# New 15 Vuln For FreeRDP

## int overflow in winpr_image_bitmap_read_buffer

it first read data to **bf** and **bi**.
**bf.bfOffBits and bi.biSizeImage** are UINT32.
**bf.bfOffBits + bi.biSizeImage** could int overflow, then lead oob

```
static int winpr_image_bitmap_read_buffer(wImage* image, const BYTE*
buffer, size_t size)
{
    // read data from stream
    if (!readBitmapFileHeader(s, &bf) || !readBitmapInfoHeader(s, &bi))
        goto fail;
    if ((bf.bfType[0] != 'B') || (bf.bfType[1] != 'M'))
        goto fail;
    image->type = WINPR_IMAGE_BITMAP;
    // int overflow, bypass check.
    if (Stream_Capacity(s) < bf.bfOffBits + bi.biSizeImage)
        goto fail;
```

## int overflow in urb_control_descriptor_request

it first read 4 byte to **OutputBufferSize**, then **36 + OutputBufferSize** could **int overflow**.
Then it could lead **oob read and write later**.

```
    Stream_Read_UINT16(s, langId);
    Stream_Read_UINT32(s, OutputBufferSize); // read size
    if (transferDir == USBD_TRANSFER_DIRECTION_OUT)
    {
        if (Stream_GetRemainingLength(s) < OutputBufferSize)
            return ERROR_INVALID_DATA;
    }
    out_size = 36 + OutputBufferSize;  // int overflow
    out = Stream_New(NULL, out_size);  // alloc small memory
    if (!out)
        return ERROR_OUTOFMEMORY;
    Stream_Seek(out, 36);
    bmRequestType = func_recipient;
    switch (transferDir)
```

```
    {
        case USBD_TRANSFER_DIRECTION_IN:
            bmRequestType |= 0x80;
            break;
        case USBD_TRANSFER_DIRECTION_OUT:
            bmRequestType |= 0x00;
            Stream_Copy(s, out, OutputBufferSize);  // oob write.
```

## int overflow in urb_control_feature_request

it first read 4 byte to **OutputBufferSize**, then **36 + OutputBufferSize** could **int overflow**. Then it could lead **oob read and write later**.

```
    Stream_Read_UINT32(s, OutputBufferSize); // read size
    switch (transferDir)
    {
        case USBD_TRANSFER_DIRECTION_OUT:
            if (Stream_GetRemainingLength(s) < OutputBufferSize)
                return ERROR_INVALID_DATA;
            break;
        default:
            break;
    }
    out = Stream_New(NULL, 36 + OutputBufferSize); // int overflow
```

## int overflow in urb_control_get_status_request

it first read 4 byte to **OutputBufferSize**, then **36 + OutputBufferSize** could **int overflow**. Then it could lead **oob read and write later**.

```
static UINT urb_control_get_status_request
{

    Stream_Read_UINT32(s, OutputBufferSize); // read
    out_size = 36 + OutputBufferSize; // int overflow
    out = Stream_New(NULL, out_size); //
```

## int overflow in urb_control_vendor_or_class_request

it first read 4 byte to **OutputBufferSize**, then **36 + OutputBufferSize** could **int overflow**. Then it could lead **oob read and write later**.

```
    Stream_Read_UINT32(s, OutputBufferSize);
    if (transferDir == USBD_TRANSFER_DIRECTION_OUT)
    {
        if (Stream_GetRemainingLength(s) < OutputBufferSize)
            return ERROR_INVALID_DATA;
    }
    out_size = 36 + OutputBufferSize;  // int overflow
    out = Stream_New(NULL, out_size);
```

## int overflow in urb_control_get_configuration_request

it first read 4 byte to **OutputBufferSize**, then **36 + OutputBufferSize** could **int overflow**. Then it could lead **oob read and write later**.

```
    Stream_Read_UINT32(s, OutputBufferSize);
    out_size = 36 + OutputBufferSize;  // int overflow
    out = Stream_New(NULL, out_size);
```

