Talos Vulnerability Report

TALOS-2021-1297

GPAC Project on Advanced Content library MPEG-4 Decoding multiple multiplication integer overflow vulnerabilities

AUGUST 16, 2021

CVE NUMBER

CVE-2021-21834, CVE-2021-21835, CVE-2021-21836, CVE-2021-21837, CVE-2021-21838, CVE-2021-21839, CVE-2021-21840, CVE-2021-21841, CVE-2021-21842, CVE-2021-21843, CVE-2021-21844, CVE-2021-21845, CVE-2021-21846, CVE-2021-21847, CVE-2021-21848, CVE-2021-21849, CVE-2021-21850, CVE-2021-21851, CVE-2021-21852

Summary

Multiple exploitable integer overflow vulnerabilities exist within the MPEG-4 decoding functionality of the GPAC Project on Advanced Content library v1.0.1. A specially crafted MPEG-4 input can cause an integer overflow due to unchecked arithmetic resulting in a heap-based buffer overflow that causes memory corruption. An attacker can convince a user to open a video to trigger this vulnerability.

Tested Versions

GPAC Project Advanced Content commit a8a8d412dabcb129e695c3e7d861fcc81f608304
GPAC Project Advanced Content v1.0.1

Product URLs

https://gpac.wp.mines-telecom.fr

CVSSv3 Score

8.8 - CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

CWE

CWE-680 - Integer Overflow to Buffer Overflow

Details

The GPAC Project on Advanced Content is an open-source cross-platform library that implements the MPEG-4 Systems Standard, and provides tools for media playback, vector graphics, and 3d rendering. It supports a variety of multimedia standards and is thus used by a number of industrial users. The project also comes with the MP4Box tool which allows one to encode or decode media containers in a number of supported formats.

When the GPAC library is used to open up an MPEG-4 container, they library will proceed to read each particular atom from the container whilst noting the atom's "type" which is referred to as a FOURCC. This "type" is then used to distinguish which particular parser will be used to parse the contents of an atom. During the parsing of the various atom types inside the MPEG-4 container, the library will read fields from the atom's contents and in some cases will use them to calculate the boundaries of the fields contained within the rest of the atom. In some of these parsers, the fields are explicitly trusted and then used to calculate the size of a heap-buffer which is then later used by the library. When the library allocates space for reading the rest of an atom, the library may miscalculate this size either due to an integer overflow, or an integer truncation which can result in an undersized heap allocation being made. Later in the atom parsing, when the library attempts to read the atom's contents into this heap buffer, a heap-based buffer overflow can be made to occur. This can result in code execution under the context of the library.

The GPAC library provides a variety of tools that the implementer may use when processing an MPEG-4 container. This would allow a user to either process the MPEG-4 container in fragments, or as a whole and complete source. When parsing a complete MPEG-4 container, a developer may use the following `gf_isom_open` function. This function is responsible for looking at the flags that it was given and chaining to the correct function in order to parse the input. At [1], the library will use the `OpenMode` flags from its parameters in order to call the `gf_isom_open_file` to process the input.

```
src/isomedia/isom_read.c:500
GF_EXPORT
GF_ISOFile *gf_isom_open(const char *fileName, GF_ISOOpenMode OpenMode, const char *tmp_dir)
{
    GF_ISOFile *movie;
    MP4_API_IO_Err = GF_OK;

    switch (OpenMode & 0xFF) {
    case GF_ISOM_OPEN_READ_DUMP:
    case GF_ISOM_OPEN_READ:
        movie = gf_isom_open_file(fileName, OpenMode, NULL);        // [1] open up the given filename for reading
        break;
...
    default:
        return NULL;
    }
    return (GF_ISOFile *) movie;
}
```

The following function is the implementation of the `gf_isom_open_file` function. The beginning of this function is responsible for opening up a media source by the library. After the library allocates the necessary data structures for supporting the parsing of a container, a call to the `gf_isom_parse_movie_boxes` function at [2] will be made. As noted in the comment, this is where the actual parsing of the contents of the input will occur. The MPEG-4 container format is based on a type-length-value format in order to define each structure's boundaries. These type-length-value structures are commonly referred to as "atoms" or "boxes". These "atoms" may be recursively defined within the given container.

```
src/isomedia/isom_intern.c:809
GF_ISOFile *gf_isom_open_file(const char *fileName, GF_ISOOpenMode OpenMode, const char *tmp_dir)
{
    GF_Err e;
    u64 bytes;
    GF_ISOFile *mov = gf_isom_new_movie();
    if (!mov || !fileName) return NULL;

    mov->fileName = gf_strdup(fileName);
    mov->openMode = OpenMode;
...
    if ( (OpenMode == GF_ISOM_OPEN_READ) || (OpenMode == GF_ISOM_OPEN_READ_DUMP) || (OpenMode == GF_ISOM_OPEN_READ_EDIT) ) {
        if (OpenMode == GF_ISOM_OPEN_READ_EDIT) {
            mov->openMode = GF_ISOM_OPEN_READ_EDIT;

            // create a memory edit map in case we add samples, typically during import
            e = gf_isom_datamap_new(NULL, tmp_dir, GF_ISOM_DATA_MAP_WRITE, & mov->editFileMap);
            if (e) {
                gf_isom_set_last_error(NULL, e);
                gf_isom_delete_movie(mov);
                return NULL;
            }
        } else {
            mov->openMode = GF_ISOM_OPEN_READ;
        }
...
    }

    //OK, let's parse the movie...
    mov->LastError = gf_isom_parse_movie_boxes(mov, NULL, &bytes, 0);           // [2] parse each of the boxes within the file
```

The `gf_isom_parse_movie_boxes` function is simply a wrapper that will lock the input that is being parsed and then call into the actual parser. After performing the necessary locking around the input, the call at [3] to the `gf_isom_parse_movie_boxes_internal` function will then be called. This function will check the position that has been requested by the caller, use it to seek to the correct position in the input, and then proceed to parse the boxes associated with the container. As the MPEG-4 container format may be recursively defined, the function call at [4] to the `gf_isom_parse_root_box` is called to parse the root element of the movie container.

```
src/isomedia/isom_intern.c:764
GF_Err gf_isom_parse_movie_boxes(GF_ISOFile *mov, u32 *boxType, u64 *bytesMissing, Bool progressive_mode)
{
    GF_Err e;
    GF_Blob *blob = NULL;
...
    e = gf_isom_parse_movie_boxes_internal(mov, boxType, bytesMissing, progressive_mode);                      // [3] \ proceed to parse
the movie boxies
...
    return e;
}
\
src/isomedia/isom_intern.c:289
static GF_Err gf_isom_parse_movie_boxes_internal(GF_ISOFile *mov, u32 *boxType, u64 *bytesMissing, Bool progressive_mode)
{
    GF_Box *a;
    u64 totSize, mdat_end=0;
    GF_Err e = GF_OK;
...
    /*while we have some data, parse our boxes*/
    while (gf_bs_available(mov->movieFileMap->bs)) {
        *bytesMissing = 0;
...
        e = gf_isom_parse_root_box(&a, mov->movieFileMap->bs, boxType, bytesMissing, progressive_mode);         // [4] start by parsing the
root box
...
    }
...
    return GF_OK;
}
```

As prior mentioned, the atoms within an MPEG-4 container are recursively defined. The GPAC library chooses to implement its parser using a recursive algorithm. The primary function within the library's implementation is the `gf_isom_box_parse_ex` function. In the following code, the `gf_isom_parse_root_box` function is simply an entry-point to the recursive parser that lies within the implementation of the `gf_isom_box_parse_ex` function. At [5], the position of the input is set, and then the function call to `gf_isom_box_parse_ex` is used. The `gf_isom_box_parse_ex` function will start by reading the 32-bit size at [6] that is stored at the beginning of an atom's structure. Once the size has been read and checked, the next part of an atom's structure will be read. The next field in an atom is the type, or the FOURCC, which is then read into a local variable at [7]. In order to support larger atom sizes that may not fit entirely within 32-bits, the MPEG-4 standard allows for a 64-bit size. This is done by setting an atom's size to 1, at which point a 64-bit field containing the actual size will follow the FOURCC. At [8], the library will check if the size is 1 and then if so will proceed by reading the next 64-bit field from the atom, and then store it into the original size variable.

```
src/isomedia/box_funcs.c:33
GF_Err gf_isom_parse_root_box(GF_Box **outBox, GF_BitStream *bs, u32 *box_type, u64 *bytesExpected, Bool progressive_mode)
{
    GF_Err ret;
    u64 start;
    start = gf_bs_get_position(bs);
    ret = gf_isom_box_parse_ex(outBox, bs, 0, GF_TRUE);                 // [5] perform the actual parsing of the root box
...
    return ret;
}
\
src/isomedia/box_funcs.c:91
GF_Err gf_isom_box_parse_ex(GF_Box **outBox, GF_BitStream *bs, u32 parent_type, Bool is_root_box)
{
    u32 type, uuid_type, hdr_size, restore_type;
    u64 size, start, comp_start, payload_start, end;
    char uuid[16];
    GF_Err e;
    GF_BitStream *uncomp_bs = NULL;
    u8 *uncomp_data = NULL;
    u32 compressed_size=0;
    GF_Box *newBox;
    Bool skip_logs = (gf_bs_get_cookie(bs) & GF_ISOM_BS_COOKIE_NO_LOGS ) ? GF_TRUE : GF_FALSE;
    Bool is_special = GF_TRUE;

...
    size = (u64) gf_bs_read_u32(bs);                                    // [6] read the 32-bit size from the box or atom
    hdr_size = 4;
    /*fix for some boxes found in some old hinted files*/
    if ((size >= 2) && (size <= 4)) {
        size = 4;
        type = GF_ISOM_BOX_TYPE_VOID;
    } else {
        type = gf_bs_read_u32(bs);                                      // [7] read the 32-bit type or FOURCC from the atom
        hdr_size += 4;
...
    }
...
    //handle large box
    if (size == 1) {                                                    // [8] if the size is 1, then
        if (gf_bs_available(bs) < 8) {
            return GF_ISOM_INCOMPLETE_FILE;
        }
        size = gf_bs_read_u64(bs);                                      // [8] read the next 64-bit integer as the size
        hdr_size += 8;
    }
```

