->context->codecs;

    if (!surface->codecs)
    {
        free(surface);
        goto fail;
    }

    surface->surfaceId = createSurface->surfaceId;
    surface->width = gfx_align_scanline(createSurface->width, 16);
    surface->height = gfx_align_scanline(createSurface->height, 16);
    surface->mappedWidth = createSurface->width;
    surface->mappedHeight = createSurface->height;
    surface->outputTargetWidth = createSurface->width;
    surface->outputTargetHeight = createSurface->height;
```

Gdi_createsurface will allocate a memory of height size

```
surface->scanline = gfx_align_scanline(surface->width * 4, 16);
surface->data = (BYTE*)_aligned_malloc(surface->scanline * surface->height, 16);
```

When parsing surface data, the size of height will be reset and its length is not checked

```
 */
static UINT gdi_SurfaceToSurface(RdpgfxClientContext* context,
                                 const RDPGFX_SURFACE_TO_SURFACE_PDU* surfaceToSurface)
{
    UINT status = ERROR_INTERNAL_ERROR;
    UINT16 index;
    BOOL sameSurface;
    UINT32 nWidth, nHeight;
    const RECTANGLE_16* rectSrc;
    RDPGFX_POINT16* destPt;
    RECTANGLE_16 invalidRect;
    gdiGfxSurface* surfaceSrc;
    gdiGfxSurface* surfaceDst;
    rdpGdi* gdi = (rdpGdi*)context->custom;
    EnterCriticalSection(&context->mux);
    rectSrc = &(surfaceToSurface->rectSrc);
    surfaceSrc = (gdiGfxSurface*)context->GetSurfaceData(context, surfaceToSurface->surfaceIdSrc);
    sameSurface =
        (surfaceToSurface->surfaceIdSrc == surfaceToSurface->surfaceIdDest) ? TRUE : FALSE;

    if (!sameSurface)
        surfaceDst =
            (gdiGfxSurface*)context->GetSurfaceData(context, surfaceToSurface->surfaceIdDest);
    else
```

Also gdi_CacheToSurface

```
1232  static UINT gdi_CacheToSurface(RdpgfxClientContext* context,
1233                                 const RDPGFX_CACHE_TO_SURFACE_PDU* cacheToSurface)
1234  {
1235      UINT status = ERROR_INTERNAL_ERROR;
1236      UINT16 index;
1237      RDPGFX_POINT16* destPt;
1238      gdiGfxSurface* surface;
1239      gdiGfxCacheEntry* cacheEntry;
1240      RECTANGLE_16 invalidRect;
1241      rdpGdi* gdi = (rdpGdi*)context->custom;
1242      EnterCriticalSection(&context->mux);
1243      surface = (gdiGfxSurface*)context->GetSurfaceData(context, cacheToSurface->surfaceId);
1244      cacheEntry = (gdiGfxCacheEntry*)context->GetCacheSlotData(context, cacheToSurface->cacheSlot);
1245
1246      if (!surface || !cacheEntry)
1247          goto fail;
1248
1249      for (index = 0; index < cacheToSurface->destPtsCount; index++)
1250      {
1251          destPt = &cacheToSurface->destPts[index];
1252
1253          if (!freerdp_image_copy(surface->data, surface->format, surface->scanline, destPt->x,
1254                                  destPt->y, cacheEntry->width, cacheEntry->height, cacheEntry->data,
1255                                  cacheEntry->format, cacheEntry->scanline, 0, 0, NULL,
```

When freerdp_image_copy is called, the copy will cause overflow

```
660                 }
661             }
662             /* Copy right */
663             else if (nXSrc < nXDst)
664             {
665                 for (y = (INT32)nHeight - 1; y >= 0; y--)
666                 {
667                     const BYTE* srcLine =
668                         &pSrcData[(y + nYSrc) * nSrcStep * srcVMultiplier + srcVOffset];
669                     BYTE* dstLine = &pDstData[(y + nYDst) * nDstStep * dstVMultiplier + dstVOffset];
670                     memmove(&dstLine[xDstOffset], &srcLine[xSrcOffset], copyDstWidth); 已用时间<=1ms
671                 }
672             }
```

100 %

局部变量

名称	值	类型
copyDstWidth	108	const unsigned int
dstByte	4	const unsigned int
DstFormat	537168008	unsigned long
dstLine	0x0000018da9f34080 <读取字符串字符时出错。>	unsigned char *
dstVMultiplier	1	int
dstVOffset	0	unsigned int
flags	0	unsigned int
nDstStep	4096	unsigned int
nHeight	41992	unsigned int
nSrcStep	4096	unsigned int
nWidth	27	unsigned int

Integer overflow 2:

```
 * @return 0 on success, otherwise a Win32 error code
 */
static UINT rdpgfx_recv_solid_fill_pdu(RDPGFX_CHANNEL_CALLBACK* callback, wStream* s)
{
    UINT16 index;
    RECTANGLE_16* fillRect;
    RDPGFX_SOLID_FILL_PDU pdu;
    RDPGFX_PLUGIN* gfx = (RDPGFX_PLUGIN*)callback->plugin;
    RdpgfxClientContext* context = (RdpgfxClientContext*)gfx->iface.pInterface;
    UINT error;

    if (Stream_GetRemainingLength(s) < 8)
    {
        WLog_Print(gfx->log, WLOG_ERROR, "not enough data!");
        return ERROR_INVALID_DATA;
    }

    Stream_Read_UINT16(s, pdu.surfaceId); /* surfaceId (2 bytes) */

    if ((error = rdpgfx_read_color32(s, &(pdu.fillPixel)))) /* fillPixel (4 bytes) */
    {
        WLog_Print(gfx->log, WLOG_ERROR, "rdpgfx_read_color32 failed with error %" PRIu32 "!",
                   error);
        return error;
    }

    Stream_Read_UINT16(s, pdu.fillRectCount); /* fillRectCount (2 bytes) */
```

if rect->bottom < rect->top

```
173 UINT rdpgfx_read_rect16(wStream* s, RECTANGLE_16* rect16)
174 {
175     if (Stream_GetRemainingLength(s) < 8)
176     {
177         WLog_ERR(TAG, "not enough data!");
178         return ERROR_INVALID_DATA;
179     }
180
181     Stream_Read_UINT16(s, rect16->left);   /* left (2 bytes) */  已用时间 <= 1ms
182     Stream_Read_UINT16(s, rect16->top);    /* top (2 bytes) */
183     Stream_Read_UINT16(s, rect16->right);  /* right (2 bytes) */
184     Stream_Read_UINT16(s, rect16->bottom); /* bottom (2 bytes) */
185     return CHANNEL_RC_OK;
186 }
```

```
1042  static UINT gdi_SolidFill(RdpgfxClientContext* context, const RDPGFX_SOLID_FILL_PDU* solidFill)
1043  {
1044      UINT status = ERROR_INTERNAL_ERROR;
1045      UINT16 index;
1046      UINT32 color;
1047      BYTE a, r, g, b;
1048      UINT32 nWidth, nHeight;
1049      RECTANGLE_16* rect;
1050      gdiGfxSurface* surface;
1051      RECTANGLE_16 invalidRect;
1052      rdpGdi* gdi = (rdpGdi*)context->custom;
1053      EnterCriticalSection(&context->mux);
1054      surface = (gdiGfxSurface*)context->GetSurfaceData(context, solidFill->surfaceId);
1055
1056      if (!surface)
1057          goto fail;
1058
1059      b = solidFill->fillPixel.B;
1060      g = solidFill->fillPixel.G;
1061      r = solidFill->fillPixel.R;
1062      /* a = solidFill->fillPixel.XA;
1063       * Ignore alpha channel, this is a solid fill. */
1064      a = 0xFF;
1065      color = FreeRDPGetColor(surface->format, r, g, b, a);
1066
1067      for (index = 0; index < solidFill->fillRectCount; index++)
1068      {
1069          rect = &(solidFill->fillRects[index]);
1070          nWidth = rect->right - rect->left;
```

```
722   BOOL freerdp_image_fill(BYTE* pDstData, DWORD DstFormat, UINT32 nDstStep, UINT32 nXDst,
723                           UINT32 nYDst, UINT32 nWidth, UINT32 nHeight, UINT32 color)
724   {
725       UINT32 x, y;
726       const UINT32 bpp = GetBytesPerPixel(DstFormat);
727       BYTE* pFirstDstLine = &pDstData[nYDst * nDstStep];
728       BYTE* pFirstDstLineXOffset = &pFirstDstLine[nXDst * bpp];
729
730       for (x = 0; x < nWidth; x++)
731       {
732           BYTE* pDst = &pFirstDstLine[(x + nXDst) * bpp];
733           WriteColor(pDst, DstFormat, color);
734       }
735
736       for (y = 1; y < nHeight; y++)
737       {
738           BYTE* pDstLine = &pDstData[(y + nYDst) * nDstStep + nXDst * bpp];
739           memcpy(pDstLine, pFirstDstLineXOffset, nWidth * bpp);
740       }
741
742       return TRUE;
```