## int overflow in urb_control_get_interface_request

it first read 4 byte to **OutputBufferSize**, then **36 + OutputBufferSize** could **int overflow**. Then it could lead **oob read and write later**.

```
    Stream_Read_UINT32(s, OutputBufferSize);
    out_size = 36 + OutputBufferSize;  // int overflow
    out = Stream_New(NULL, out_size);
```

## int overflow in urb_os_feature_descriptor_request

it first read 4 byte to **OutputBufferSize**, then **36 + OutputBufferSize** could **int overflow**. Then it could lead **oob read and write later**.

```
    Stream_Read_UINT32(s, OutputBufferSize);
    switch (transferDir)
    {
        case USBD_TRANSFER_DIRECTION_OUT:
            if (Stream_GetRemainingLength(s) < OutputBufferSize)
                return ERROR_INVALID_DATA;
    }
    out_size = 36 + OutputBufferSize; // int overflow
    out = Stream_New(NULL, out_size);
```

## int overflow in msusb_msinterface_read

first read 4 byte NumberOfPipes, NumberOfPipes is UINT32， so it could always enter **msusb_mspipes_read**

```
MSUSB_INTERFACE_DESCRIPTOR* msusb_msinterface_read(wStream* s)
{

    Stream_Read_UINT32(s, MsInterface->NumberOfPipes);

    if (MsInterface->NumberOfPipes > 0)
    {
        MsInterface->MsPipes = msusb_mspipes_read(s, MsInterface-
>NumberOfPipes);
```

if **NumberOfPipes=0XFFFFFFFF**, it could bypass the check, and lead int overflow, then it could **oob read and write** later.

```
static MSUSB_PIPE_DESCRIPTOR** msusb_mspipes_read(wStream* s, UINT32
NumberOfPipes)
{
    // int overflow
    if (Stream_GetRemainingCapacity(s) < 12 * NumberOfPipes)
        return NULL;
    // alloc memory
    MsPipes = (MSUSB_PIPE_DESCRIPTOR**)calloc(NumberOfPipes,
sizeof(MSUSB_PIPE_DESCRIPTOR*));
    // oob read and write.
    for (pnum = 0; pnum < NumberOfPipes; pnum++)
    {
        MSUSB_PIPE_DESCRIPTOR* MsPipe = msusb_mspipe_new();
        if (!MsPipe)
            goto out_error;
        Stream_Read_UINT16(s, MsPipe->MaximumPacketSize);
        ......................
        MsPipes[pnum] = MsPipe;
```

## [dup]int overflow in urb_select_configuration

https://github.com/FreeRDP/FreeRDP/blob/master/channels/urbdrc/client/data_transfer.c#L429

urb_select_configuration first read 4 byte to NumInterfaces, then pass to **msusb_msconfig_read**

```
static UINT urb_select_configuration
    {

        // read 4 byte to NumInterfaces
        Stream_Read_UINT32(s, NumInterfaces);
        if (ConfigurationDescriptorIsValid)
        {
            /* parser data for struct config */
            MsConfig = msusb_msconfig_read(s, NumInterfaces);
```

**msusb_msconfig_read** could check NumInterfaces, but it could bypass by **int overflow,** for example: **NumInterfaces=0xFFFFFFFF**

```
MSUSB_CONFIG_DESCRIPTOR* msusb_msconfig_read
{
    // bypass by int overflow
    if (Stream_GetRemainingCapacity(s) < 6 + NumInterfaces * 2)
```

```
        return NULL;

    MsConfig->MsInterfaces = msusb_msinterface_read_list(s,
NumInterfaces);
```

Then it could read data in **msusb_msinterface_read_list**.

```
static MSUSB_INTERFACE_DESCRIPTOR** msusb_msinterface_read_list
{
    // int overflow
    MsInterfaces =
        (MSUSB_INTERFACE_DESCRIPTOR**)calloc(NumInterfaces,
sizeof(MSUSB_INTERFACE_DESCRIPTOR*));
    if (!MsInterfaces)
        return NULL;
    // oob write.
    for (inum = 0; inum < NumInterfaces; inum++)
    {
        MsInterfaces[inum] = msusb_msinterface_read(s);
        if (!MsInterfaces[inum])
            goto fail;
    }
```

this bug could **lead oob write in msusb_msinterface_read_list**.