Continuing through the implementation of the `gf_isom_box_parse_ex` function, the function will use the type and size that was read to parse the contents of the atom. This parsed atom will then later be appended to a linked list so that the container may be processed by the library. Within this library, an atom is stored within a structure that is of the type `GF_Box` which is then casted into the actual atom type after it has been constructed. In the following code, the `GF_Box` is first constructed at [9] using the `gf_isom_box_new_ex` function with the type and the atom's parent type as its parameters. After the `GF_Box` has been constructed, it will then be passed to the `gf_isom_full_box_read` function call at [10] in order to read a specific header if the FOURCC requires it, and then to the `gf_isom_box_read` function call at [11] to actually parse the atom.

```
src/isomedia/box_funcs.c:217
    //some special boxes (references and track groups) are handled by a single generic box with an associated ref/group type
    if (parent_type && (parent_type == GF_ISOM_BOX_TYPE_TREF)) {
...
    } else {
        //OK, create the box based on the type
        is_special = GF_FALSE;
        newBox = gf_isom_box_new_ex(uuid_type ? uuid_type : type, parent_type, skip_logs, is_root_box);         // [9] construct space for a
Box (or atom)
        if (!newBox) return GF_OUT_OF_MEM;
    }

...
    newBox->size = size - hdr_size;

    e = gf_isom_full_box_read(newBox, bs);                                                                      // [10] parse an atom's
FullBox header
    if (!e) e = gf_isom_box_read(newBox, bs);                                                                   // [11] parse the contents
of the atom
    if (e) {
        if (gf_opts_get_bool("core", "no-check"))
            e = GF_OK;
    }
    newBox->size = size;
    end = gf_bs_get_position(bs);

...
    return e;
}
```

In order to determine how to construct the `GF_Box` type that is used during parsing, the current atom's type and its parent type are passed to the following function, `gf_isom_box_new_ex`. This function is responsible for looking up the atom's type inside a global array named `box_registry`, allocating the respective `GF_Box` structure, and initialize it with the necessary values prior to it being used. The global array, `box_registry` contains a list of all of the available atom types and is keyed by their FOURCC code. In order to find the index of the FOURCC for the atom being parsed, a call to the `get_box_reg_idx` function is made at [12] and given the FOURCC for the current atom along with the FOURCC of the current atom's parent. Inside the `get_box_reg_idx` function, the library will prepare to do a linear search through the global `box_registry` at [13] by first getting the total number of available FOURCC codes, and then converting the atom's parent FOURCC to a string. Afterwards these values will be used in the loop that follows in order to iterate through each defined element within the `box_registry`. At [14], the loop will then compare the FOURCC code that was passed as one of the function's parameters, and then check if the parent's FOURCC code was found within the current element. If these match the FOURCC provided in the function's parameters, then the index will be returned to the caller which will then use it at [15] to call the constructor that will allocate the real structure for the found FOURCC. Prior to returning to the caller, the `gf_isom_box_new_ex` function will update the `GF_Box` that was constructed with the registry that was used.

```
src/isomedia/box_funcs.c:1630
GF_Box *gf_isom_box_new_ex(u32 boxType, u32 parentType, Bool skip_logs, Bool is_root_box)
{
    GF_Box *a;
    s32 idx = get_box_reg_idx(boxType, parentType, 0);                                  // [12] figure out the index in the registry
    if (idx==0) {
\
src/isomedia/box_funcs.c:1589
static u32 get_box_reg_idx(u32 boxCode, u32 parent_type, u32 start_from)
{
    u32 i=0, count = gf_isom_get_num_supported_boxes();                                 // [13] get available number of boxes
    const char *parent_name = parent_type ? gf_4cc_to_str(parent_type) : NULL;          // [13] convert the parent type to a string

    if (!start_from) start_from = 1;

    for (i=start_from; i<count; i++) {                                                  // [13] enter loop
        u32 start_par_from;
        if (box_registry[i].box_4cc != boxCode)                                         // [14] compare the FOURCC code for the current
registry entry
            continue;

        if (!parent_type)
            return i;
        if (strstr(box_registry[i].parents_4cc, parent_name) != NULL)                   // [14] check that the parent's FOURCC is a
valid type
            return i;
        if (strstr(box_registry[i].parents_4cc, "*") != NULL)
            return i;

        if (strstr(box_registry[i].parents_4cc, "sample_entry") == NULL)
            continue;
...
    }
    return 0;
}
/
src/isomedia/box_funcs.c:1671
    a = box_registry[idx].new_fn();                                                     // [15] construct the GF_Box structure

    if (a) {
...
        a->registry = &box_registry[idx];                                               // [15] assign the registry that was used

        if ((a->type==GF_ISOM_BOX_TYPE_COLR) && (parentType==GF_ISOM_BOX_TYPE_JP2H)) {
            ((GF_ColourInformationBox *)a)->is_jp2 = GF_TRUE;
        }
    }
    return a;
}
```

Once the correct box structure has been constructed, then execution will then return back to the `gf_isom_box_parse_ex` function in order to actually use the `GF_Box`. At [10], the `gf_isom_full_box_read` function will be called to parse a particular category of FOURCC code. Upon entry into the `gf_isom_full_box_read` function, the library will check the `box_registry` entry for the FOURCC to see if it has a version associated with it. If so, the library will read a byte for the version and 3 bytes which maintain the flags for the currently read atom. After it has been read and the `GF_Box` structure has been updated, the library will return back to the `gf_isom_box_parse_ex` function and then pass the current `GF_Box` structure to the `gf_isom_box_read` function at [11]. This function is directly responsible for parsing the atom with the FOURCC that was previously looked up in the global `box_registry` array.

```
src/isomedia/box_funcs.c:262
    newBox->size = size - hdr_size;

    e = gf_isom_full_box_read(newBox, bs);                                                                  // [10] \ parse an atom's
FullBox header
    if (!e) e = gf_isom_box_read(newBox, bs);                                                               // [11] parse the contents
of the atom
    if (e) {
        if (gf_opts_get_bool("core", "no-check"))
            e = GF_OK;
    }
    newBox->size = size;
    end = gf_bs_get_position(bs);
\
src/isomedia/box_funcs.c:1927
static GF_Err gf_isom_full_box_read(GF_Box *ptr, GF_BitStream *bs)
{
    if (ptr->registry->max_version_plus_one) {
        GF_FullBox *self = (GF_FullBox *) ptr;
        ISOM_DECREASE_SIZE(ptr, 4)
        self->version = gf_bs_read_u8(bs);
        self->flags = gf_bs_read_u24(bs);
    }
    return GF_OK;
}
```

In the following code, the library will look at the `registry` field from the `GF_Box` that was passed as its parameter, and use it to access the entry that was discovered when searching the global `box_registry` array for the FOURCC code belonging to the atom read from the input. At [12], the `read_fn` field from the `box_registry` entry is dereferenced in order to continue to parse the contents of the atom that is being processed by the `gf_isom_box_parse_ex` function.

```
src/isomedia/box_funcs.c:1801
GF_Err gf_isom_box_read(GF_Box *a, GF_BitStream *bs)
{
    if (!a) return GF_BAD_PARAM;
    if (!a->registry) {
        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Read invalid box type %s without registry\n", gf_4cc_to_str(a->type) ));
        return GF_ISOM_INVALID_FILE;
    }
    return a->registry->read_fn(a, bs);                    // [12] dispatch to the parser that was stored in the GF_Box registry field.
}
```

CVE-2021-21834 - “co64” decoder

When decoding the atom for the “co64” FOURCC, the following function is used. In this function, at [15] the library first reads the value of the “nb_entries” from the atom as a 32-bit integer, and then checks that it is not larger than the input size. As the input size is a 64-bit integer, this check is not sufficient. Afterwards, the number of entries is multiplied by the size of a `u64` which can result in an integer overflow on 32-bit platforms. Due to this, the allocation may result in an undersized buffer at which point the population of the array that was allocated at [17] will write outside the bounds of the buffer.

```
src/isomedia/box_code_base.c:41
GF_Err co64_box_read(GF_Box *s,GF_BitStream *bs)
{
    u32 entries;
    GF_ChunkLargeOffsetBox *ptr = (GF_ChunkLargeOffsetBox *) s;
    ptr->nb_entries = gf_bs_read_u32(bs);                                       // [15] read nb_entries from file

    ISOM_DECREASE_SIZE(ptr, 4)

    if (ptr->nb_entries > ptr->size / 8) {                                      // [15] check nb_entries is not larger than input size
        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in co64\n", ptr->nb_entries));
        return GF_ISOM_INVALID_FILE;
    }

    ptr->offsets = (u64 *) gf_malloc(ptr->nb_entries * sizeof(u64) );           // [16] allocate space for offsets
    if (ptr->offsets == NULL) return GF_OUT_OF_MEM;
    ptr->alloc_size = ptr->nb_entries;
    for (entries = 0; entries < ptr->nb_entries; entries++) {
        ptr->offsets[entries] = gf_bs_read_u64(bs);                             // [17] read nb_entries into undersized array
    }
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x20000000 which will result in a zero-sized allocation being made. Afterwards, a 64-bit integer is read into the zero-sized buffer on the heap causing memory corruption.

```
=================================================================
==153==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200757 at pc 0xf504ace1 bp 0xffd41268 sp 0xffd41260
WRITE of size 8 at 0xf1200757 thread T0
    #0 0xf504ace0 in co64_box_read /root/src/isomedia/box_code_base.c:58:25
    #1 0xf5254097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf524ed5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf52a4ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf52a4ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf52a3bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52b73f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf52d0061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3a31ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200757 is located 6 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf504a4d9 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf504a4d9 in co64_box_read /root/src/isomedia/box_code_base.c:54:25
    #3 0xf5254097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf524ed5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf52a4ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf52a4ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf52a3bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52b73f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf52d0061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3a31ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:58:25 in co64_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==153==ABORTING
```

CVE-2021-21835 - “csgp” decoder pattern count

When decoding the atom associated with the “csgp” FOURCC, the following function is used. At [17] after reading the atom’s header the library will read a `u32` from the input and store it into the “pattern_count” variable of the `GF_CompactSampleGroupBox` structure. After checking its value against the size of the input, at [18] the library will use its value in a multiplication in order to allocate space for the number of patterns described within the atom. Due to an integer overflow, this can result in an undersized allocation. Later at [19], the library will read integers from the input into the undersized array leading to a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:12213
GF_Err csgp_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i, bits, gidx_mask;
    Bool index_msb_indicates_fragment_local_description, grouping_type_parameter_present;
    u32 pattern_size, scount_size, index_size;
    GF_CompactSampleGroupBox *ptr = (GF_CompactSampleGroupBox *)s;
...
    ISOM_DECREASE_SIZE(ptr, 4);
    ptr->pattern_count = gf_bs_read_u32(bs);                                                    // [17] read u32 from input


    if (ptr->size / ( (pattern_size + scount_size) / 8 ) < ptr->pattern_count )             // [17] check against size
        return GF_ISOM_INVALID_FILE;

    ptr->patterns = gf_malloc(sizeof(GF_CompactSampleGroupPattern) * ptr->pattern_count);   // [18] allocate space for patterns
    if (!ptr->patterns) return GF_OUT_OF_MEM;

    bits = 0;
    for (i=0; i<ptr->pattern_count; i++) {
        ptr->patterns[i].length = gf_bs_read_int(bs, pattern_size);                         // [19] read patterns from input into undersized
array
        ptr->patterns[i].sample_count = gf_bs_read_int(bs, scount_size);
        bits += pattern_size + scount_size;
        if (! (bits % 8)) {
            bits/=8;
            ISOM_DECREASE_SIZE(ptr, bits);
            bits=0;
        }
...
    }
...
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the length to 0x15555556, which when multiplied by the size of a `GF_CompactSampleGroupPattern` on a 32-bit platform, will result in an 8-byte buffer being allocated. As the loop reads 32-bit integers from the atom, it will eventually write outside the bounds of the buffer by 4 bytes.

```
=================================================================
==8==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200758 at pc 0xf516a239 bp 0xffbd12b8 sp 0xffbd12b0
WRITE of size 4 at 0xf1200758 thread T0
    #0 0xf516a238 in csgp_box_read /root/src/isomedia/box_code_base.c:12261:53
    #1 0xf5245097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf523fc28 in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf5295ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf5295ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf5294bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52a83f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf52c1061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3a22ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200758 is located 0 bytes to the right of 8-byte region [0xf1200750,0xf1200758)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf5167aac in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf5167aac in csgp_box_read /root/src/isomedia/box_code_base.c:12248:18
    #3 0xf5245097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf523fc28 in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf5295ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf5295ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf5294bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52a83f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf52c1061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3a22ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:12261:53 in csgp_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa 00[fa]fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==8==ABORTING
```

CVE-2021-21836 - "ctts" decoder

In order to decode an atom using the “ctts” FOURCC code, the following function is used. This function will first read the number of entries from the input at [20], and then check this against the size of the input. Due to the size of the input being 64-bit, this check is insufficient. Afterwards at [21], the library will take the number of entries, store it to a field, and then multiply it by the size of the `GF_DttsEntry` structure. Due to an integer overflow on 32-bit systems, this can result in an undersized array being allocated. Afterwards at [22], the library will read entries from the atom into this undersized buffer. This will write outside the bounds of the undersized buffer which will result in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:386
GF_Err ctts_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i;
    u32 sampleCount;
    GF_CompositionOffsetBox *ptr = (GF_CompositionOffsetBox *)s;

    ISOM_DECREASE_SIZE(ptr, 4);
    ptr->nb_entries = gf_bs_read_u32(bs);                                           // [20] read u32 from input

    if (ptr->nb_entries > ptr->size / 8) {                                          // [20] check entries againsti nput
        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in ctts\n", ptr->nb_entries));
        return GF_ISOM_INVALID_FILE;
    }

    ptr->alloc_size = ptr->nb_entries;                                              // [21] assign number of entries to "alloc_size"
field
    ptr->entries = (GF_DttsEntry *)gf_malloc(sizeof(GF_DttsEntry)*ptr->alloc_size);     // [20] calculate size of allocation using
"alloc_size" field
    if (!ptr->entries) return GF_OUT_OF_MEM;
    sampleCount = 0;
    for (i=0; i<ptr->nb_entries; i++) {
        ISOM_DECREASE_SIZE(ptr, 8);
        ptr->entries[i].sampleCount = gf_bs_read_u32(bs);                           // [22] read entries from atom into undersized array
        if (ptr->version)
            ptr->entries[i].decodingOffset = gf_bs_read_int(bs, 32);
        else
            ptr->entries[i].decodingOffset = (s32) gf_bs_read_u32(bs);
...
    }
...
    return GF_OK;
}
```

Crash Information

When using the provided proof-of-concept, the number of entries is set to 0x20000000 which when multiplied by the size of a `GF_DttsEntry` will result in a zero-sized allocation. Each entry will read a 32-bit integer into the zero-sized buffer, resulting in an overflowing the buffer by 4 bytes.

```
=================================================================
==163==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200750 at pc 0xf507ef40 bp 0xffb7a728 sp 0xffb7a720
WRITE of size 4 at 0xf1200750 thread T0
    #0 0xf507ef3f in ctts_box_read /root/src/isomedia/box_code_base.c:406:31
    #1 0xf5280097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf527ad5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf52d0ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf52d0ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf52cfbc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52e33f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf52fc061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3a5dee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200751 is located 0 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf507d9b0 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf507d9b0 in ctts_box_read /root/src/isomedia/box_code_base.c:401:33
    #3 0xf5280097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf527ad5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf52d0ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf52d0ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf52cfbc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52e33f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf52fc061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3a5dee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:406:31 in ctts_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==163==ABORTING
```

CVE-2021-21837 - "fecr" decoder

The function that is dispatched to parse the atom for the "fecr" FOURCC code is as follows. At [23], the library will read a `u32` for the number of entries from the input. After verifying that this data is available in the input, at [24] the library will multiply the number of entries by the size of the `FECReservoirEntry`. Due to an integer overflow, on 32-bit systems this can result in the allocation being of a smaller size than required. After verifying the buffer has been successfully allocated, at [25] the library will read integers from the input into the undersized buffer. This will write past the boundaries of the undersized allocation, resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:10623
GF_Err fecr_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i;
    FECReservoirBox *ptr = (FECReservoirBox *)s;

    ISOM_DECREASE_SIZE(ptr, (ptr->version ? 4 : 2) );
    ptr->nb_entries = gf_bs_read_int(bs, ptr->version ? 32 : 16);               // [23] read u32 for number of entries from input

    ISOM_DECREASE_SIZE(ptr, ptr->nb_entries * (ptr->version ? 8 : 6) );         // [23] ensure that input contains enough space
    GF_SAFE_ALLOC_N(ptr->entries, ptr->nb_entries, FECReservoirEntry);          // [24] | allocate space for number of entries
    if (!ptr->entries) return GF_OUT_OF_MEM;

    for (i=0; i<ptr->nb_entries; i++) {
        ptr->entries[i].item_id = gf_bs_read_int(bs, ptr->version ? 32 : 16);   // [25] read entries from atom into undersized array
        ptr->entries[i].symbol_count = gf_bs_read_u32(bs);
    }
    return GF_OK;
}
|
include/gpac/tools.h:242
#define GF_SAFE_ALLOC_N(__ptr, __n, __struct) {\
        (__ptr) = (__struct *) gf_malloc( __n * sizeof(__struct));\             // [24] multiply fields in order to make allocation
        if (__ptr) {\
            memset((void *) (__ptr), 0, __n * sizeof(__struct));\
        }\
    }
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x20000000. When this value is multiplied by the size of an `FECReservoirEntry`, will result in a zero-sized allocation being made. As the version determines whether a 16-bit or a 32-bit integer is read from the atom, a 32-bit buffer overflow will occur.

```
=================================================================
==168==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200750 at pc 0xf50e01e7 bp 0xffaeb058 sp 0xffaeb050
WRITE of size 4 at 0xf1200750 thread T0
    #0 0xf50e01e6 in fecr_box_read /root/src/isomedia/box_code_base.c:10636:27
    #1 0xf51ee097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf51e8d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf523eced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf523eced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf523dbc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52513f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf526a061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf39cbee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200751 is located 0 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf50df66d in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf50df66d in fecr_box_read /root/src/isomedia/box_code_base.c:10632:2
    #3 0xf51ee097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf51e8d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf523eced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf523eced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf523dbc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52513f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf526a061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf39cbee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:10636:27 in fecr_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==168==ABORTING
```