## int overflow in urbdrc_process_io_control

https://github.com/FreeRDP/FreeRDP/blob/master/channels/urbdrc/client/data_transfer.c#L237

urbdrc_process_io_control first read 4 byte to **InputBufferSize**, then use **InputBufferSize + 8** to check.

```
        Stream_Read_UINT32(s, IoControlCode);
        Stream_Read_UINT32(s, InputBufferSize);

        if (Stream_GetRemainingLength(s) < InputBufferSize + 8)
            return ERROR_INVALID_DATA;

        Stream_Seek(s, InputBufferSize);
        Stream_Read_UINT32(s, OutputBufferSize);
        Stream_Read_UINT32(s, RequestId);
```

if **InputBufferSize=0xffffffff， and Stream_GetRemainingLength(s) > 8,** it could pass the check.

then **it could crash when read data after Stream_Seek**.

## int overflow in urbdrc_process_internal_io_control

urbdrc_process_internal_io_control first read 4 byte to **InputBufferSize**, then use **InputBufferSize + 8** to check.

```
static UINT urbdrc_process_internal_io_control(IUDEVICE* pdev,
URBDRC_CHANNEL_CALLBACK* callback,
                                               wStream* s, UINT32
MessageId, IUDEVMAN* udevman)
{
    Stream_Read_UINT32(s, IoControlCode);
    Stream_Read_UINT32(s, InputBufferSize);
    if (Stream_GetRemainingLength(s) < InputBufferSize + 8)
        return ERROR_INVALID_DATA;
    Stream_Seek(s, InputBufferSize);
    Stream_Read_UINT32(s, OutputBufferSize);
    Stream_Read_UINT32(s, RequestId);
```

if **InputBufferSize=0xffffffff， and Stream_GetRemainingLength(s) > 8,** it could pass the check.

then **it could crash** when **read data after Stream_Seek**.

## int overflow in smartcard_unpack_locate_cards_w_call

https://github.com/FreeRDP/FreeRDP/blob/master/channels/smartcard/client/smartcard_pack.c#L3270

**smartcard_unpack_locate_cards_w_call** first read **4 byte** to **call->cBytes**, then pass to **smartcard_ndr_read_fixed_string_w**

```
        status =
                smartcard_ndr_read_fixed_string_w(s, &call->mszCards,
call->cBytes, NDR_PTR_SIMPLE);
```

then it could call to **smartcard_ndr_read**, **sizeof(WCHAR) is 2**.

```
static LONG smartcard_ndr_read_fixed_string_w(wStream* s, WCHAR** data,
size_t min, ndr_ptr_t type)
    {
        union {
            WCHAR** ppc;
            BYTE** ppv;
        } u;
        u.ppc = data;
        return smartcard_ndr_read(s, u.ppv, min, sizeof(WCHAR), type);
    }
```

**len = call->cBytes** and **elementSize = 2.**

```
static LONG smartcard_ndr_read(wStream* s, BYTE** data, size_t min, size_t elementSize,
                                    ndr_ptr_t type)
    {
      ..............................
      case NDR_PTR_SIMPLE:
                    Stream_Read_UINT32(s, len);
                    if ((len != min) && (min > 0))
      ..............................
      len *= elementSize;

      r = calloc(len + 1, sizeof(CHAR));  // int overflow
      if (!r)
        return SCARD_E_NO_MEMORY;
      Stream_Read(s, r, len);  // oob write.
```

when **call->cBytes** is large enough, it could **int overflow** when call **calloc**, then **oob write** in Stream_Read.