CVE-2021-21838 - “fpar” decoder

In order to parse an atom with the “fpar” FOURCC code, the following function is used. After reading a few fields from the beginning of the atom, at [25] the library will read 32-bits from the input if the “version” field is set to a value other than 0. Due to it being verified against a 64-bit size, the check is insufficient. Afterwards at [26], the library will use the `GF_SAFE_ALLOC_N` macro to multiply the number of entries with the size of the `FilePartitionEntry` type. This multiplication can result in an integer overflow which can result in an undersized buffer being made. Later at [27], the library will read integers from the input directly into this undersized buffer. This will eventually write outside the bounds of the allocation, resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:10524
GF_Err fpar_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i;
    GF_Err e;
    FilePartitionBox *ptr = (FilePartitionBox *)s;
...
    ISOM_DECREASE_SIZE(ptr, (ptr->version ? 4 : 2) );
    ptr->nb_entries = gf_bs_read_int(bs, ptr->version ? 32 : 16);           // [25] read 32-bit integer from input if "version" is not 0
    if (ptr->nb_entries > ptr->size / 6)                                    // [25] check entries against 64-bit input size
        return GF_ISOM_INVALID_FILE;

    ISOM_DECREASE_SIZE(ptr, ptr->nb_entries * 6 );
    GF_SAFE_ALLOC_N(ptr->entries, ptr->nb_entries, FilePartitionEntry);     // [26] | allocate space for number of entries
    if (!ptr->entries) return GF_OUT_OF_MEM;

    for (i=0;i < ptr->nb_entries; i++) {
        ptr->entries[i].block_count = gf_bs_read_u16(bs);                   // [27] read entries in atom into undersized array
        ptr->entries[i].block_size = gf_bs_read_u32(bs);
    }
    return GF_OK;
}
|
include/gpac/tools.h:242
#define GF_SAFE_ALLOC_N(__ptr, __n, __struct) {\
        (__ptr) = (__struct *) gf_malloc( __n * sizeof(__struct));\         // [26] multiply fields in order to make allocation
        if (__ptr) {\
            memset((void *) (__ptr), 0, __n * sizeof(__struct));\
        }\
    }
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x20000000. When the library multiplies this by the size of a `FilePartitionEntry`, this will result in a zero-sized allocation being made. As the loop starts out by reading a 16-bit integer, followed by a 32-bit integer, this will start by writing 16-bits past the zero-sized buffer.

```
=================================================================
==173==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1100730 at pc 0xf508e2e7 bp 0xffb2c8f8 sp 0xffb2c8f0
WRITE of size 2 at 0xf1100730 thread T0
    #0 0xf508e2e6 in fpar_box_read /root/src/isomedia/box_code_base.c:10553:31
    #1 0xf519f097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf5199d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf51efced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf51efced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf51eebc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52023f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf521b061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf397cee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1100731 is located 0 bytes to the right of 1-byte region [0xf1100730,0xf1100731)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf508d445 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf508d445 in fpar_box_read /root/src/isomedia/box_code_base.c:10549:2
    #3 0xf519f097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf5199d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf51efced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf51efced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf51eebc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52023f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf521b061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf397cee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:10553:31 in fpar_box_read
Shadow bytes around the buggy address:
  0x3e220090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2200e0: fa fa fa fa fa fa[01]fa fa fa 00 02 fa fa fd fa
  0x3e2200f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e220100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e220110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e220120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e220130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==173==ABORTING
```

CVE-2021-21839 - "pcrb" decoder

The following function is used by the library to parse atoms identified by the "pcrb" FOURCC code. At [28], the library will read a `u32` from the input and store it as the number of sub-segments. Afterwards at [29], this number will be multiplied by the size of a `u64` which can result in an integer overflow. Due to the integer overflow, this can result in an undersized allocation being made. Later at [30], the library will begin to read integers from the input and then combine them when writing them into the buffer that was allocated. Due to the buffer being

undersized, the assignment to the “pcr_values” field will write outside the bounds of the buffer. This is a heap-based buffer overflow and will result in memory corruption.

```
src/isomedia/box_code_base.c:9048
GF_Err pcrb_box_read(GF_Box *s,GF_BitStream *bs)
{
    u32 i;
    GF_PcrInfoBox *ptr = (GF_PcrInfoBox*) s;

    ISOM_DECREASE_SIZE(ptr, 4);
    ptr->subsegment_count = gf_bs_read_u32(bs);                         // [28] read u32 from input

    ptr->pcr_values = gf_malloc(sizeof(u64)*ptr->subsegment_count);     // [29] allocate space for number of subsegments
    if (!ptr->pcr_values) return GF_OUT_OF_MEM;
    for (i=0; i<ptr->subsegment_count; i++) {
        u64 data1 = gf_bs_read_u32(bs);                                 // [30] read 32-bit integer from input
        u64 data2 = gf_bs_read_u16(bs);                                 // [30] read 16-bit integer from input
        ISOM_DECREASE_SIZE(ptr, 6);
        ptr->pcr_values[i] = (data1 << 10) | (data2 >> 6);              // [30] combine integers and write into undersized buffer

    }
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of sub-segments to 0x20000000. When this value is multiplied by the size of a `u64` type, this resulted in a zero-sized allocation being made. The loop reads a 32-bit integer followed by a 16-bit integer which is then combined into a 64-bit number which is then written into the zero-sized buffer.

```
=================================================================
==187==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1100757 at pc 0xf5082ffc bp 0xffa21578 sp 0xffa21570
WRITE of size 8 at 0xf1100757 thread T0
    #0 0xf5082ffb in pcrb_box_read /root/src/isomedia/box_code_base.c:9062:22
    #1 0xf51b9097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf51b3c28 in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf5209ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf5209ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf5208bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf521c3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf5235061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3996ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1100757 is located 6 bytes to the right of 1-byte region [0xf1100750,0xf1100751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf508264b in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf508264b in pcrb_box_read /root/src/isomedia/box_code_base.c:9056:20
    #3 0xf51b9097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf51b3c28 in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf5209ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf5209ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf5208bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf521c3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf5235061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3996ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:9062:22 in pcrb_box_read
Shadow bytes around the buggy address:
  0x3e220090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2200e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2200f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e220100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e220110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e220120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e220130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==187==ABORTING
```

CVE-2021-21840 - “saio” decoder

The following function is used to process an atom using the “saio” FOURCC code. This function will first read a 32-bit integer representing the number of entries at [31]. After verifying the number of entries against the atom’s 64-bit size, the library will take the 32-bit count and multiply it by the size of a `u32` or `u64` depending on the “version” field that was previously read by the `gf_isom_full_box_read` function. Due to an integer overflow, when passing result to the `gf_malloc` function at [32], this can result in the allocated buffer being of a smaller size than expected. Later at [33] when reading data from the input, these loops will read data into the buffer writing outside its bounds and resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:10021
GF_Err saio_box_read(GF_Box *s, GF_BitStream *bs)
{
    GF_SampleAuxiliaryInfoOffsetBox *ptr = (GF_SampleAuxiliaryInfoOffsetBox *)s;

...
    ISOM_DECREASE_SIZE(ptr, 4);
    ptr->entry_count = gf_bs_read_u32(bs);                                  // [31] read 32-bit count from atom

    if (ptr->entry_count) {
        u32 i;
        if (ptr->size / (ptr->version == 0 ? 4 : 8) < ptr->entry_count)     // [31] check 64-bit size against 32-bit count
            return GF_ISOM_INVALID_FILE;
        ptr->offsets = gf_malloc(sizeof(u64)*ptr->entry_count);             // [32] allocate heap buffer
        if (!ptr->offsets)
            return GF_OUT_OF_MEM;
        ptr->entry_alloc = ptr->entry_count;
        if (ptr->version==0) {
            ISOM_DECREASE_SIZE(ptr, 4*ptr->entry_count);
            for (i=0; i<ptr->entry_count; i++)
                ptr->offsets[i] = gf_bs_read_u32(bs);                       // [33] read 32-bit integer from input in loop
        } else {
            ISOM_DECREASE_SIZE(ptr, 8*ptr->entry_count);
            for (i=0; i<ptr->entry_count; i++)
                ptr->offsets[i] = gf_bs_read_u64(bs);                       // [33] read 64-bit integer from input in loop
        }
    }
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x20000000. As this number is multiplied by the size of a `u64`, this will result in a zero-sized allocation being made. Depending on the “Version” that was specified at the beginning of the atom, either a 32-bit or 64-bit integer will be read into the zero-sized buffer.

```
=================================================================
==197==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200757 at pc 0xf5128d45 bp 0xffe7ac18 sp 0xffe7ac10
WRITE of size 8 at 0xf1200757 thread T0
    #0 0xf5128d44 in saio_box_read /root/src/isomedia/box_code_base.c:10048:21
    #1 0xf5247097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf5241d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf5297ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf5297ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf5296bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52aa3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf52c3061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3a24ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200757 is located 6 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf5127463 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf5127463 in saio_box_read /root/src/isomedia/box_code_base.c:10037:18
    #3 0xf5247097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf5241d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf5297ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf5297ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf5296bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52aa3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf52c3061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3a24ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:10048:21 in saio_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==197==ABORTING
```

CVE-2021-21841 - “sbgp" decoder

When reading an atom using the “sbgp” FOURCC code, the following function will be used. This function will read a 32-bit integer from the atom at [33] representing the number of sample group entries. Afterwards it will check the number of entries against the atom’s size. Due to the size being 64-bit, this check is insufficient. The library will then use the number of entries and multiply it by the size of the `GF_SampleGroupEntry` structure. This can result in an integer overflow which when passed to the `gf_malloc` function will cause an undersized buffer to be

returned. Later, the library will continue to read two 32-bit integers for the number of sample group entries. Due to the integer overflow causing the allocation to be undersized, this loop will write outside the bounds of the heap buffer causing a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:9366
GF_Err sbgp_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i;
    GF_SampleGroupBox *ptr = (GF_SampleGroupBox *)s;
...
    ptr->entry_count = gf_bs_read_u32(bs);                                          // [33] read 32-bit number of entries from atom

    if (ptr->size < sizeof(GF_SampleGroupEntry)*ptr->entry_count)                   // [34] check its against the 64-bit size
        return GF_ISOM_INVALID_FILE;

    ptr->sample_entries = gf_malloc(sizeof(GF_SampleGroupEntry)*ptr->entry_count);  // [35] multiply number of entries and use it to
allocate memory
    if (!ptr->sample_entries) return GF_OUT_OF_MEM;

    for (i=0; i<ptr->entry_count; i++) {
        ISOM_DECREASE_SIZE(ptr, 8);
        ptr->sample_entries[i].sample_count = gf_bs_read_u32(bs);                   // [36] read data from file into undersized buffer
        ptr->sample_entries[i].group_description_index = gf_bs_read_u32(bs);
    }
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept specifies the number of entries to be 0x20000000. When this number is multiplied by the size of the `GF_SampleGroupEntry` type, this will result in a zero-sized allocation being made. The loop that will use this array will read two 32-bit integers for each entry into the zero-sized array.

```
=================================================================
==202==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200750 at pc 0xf50e5fa5 bp 0xffec9498 sp 0xffec9490
WRITE of size 4 at 0xf1200750 thread T0
    #0 0xf50e5fa4 in sbgp_box_read /root/src/isomedia/box_code_base.c:9388:39
    #1 0xf5214097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf520ed5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf5264ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf5264ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf5263bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52773f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf5290061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf39f1ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200751 is located 0 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf50e5333 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf50e5333 in sbgp_box_read /root/src/isomedia/box_code_base.c:9383:24
    #3 0xf5214097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf520ed5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf5264ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf5264ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf5263bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52773f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf5290061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf39f1ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:9388:39 in sbgp_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==202==ABORTING
```

CVE-2021-21842 - "ssix" decoder subsegment count

When processing an atom using the "ssix" FOURCC code, the following function will be used by the library. This parser will first read a 32-bit integer from the atom at [37] representing the number of sub-segments, and then check it against the 64-bit atom size. Due to the atom size being 64-bit, this check is insufficient. Afterwards, this number of sub-segments will be multiplied by the size of the `GF_SubsegmentInfo` structure at [38]. Due to an integer overflow, this can result in an undersized allocation being made. Afterwards at [39], the library will continue to populate the sub-segment information array. Due to the array being undersized as a result of the integer overflow, this loop will read data from the atom outside the bounds of the

array. This will corrupt memory resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:8854
GF_Err ssix_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i,j;
    GF_SubsegmentIndexBox *ptr = (GF_SubsegmentIndexBox*)s;

    ISOM_DECREASE_SIZE(ptr, 4)
    ptr->subsegment_count = gf_bs_read_u32(bs);                                     // [37] read 32-bit number of subsegments
    //each subseg has at least one range_count (4 bytes), abort if not enough bytes (broken box)
    if (ptr->size / 4 < ptr->subsegment_count)                                      // [37] check against 64-bit size
        return GF_ISOM_INVALID_FILE;

    ptr->subsegment_alloc = ptr->subsegment_count;
    GF_SAFE_ALLOC_N(ptr->subsegments, ptr->subsegment_count, GF_SubsegmentInfo);    // [38] | allocate space for number of subsegments
    if (!ptr->subsegments)
        return GF_OUT_OF_MEM;
    for (i = 0; i < ptr->subsegment_count; i++) {
        GF_SubsegmentInfo *subseg = &ptr->subsegments[i];
        ISOM_DECREASE_SIZE(ptr, 4)
        subseg->range_count = gf_bs_read_u32(bs);                                   // [39] read into undersized allocation
...
    }
    return GF_OK;
}
|
include/gpac/tools.h:242
#define GF_SAFE_ALLOC_N(__ptr, __n, __struct) {\
        (__ptr) = (__struct *) gf_malloc( __n * sizeof(__struct));\                 // [38] multiply fields in order to make allocation
        if (__ptr) {\
            memset((void *) (__ptr), 0, __n * sizeof(__struct));\
        }\
    }
```

Crash Information

The provided proof-of-concept sets the number of sub-segments to 0x20000001. When this value is multiplied by the size of the `GF_SubsegmentInfo` structure, this will result in an allocation of 8-bytes in size being made on 32-bit platforms, or 16-bytes on 64-bit platforms. After the allocation, the loop will iterate 0x20000001 times whilst reading 32-bit integers and some other fields into the undersized buffer.

```
=================================================================
==18==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200758 at pc 0xf50cf530 bp 0xffae7078 sp 0xffae7070
WRITE of size 4 at 0xf1200758 thread T0
    #0 0xf50cf52f in ssix_box_read /root/src/isomedia/box_code_base.c:8872:23
    #1 0xf520b097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf5205d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf525bced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf525bced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf525abc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf526e3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf5287061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf39e8ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200758 is located 0 bytes to the right of 8-byte region [0xf1200750,0xf1200758)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf50ce212 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf50ce212 in ssix_box_read /root/src/isomedia/box_code_base.c:8866:2
    #3 0xf520b097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf5205d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf525bced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf525bced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf525abc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf526e3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf5287061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf39e8ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:8872:23 in ssix_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa 00[fa]fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==18==ABORTING
```

CVE-2021-21843 - “ssix” decoder range count

As previously mentioned, the following function is used to parse the contents of an atom using the “ssix” FOURCC code. When reading each sub-segment, at [40] the library will first read a 32-bit integer representing the number of ranges. These ranges are then checked against the size of the atom. Due to the size being 64-bit, this check is insufficient. After validating the number of ranges, at [41] the library will multiply the count by the size of the `GF_SubsegmentRangeInfo` structure. On a 32-bit platform, this multiplication can result in an integer overflow causing the space of the array being allocated to be less than expected. During the initialization of this array at [42], the library will write outside the bounds of the allocation resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:8854
GF_Err ssix_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i,j;
    GF_SubsegmentIndexBox *ptr = (GF_SubsegmentIndexBox*)s;
...
    for (i = 0; i < ptr->subsegment_count; i++) {
        GF_SubsegmentInfo *subseg = &ptr->subsegments[i];
        ISOM_DECREASE_SIZE(ptr, 4)
        subseg->range_count = gf_bs_read_u32(bs);                                                                   // [40] read 32-bit
integer for range count
        //each range is 4 bytes, abort if not enough bytes
        if (ptr->size / 4 < subseg->range_count)                                                                    // [40] check range
count against 64-bit size
            return GF_ISOM_INVALID_FILE;
        subseg->ranges = (GF_SubsegmentRangeInfo*) gf_malloc(sizeof(GF_SubsegmentRangeInfo) * subseg->range_count); // [41] allocate space
for ranges
        if (!subseg->ranges) return GF_OUT_OF_MEM;
        for (j = 0; j < subseg->range_count; j++) {
            ISOM_DECREASE_SIZE(ptr, 4)
            subseg->ranges[j].level = gf_bs_read_u8(bs);                                                            // [42] read ranges
directly into undersized array
            subseg->ranges[j].range_size = gf_bs_read_u24(bs);
        }
    }
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the “range_count” field of one of the sub-segment entries to 0x20000000. When this value is multiplied by the size of the `GF_SubsegmentRangeInfo` type, this will result in a zero-sized allocation being made. The loop responsible for reading each entry will write 32-bits into the zero-sized buffer.

```
=================================================================
==22==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200734 at pc 0xf50dc5c4 bp 0xffe2a6f8 sp 0xffe2a6f0
WRITE of size 4 at 0xf1200734 thread T0
    #0 0xf50dc5c3 in ssix_box_read /root/src/isomedia/box_code_base.c:8881:33
    #1 0xf5218097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf5212d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf5268ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf5268ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf5267bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf527b3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf5294061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf39f5ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200734 is located 3 bytes to the right of 1-byte region [0xf1200730,0xf1200731)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf50db496 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf50db496 in ssix_box_read /root/src/isomedia/box_code_base.c:8876:46
    #3 0xf5218097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf5212d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf5268ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf5268ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf5267bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf527b3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf5294061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf39f5ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:8881:33 in ssix_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa[01]fa fa fa 00 fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==22==ABORTING
```

CVE-2021-21844 - “stco” decoder

When encountering an atom using the “stco” FOURCC code, the library will use the following function. This function contains a number of offsets for a sample stored using an array of `u32` entries. To read the contents of this atom, at [43], the function will read the number of offsets from the atom and then check the result against the size of the atom. Due to the size of the atom being 64-bit, this check is not sufficient for checking the bounds of the number. At [44], the library will then take the number of offsets and multiply them by the size of the `u32` type. This multiplication can result in an integer overflow, at which point an undersized integer will be used to allocate space on the heap. After validating that the heap allocation was successful, at [45], the function will continue to read the contents of the atom into the array. Due to the integer overflow, this loop will write outside the bounds of the array causing a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:5091
GF_Err stco_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 entries;
    GF_ChunkOffsetBox *ptr = (GF_ChunkOffsetBox *)s;