## int overflow in printer_custom_component

https://github.com/FreeRDP/FreeRDP/blob/master/channels/printer/client/printer_main.c#L119

```
static BOOL printer_write_setting(const char* path, prn_conf_t type, const void* data,
                                  size_t length)
{
    if (length > 0)
    {
        base64 = crypto_base64_encode(data, length);
```

the type of **length** is **size_t**, which is **unsigned**, and in printer_write_setting it check **length > 0, this check is always bypass.**

printer_custom_component could read some data from stream, which max size is **0xffffffff**
https://github.com/FreeRDP/FreeRDP/blob/master/channels/printer/client/printer_main.c#L670

```
            Stream_Read_UINT32(s, PnPNameLen);
            Stream_Read_UINT32(s, DriverNameLen);
            Stream_Read_UINT32(s, PrintNameLen);
            Stream_Read_UINT32(s, CacheFieldsLen);
```

then it could pass to **printer_write_setting**, because length is **size_t** , so it could enter crypto_base64_encode.
**crypto_base64_encode** could int overflow, and could lead oob write later.

```
char* crypto_base64_encode(const BYTE* data, int length)
{
    q = data;
    // if length is 0xffffffff, int overflow.
    p = ret = (char*)malloc((length + 3) * 4 / 3 + 1);
    if (!p)
```

## int overflow in urb_control_descriptor_request

urb_control_descriptor_request could first read **4 byte** to OutputBufferSize, then use **OutputBufferSize+36** to allocate memory for **out->buffer**.
so it could **int overflow** , and allocate **smaller memory**.

```
        Stream_Read_UINT32(s, OutputBufferSize);

        if (transferDir == USBD_TRANSFER_DIRECTION_OUT)
        {
            if (Stream_GetRemainingLength(s) < OutputBufferSize)
                return ERROR_INVALID_DATA;
        }

        out_size = 36 + OutputBufferSize;
        out = Stream_New(NULL, out_size);
```

later could just copy **OutputBufferSize** bytes data to **out->buffer by Stream_Copy**

```
    switch (transferDir)
    {
        case USBD_TRANSFER_DIRECTION_IN:
            bmRequestType |= 0x80;
            break;
        case USBD_TRANSFER_DIRECTION_OUT:
            bmRequestType |= 0x00;
            Stream_Copy(s, out, OutputBufferSize);  // oob write.
            Stream_Rewind(out, OutputBufferSize);
            break;
```

This could lead **oob write.**

## int overflow in video_read_tsmm_presentation_req

video_read_tsmm_presentation_req could read some data from stream
https://github.com/FreeRDP/FreeRDP/blob/master/channels/video/client/video_main.c#L512

```
    Stream_Read_UINT32(s, req.SourceWidth);
        Stream_Read_UINT32(s, req.SourceHeight);
        Stream_Read_UINT32(s, req.ScaledWidth);
```

```
        Stream_Read_UINT32(s, req.ScaledHeight);
        Stream_Read_UINT64(s, req.hnsTimestampOffset);
        Stream_Read_UINT64(s, req.GeometryMappingId);
        Stream_Read(s, req.VideoSubtypeId, 16);
        return video_PresentationRequest(context, &req);
```

Then it could pass data to video_PresentationRequest, and it could use PresentationContext_new to allocate some memory.
https://github.com/FreeRDP/FreeRDP/blob/master/channels/video/client/video_main.c#L449

```
        WLog_DBG(TAG, "creating presentation 0x%x", req->PresentationId);
            presentation = PresentationContext_new(
                video, req->PresentationId, geom->topLevelLeft + geom->left,
                geom->topLevelTop + geom->top, req->SourceWidth, req->SourceHeight);
```

BufferPool_Take
https://github.com/FreeRDP/FreeRDP/blob/master/channels/video/client/video_main.c#L246

```
ret->surfaceData = BufferPool_Take(priv->surfacePool, width * height * 4);
```

**width , height is from network data, is all 4byte, it could int overflow, and could allocate small memory, and when later use it could lead oob write.**