    ISOM_DECREASE_SIZE(ptr, 4);
    ptr->nb_entries = gf_bs_read_u32(bs);                                   // [43] read 32-bit integer from atom
    if (ptr->nb_entries > ptr->size / 4) {
        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in stco\n", ptr->nb_entries));
        return GF_ISOM_INVALID_FILE;
    }

    if (ptr->nb_entries) {
        ptr->offsets = (u32 *) gf_malloc(ptr->nb_entries * sizeof(u32) );   // [44] allocate memory for offsets
        if (ptr->offsets == NULL) return GF_OUT_OF_MEM;
        ptr->alloc_size = ptr->nb_entries;

        for (entries = 0; entries < ptr->nb_entries; entries++) {
            ptr->offsets[entries] = gf_bs_read_u32(bs);                     // [45] read 32-bit integers from file into undersized array
        }
    }
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x40000000. When this value is multiplied by the size of a `u32`, this will result in a zero-sized allocation being made. When the implementation reads the contents of the atom, it reads 32-bit integers at a time, thus writing 32-bits past the zero-sized buffer.

```
=================================================================
==222==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1100750 at pc 0xf5019042 bp 0xffe474c8 sp 0xffe474c0
WRITE of size 4 at 0xf1100750 thread T0
    #0 0xf5019041 in stco_box_read /root/src/isomedia/box_code_base.c:5109:26
    #1 0xf51af097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf51a9d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf51ffced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf51ffced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf51febc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52123f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf522b061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf398cee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1100751 is located 0 bytes to the right of 1-byte region [0xf1100750,0xf1100751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf501888e in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf501888e in stco_box_read /root/src/isomedia/box_code_base.c:5104:26
    #3 0xf51af097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf51a9d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf51ffced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf51ffced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf51febc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52123f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf522b061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf398cee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:5109:26 in stco_box_read
Shadow bytes around the buggy address:
  0x3e220090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2200d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2200e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2200f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e220100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e220110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e220120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e220130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==222==ABORTING
```

CVE-2021-21845 - “stsc” decoder

When processing an atom with the “stsc” FOURCC code, the library will use the following function. This function will read the atom and extract the information required to produce the samples for the input video. At [46], the library will read the number of sample entries from the atom, and then check it against the 64-bit atom size. Due to the atom size being 64-bits, this check is insufficient. Afterwards at [48], the function will take the number of entries and multiply it by the size of the `GF_StscEntry` structure. Due to an integer overflow, the product of this calculation can result in the allocation for the space of the array to be smaller than required for reading each entry of the atom. Later at [49], when the function reads entries into each element of this undersized array, the loop will write outside its bounds. This will result in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:5222
GF_Err stsc_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i;
    GF_SampleToChunkBox *ptr = (GF_SampleToChunkBox *)s;

    ISOM_DECREASE_SIZE(ptr, 4);
    ptr->nb_entries = gf_bs_read_u32(bs);                                   // [46] read number of sample entries

    if (ptr->nb_entries > ptr->size / 12) {                                 // [46] check against atom size
        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in stsc\n", ptr->nb_entries));
        return GF_ISOM_INVALID_FILE;
    }

    ptr->alloc_size = ptr->nb_entries;                                      // [47] store to u32 field
    ptr->entries = NULL;
    if (ptr->nb_entries) {
        ptr->entries = gf_malloc(sizeof(GF_StscEntry)*ptr->alloc_size);     // [48] allocate array for sample entries
        if (!ptr->entries) return GF_OUT_OF_MEM;
    }

    for (i = 0; i < ptr->nb_entries; i++) {
        ptr->entries[i].firstChunk = gf_bs_read_u32(bs);                    // [49] write to undersized array
        ptr->entries[i].samplesPerChunk = gf_bs_read_u32(bs);
        ptr->entries[i].sampleDescriptionIndex = gf_bs_read_u32(bs);
        ptr->entries[i].isEdited = 0;
        ptr->entries[i].nextChunk = 0;
...
    }
...
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of entries to 0xcccccccd. When this is multiplied by the size of the aligned `GF_StscEntry` type, this will result in a 4-byte allocation being made. The loop will read a number of 32-bit integers into this undersized heap buffer resulting in a 32-bit buffer overflow.

```
=================================================================
==232==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200758 at pc 0xf50a2b1c bp 0xffed2978 sp 0xffed2970
WRITE of size 4 at 0xf1200758 thread T0
    #0 0xf50a2b1b in stsc_box_read /root/src/isomedia/box_code_base.c:5244:35
    #1 0xf5235097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf522fd5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf5285ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf5285ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf5284bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52983f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf52b1061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3a12ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200758 is located 4 bytes to the right of 4-byte region [0xf1200750,0xf1200754)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf50a17a7 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf50a17a7 in stsc_box_read /root/src/isomedia/box_code_base.c:5238:18
    #3 0xf5235097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf522fd5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf5285ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf5285ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf5284bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52983f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf52b1061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3a12ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:5244:35 in stsc_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa 04[fa]fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==232==ABORTING
```

CVE-2021-21846 - "stsz" decoder

The parser for an atom using the "stsz" FOURCC code is implemented by the following function. In this function, after verifying the FOURCC code, the number of samples is read as a 32-bit integer from the atom at [50]. After checking the number of samples against the 64-bit size, at [52] the function will use the product of the number of samples with the size of the `u32` type in order to determine how much space is to be allocated on the heap. Due to an integer overflow, this can result in the allocation returning less space than expected. At [53], when the contents of the atom is read into the array, the function will write outside the bounds of the array resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:5528
GF_Err stsz_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i, estSize;
    GF_SampleSizeBox *ptr = (GF_SampleSizeBox *)s;
    if (ptr == NULL) return GF_BAD_PARAM;

    //support for CompactSizes
    if (s->type == GF_ISOM_BOX_TYPE_STSZ) {
        ISOM_DECREASE_SIZE(ptr, 8);
        ptr->sampleSize = gf_bs_read_u32(bs);
        ptr->sampleCount = gf_bs_read_u32(bs);                                  // [50] read 32-bit integer for sample count
    } else {
...
    }
    if (s->type == GF_ISOM_BOX_TYPE_STSZ) {
        if (! ptr->sampleSize && ptr->sampleCount) {
            if (ptr->sampleCount > ptr->size / 4) {                             // [51] check against 64-bit atom size
                GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in stsz\n", ptr->sampleCount));
                return GF_ISOM_INVALID_FILE;
            }
            ptr->sizes = (u32 *) gf_malloc(ptr->sampleCount * sizeof(u32));     // [52] use sample count to make allocation
            if (! ptr->sizes) return GF_OUT_OF_MEM;
            ptr->alloc_size = ptr->sampleCount;
            for (i = 0; i < ptr->sampleCount; i++) {
                ptr->sizes[i] = gf_bs_read_u32(bs);                             // [53] populate undersized array
                if (ptr->max_size < ptr->sizes[i])
                    ptr->max_size = ptr->sizes[i];
                ptr->total_size += ptr->sizes[i];
                ptr->total_samples++;
            }
        }
...
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of samples to 0x40000000. When this value is multiplied by the size of the `u32` type, this will result in the value 0x10000000 which when truncated will cause the `gf_malloc` function to return a zero-sized buffer. The loop reads 32-bit integers from the atom which will result in writing 32-bits outside the bounds of the heap buffer.

```
=================================================================
==242==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200750 at pc 0xf50a0227 bp 0xffa869a8 sp 0xffa869a0
WRITE of size 4 at 0xf1200750 thread T0
    #0 0xf50a0226 in stsz_box_read /root/src/isomedia/box_code_base.c:5579:19
    #1 0xf522a097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf5224d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf527aced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf527aced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf5279bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf528d3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf52a6061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3a07ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200751 is located 0 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf509dd49 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf509dd49 in stsz_box_read /root/src/isomedia/box_code_base.c:5575:25
    #3 0xf522a097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf5224d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf527aced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf527aced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf5279bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf528d3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf52a6061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3a07ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:5579:19 in stsz_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==242==ABORTING
```

CVE-2021-21847 - “stts” decoder

When encountering an atom with the “stts” FOURCC code, the library will use the following function. This atom is used to convert a unit of time into the actual samples used to decode the video input. When decoding this atom, at [54] the function will read a 32-bit number in order to store the number of entries within the atom. After reading the number of sample entries, the function will then check them against the atom’s size. Due to the atom’s size being a 64-bit integer, this check is insufficient. Once the number of entries has been checked, the function will use the product of the number of entries with the size of the `GF_SttsEntry` structure, at [55], in order to allocate space on the heap for storing the atom’s contents. Due to an integer overflow, this multiplication can result in the array being undersized. Thus, when reading the contents of each entry at [56], the function will write outside the bounds of the array, resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:5762
GF_Err stts_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i;
    GF_TimeToSampleBox *ptr = (GF_TimeToSampleBox *)s;
...
    ISOM_DECREASE_SIZE(ptr, 4);
    ptr->nb_entries = gf_bs_read_u32(bs);                               // [54] read 32-bit number of sample entries
    if (ptr->size / 8 < ptr->nb_entries) {                              // [54] check them against 64-bit atom size
        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in stts\n", ptr->nb_entries));
        return GF_ISOM_INVALID_FILE;
    }

    ptr->alloc_size = ptr->nb_entries;                                  // [55] store the number of entries
    ptr->entries = gf_malloc(sizeof(GF_SttsEntry)*ptr->alloc_size);     // [55] use the number of entries to calculate size of array
    if (!ptr->entries) return GF_OUT_OF_MEM;

    for (i=0; i<ptr->nb_entries; i++) {
        ptr->entries[i].sampleCount = gf_bs_read_u32(bs);               // [56] read entries into undersized array
        ptr->entries[i].sampleDelta = gf_bs_read_u32(bs);
...
    }
...
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x20000000. This value is multiplied by the size of a `GF_SttsEntry` type which will result in a zero-sized allocation being made. Later, the parser will start by reading two 32-bit integers into the zero-sized heap buffer.

```
=================================================================
==247==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200750 at pc 0xf50acb90 bp 0xffe79a88 sp 0xffe79a80
WRITE of size 4 at 0xf1200750 thread T0
    #0 0xf50acb8f in stts_box_read /root/src/isomedia/box_code_base.c:5783:31
    #1 0xf5232097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf522cd5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf5282ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf5282ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf5281bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52953f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf52ae061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3a0fee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200751 is located 0 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf50ab0c5 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf50ab0c5 in stts_box_read /root/src/isomedia/box_code_base.c:5779:17
    #3 0xf5232097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf522cd5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf5282ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf5282ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf5281bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52953f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf52ae061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3a0fee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:5783:31 in stts_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==247==ABORTING
```

CVE-2021-21848 - “stz2” decoder

The library will actually reuse the parser for atoms with the “stsz” FOURCC code when parsing atoms that use the “stz2” FOURCC code. Similarly, at [57] the function will read a 32-bit integer representing the number of samples from the beginning of the atom. After reading a few more fields such as the “sampleSize”, the function will use this “sampleSize” to check the number of samples against the 64-bit size of the atom. Due to the atom size being 64-bits, this check will be insufficient. Afterwards at [59], the function will multiply the number of samples against the size of a `u32` type which can result in an integer overflow. Due to the integer overflow, this allocation can result in a smaller size than required. Thus at [60], when the function reads the contents of the atom into this array, a heap-based buffer overflow can be made to occur.

```
src/isomedia/box_code_base.c:5528
GF_Err stsz_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i, estSize;
    GF_SampleSizeBox *ptr = (GF_SampleSizeBox *)s;
    if (ptr == NULL) return GF_BAD_PARAM;

    //support for CompactSizes
    if (s->type == GF_ISOM_BOX_TYPE_STSZ) {
...
    } else {
        //24-reserved
        ISOM_DECREASE_SIZE(ptr, 8);
        gf_bs_read_int(bs, 24);
        i = gf_bs_read_u8(bs);
        ptr->sampleCount = gf_bs_read_u32(bs);                              // [57] read the 32-bit number of samples from the atom
...
    }
    if (s->type == GF_ISOM_BOX_TYPE_STSZ) {
...
    } else {
        if (ptr->sampleSize==4) {
            if (ptr->sampleCount / 2 > ptr->size) {                         // [58] check the number of samples
                GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in stsz\n", ptr->sampleCount));
                return GF_ISOM_INVALID_FILE;
            }
        } else {
            if (ptr->sampleCount > ptr->size / (ptr->sampleSize/8)) {       // [58] check the number of samples
                GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in stsz\n", ptr->sampleCount));
                return GF_ISOM_INVALID_FILE;
            }
        }
        //note we could optimize the mem usage by keeping the table compact
        //in memory. But that would complicate both caching and editing
        //we therefore keep all sizes as u32 and uncompress the table
        ptr->sizes = (u32 *) gf_malloc(ptr->sampleCount * sizeof(u32));     // [59] allocate space for each entry
        if (! ptr->sizes) return GF_OUT_OF_MEM;
        ptr->alloc_size = ptr->sampleCount;

        for (i = 0; i < ptr->sampleCount; ) {
            switch (ptr->sampleSize) {
            case 4:
                ptr->sizes[i] = gf_bs_read_int(bs, 4);                      // [60] read contents of atom into undersized array
    ...
            }
    ...
        }
    }
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x40000000. This value is multiplied by the size of a `u32` type which will result in a zero-sized allocation being made. Depending on the “sampleSize” field, a 32-bit integer can then be read into the zero-sized buffer resulting in the buffer overflow.

```
=================================================================
==252==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200750 at pc 0xf50710f7 bp 0xffc470c8 sp 0xffc470c0
WRITE of size 4 at 0xf1200750 thread T0
    #0 0xf50710f6 in stsz_box_read /root/src/isomedia/box_code_base.c:5618:19
    #1 0xf51fb097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf51f5d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf524bced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf524bced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf524abc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf525e3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf5277061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf39d8ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200751 is located 0 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf506f610 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf506f610 in stsz_box_read /root/src/isomedia/box_code_base.c:5601:24
    #3 0xf51fb097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf51f5d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf524bced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf524bced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf524abc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf525e3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf5277061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf39d8ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:5618:19 in stsz_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==252==ABORTING
```

CVE-2021-21849 - "tfra" decoder

When the library encounters an atom using the "tfra" FOURCC code, the following function will be used to parse it. This function will use the "version" field that was parsed by the `gf_isom_full_box_read` in order to determine the size that is checked. At [61], the function will read a 32-bit integer from the atom as the number of entries, and then check it against the 64-bit atom size. As the atom size is 64-bits, both of these checks are insufficient. After checking the size, the function will multiply the number of entries by the size of the `GF_RandomAccessEntry` structure which is used to store each entry. As this multiplication is the product of a 32-bit number and the structure's size, the result will be more than 32-bits which is an integer overflow. At [63], the overflow result will be passed to the `gf_malloc` function resulting in an undersized array. At [64], the function will assign a pointer to the beginning of the array, and then enter a loop writing the contents of the atom to said pointer. Due to the integer overflow, this loop will write outside the bounds of the array resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:3171
GF_Err tfra_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i;
    GF_RandomAccessEntry *p = 0;
    GF_TrackFragmentRandomAccessBox *ptr = (GF_TrackFragmentRandomAccessBox *)s;

    ISOM_DECREASE_SIZE(ptr, 12);
...
    ptr->nb_entries = gf_bs_read_u32(bs);                                                   // [61] read 32-bit number of
entries

    if (ptr->version == 1) {
        if (ptr->nb_entries > ptr->size / (16+(ptr->traf_bits+ptr->trun_bits+ptr->sample_bits)/8)) {    // [62] check against atom size
            GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in traf\n", ptr->nb_entries));
            return GF_ISOM_INVALID_FILE;
        }
    } else {
        if (ptr->nb_entries > ptr->size / (8+(ptr->traf_bits+ptr->trun_bits+ptr->sample_bits)/8)) {     // [62] check against atom size
            GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in traf\n", ptr->nb_entries));
            return GF_ISOM_INVALID_FILE;
        }
    }

    if (ptr->nb_entries) {
        p = (GF_RandomAccessEntry *) gf_malloc(sizeof(GF_RandomAccessEntry) * ptr->nb_entries);         // [63] allocate array
        if (!p) return GF_OUT_OF_MEM;
    }

    ptr->entries = p;                                                                       // [64] store pointer to array

    for (i=0; i<ptr->nb_entries; i++) {
        memset(p, 0, sizeof(GF_RandomAccessEntry));                                         // [64] zero memory currently
pointed to

        if (ptr->version == 1) {
            p->time = gf_bs_read_u64(bs);
            p->moof_offset = gf_bs_read_u64(bs);
        } else {
            p->time = gf_bs_read_u32(bs);
            p->moof_offset = gf_bs_read_u32(bs);
        }
        p->traf_number = gf_bs_read_int(bs, ptr->traf_bits);
        p->trun_number = gf_bs_read_int(bs, ptr->trun_bits);
        p->sample_number = gf_bs_read_int(bs, ptr->sample_bits);

        ++p;
    }

    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x924924a. When this value is multiplied by the size of the `GF_RandomAccessEntry` type, the resulting size will be 0x100000018. When this value is truncated to 32-bits, an allocation of the size 0x18 will then be made. In the loop that follows, the function will first zero out the current position into the array prior to reading the rest of the integers from the atom triggering the buffer overflow.

```
=================================================================
==262==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf0800b98 at pc 0x004dc1e0 bp 0xffb75fc8 sp 0xffb75ba8
WRITE of size 28 at 0xf0800b98 thread T0
    #0 0x4dc1df in __asan_memset (/root/harness/parser32.asan+0x4dc1df)
    #1 0xf505186a in tfra_box_read /root/src/isomedia/box_code_base.c:3209:3
    #2 0xf521c097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #3 0xf5216d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #4 0xf526cced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #5 0xf526cced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #6 0xf526bbc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #7 0xf527f3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #8 0xf5298061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #9 0x512f6a in main /root/harness/parser.c:50:13
    #10 0xf39f9ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #11 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf0800b98 is located 0 bytes to the right of 24-byte region [0xf0800b80,0xf0800b98)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf5051730 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf5051730 in tfra_box_read /root/src/isomedia/box_code_base.c:3202:32
    #3 0xf521c097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf5216d5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf526cced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf526cced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf526bbc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf527f3f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf5298061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf39f9ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow (/root/harness/parser32.asan+0x4dc1df) in __asan_memset
Shadow bytes around the buggy address:
  0x3e100120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e100130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e100140: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e100150: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e100160: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e100170: 00 00 00[fa]fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e100180: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e100190: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e1001a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e1001b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e1001c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==262==ABORTING
```

CVE-2021-21850 - “trun” decoder

When the library encounters an atom using the “trun” FOURCC code, it will use the following function to parse it. This function will read a number of flags in order to determine whether certain fields exist, and thus these flags will need to be identified in order to determine how to process the atom. At [65], the function will read a 32-bit integer from the atom in order to determine the number of samples, and then use the flags that were read by the `gf_isom_full_box_read` in order to determine which other fields need to be read. After the function performs an insufficient check of the number of samples against the 64-bit atom size, the function will take the product of the 32-bit number of samples and the size of the `GF_TrunEntry` structure. As the product of a 32-bit integer and another number greater than 0 will be more than 32-bits, this will result in an integer overflow which can cause the `gf_malloc` function at [66] to return a zero-sized buffer. Later at [67], when the function uses the flags in order to determine how to populate the array, the function will write data outside the bounds of the undersized array resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:7337
GF_Err trun_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i;
    GF_TrackFragmentRunBox *ptr = (GF_TrackFragmentRunBox *)s;
...
    ISOM_DECREASE_SIZE(ptr, 4);
    ptr->sample_count = gf_bs_read_u32(bs);                                     // [65] read 32-bit number of samples
...
    if (! (ptr->flags & (GF_ISOM_TRUN_DURATION | GF_ISOM_TRUN_SIZE | GF_ISOM_TRUN_FLAGS | GF_ISOM_TRUN_CTS_OFFSET) ) ) {
...
    } else {
        //if we get here, at least one flag (so at least 4 bytes) is set, check size
        if (ptr->sample_count * 4 > ptr->size) {                                // [66] check against 64-bit atom size
            ISOM_DECREASE_SIZE(ptr, ptr->sample_count*4);
        }
        ptr->samples = gf_malloc(sizeof(GF_TrunEntry) * ptr->sample_count);     // [66] allocate space for number of samples
        if (!ptr->samples) return GF_OUT_OF_MEM;
        ptr->sample_alloc = ptr->nb_samples = ptr->sample_count;
        //memset to 0 upfront
        memset(ptr->samples, 0, ptr->sample_count * sizeof(GF_TrunEntry));

        //read each entry (even though nothing may be written)
        for (i=0; i<ptr->sample_count; i++) {
            u32 trun_size = 0;
            GF_TrunEntry *p = &ptr->samples[i];

            if (ptr->flags & GF_ISOM_TRUN_DURATION) {
                p->Duration = gf_bs_read_u32(bs);                               // [67] read into undersized array
                trun_size += 4;
            }
...
        }
    }
...
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x8000000. When this value is multiplied by the size of the `GF_TrunEntry`, the value will be 0x100000000 which when truncated to 32-bits results in a zero-sized allocation being made. The atom's parser will proceed by writing 32-bit integers directly into this zero-sized buffer causing the buffer overflow.

```
=================================================================
==272==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200750 at pc 0xf512c419 bp 0xffd7b898 sp 0xffd7b890
WRITE of size 4 at 0xf1200750 thread T0
    #0 0xf512c418 in trun_box_read /root/src/isomedia/box_code_base.c:7390:17
    #1 0xf5283097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf527dd5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf52d3ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf52d3ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf52d2bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52e63f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf52ff061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3a60ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200751 is located 0 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf512a130 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf512a130 in trun_box_read /root/src/isomedia/box_code_base.c:7378:18
    #3 0xf5283097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf527dd5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf52d3ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf52d3ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf52d2bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52e63f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf52ff061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3a60ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:7390:17 in trun_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==272==ABORTING
```

CVE-2021-21851 - “csgp” decoder sample group description indices

As previously mentioned, the following `csgp_box_read` function is used to parse an atom that uses the “csgp” FOURCC code. This parser starts out by reading an 8-bit integer, followed by a 24-bit integer at [68] which contains flags describing the sizes of certain fields within the atom. At [69], the function will use the `get_size_by_code` implementation in order to determine how many bits are used for a dependant field. Depending on the bits that are set within the flags, an integer may use 4, 8, 16, or 32-bits for its value. At [70], the number of bits described by the “pattern_size” variable is used to read a length from an individual pattern within the atom. If the “pattern_size” field has all of its bits set, this will result in the length being stored with 32-bits. After reading the length, at [71], the function will use the product of the length and the size of the `u32` type in order to perform an allocation. If the size of the length field is 32-bits, then its product will result in more than 32-bits causing an integer overflow. Due to this integer-overflow, this allocation can be made to be smaller than required by the parser. After reading each individual pattern, the function will iterate through each pattern again in order to load the indices that were specified by the length. At [72] the inner-most loop will read an integer of “index_size” bits, and then store them into the array that was allocated for each index. Due to the integer-overflow, this array will be undersized and thus this loop will write outside of its bounds resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:12213
GF_Err csgp_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i, bits, gidx_mask;
    Bool index_msb_indicates_fragment_local_description, grouping_type_parameter_present;
    u32 pattern_size, scount_size, index_size;
    GF_CompactSampleGroupBox *ptr = (GF_CompactSampleGroupBox *)s;

    ISOM_DECREASE_SIZE(ptr, 8);
    ptr->version = gf_bs_read_u8(bs);
    ptr->flags = gf_bs_read_u24(bs);                                                                    // [68] read 24-bits flags
...

    pattern_size = get_size_by_code( ((ptr->flags>>4) & 0x3) );                                         // [69] | use 2-bits of
flags to determine pattern integer size
    scount_size = get_size_by_code( ((ptr->flags>>2) & 0x3) );
    index_size = get_size_by_code( (ptr->flags & 0x3) );

...
    ptr->patterns = gf_malloc(sizeof(GF_CompactSampleGroupPattern) * ptr->pattern_count);
    if (!ptr->patterns) return GF_OUT_OF_MEM;

    bits = 0;
    for (i=0; i<ptr->pattern_count; i++) {
        ptr->patterns[i].length = gf_bs_read_int(bs, pattern_size);                                     // [70] read pattern_size
bits integer for length
        ptr->patterns[i].sample_count = gf_bs_read_int(bs, scount_size);
...
        ptr->patterns[i].sample_group_description_indices = gf_malloc(sizeof(u32) * ptr->patterns[i].length);   // [71] allocate memory
using length
        if (!ptr->patterns[i].sample_group_description_indices) return GF_OUT_OF_MEM;
    }
    bits = 0;
    gidx_mask = ((u32)1) << (index_size-1);
    for (i=0; i<ptr->pattern_count; i++) {
        u32 j;
        for (j=0; j<ptr->patterns[i].length; j++) {
            u32 idx = gf_bs_read_int(bs, index_size);                                                   // [72] read index_size bit
integer from atom
...
            ptr->patterns[i].sample_group_description_indices[j] = idx;                                 // [72] write into
undersized array
            bits += index_size;

            if (! (bits % 8)) {
                bits/=8;
                ISOM_DECREASE_SIZE(ptr, bits);
                bits=0;
            }
        }
    }
...
    return GF_OK;
}
|
src/isomedia/box_code_base.c:12206
u32 get_size_by_code(u32 code)
{
    if (code==0) return 4;
    if (code==1) return 8;
    if (code==2) return 16;
    return 32;
}
```

Crash Information

The provided proof-of-concept sets the 2-bits for the “index_size” so that all integers are 32-bits. Afterwards, the length for the first pattern is set to 0x40000000. When this value is multiplied by the size of a `u32` type, this will result in the value 0x100000000 which when truncated will cause a zero-sized allocation to be made. After reading each individual pattern, and allocating the necessary “sample_group_description_indices” array, the following loop will be used to read “index_size” integers into the zero-sized buffer for the current pattern. As the provided proof-of-concept sets all the bits for the “index_size”, this will result in a 32-bit integer being used to write outside the bounds of the zero-sized buffer.

```
=================================================================
==2682==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf2000730 at pc 0xf59a6c02 bp 0xffde1f88 sp 0xffde1f80
WRITE of size 4 at 0xf2000730 thread T0
    #0 0xf59a6c01 in csgp_box_read /root/src/isomedia/box_code_base.c:12276:57
    #1 0xf5a604e4 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf5a604e4 in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf5aad638 in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf5aad638 in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf5aad638 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf5abc6f2 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf5acebf1 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf4857ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf2000731 is located 0 bytes to the right of 1-byte region [0xf2000730,0xf2000731)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf59a51d8 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf59a51d8 in csgp_box_read /root/src/isomedia/box_code_base.c:12261:55
    #3 0xf5a604e4 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf5a604e4 in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf5aad638 in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf5aad638 in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf5aad638 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf5abc6f2 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf5acebf1 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf4857ee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:12276:57 in csgp_box_read
Shadow bytes around the buggy address:
  0x3e400090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e4000a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e4000b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e4000c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e4000d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e4000e0: fa fa fa fa fa fa[01]fa fa fa 00 04 fa fa fd fa
  0x3e4000f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e400100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e400110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e400120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e400130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==2682==ABORTING
```

CVE-2021-21852 - “stss” decoder

The following function is used by the library when parsing an atom using the “stss” FOURCC code. This function will first read a 32-bit number from the atom at [73] for the number of entries. After reading the number of entries, the function will then check them against the 64-bit atom size. Due to the atom size being 64-bits, this check is insufficient. Afterwards, the function will store the number of entries into a 32-bit field, and then use it to perform a calculation for allocating memory from the heap. At [74], the number of entries will be multiplied by the size of the `u32` type. As any number other than zero multiplied by a 32-bit number can result in a value larger than 32-bits, this will cause an integer overflow which will result in a smaller size being allocated for the array. At [75], the library will proceed to read integers from the atom into the undersized array resulting in a heap-based buffer overflow.

```
src/isomedia/box_code_base.c:5463
GF_Err stss_box_read(GF_Box *s, GF_BitStream *bs)
{
    u32 i;
    GF_SyncSampleBox *ptr = (GF_SyncSampleBox *)s;

    ISOM_DECREASE_SIZE(ptr, 4);
    ptr->nb_entries = gf_bs_read_u32(bs);                                   // [73] read number of entries
    if (ptr->size / 4 <  ptr->nb_entries) {                                 // [73] check them against 64-bit size
        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("[iso file] Invalid number of entries %d in stss\n", ptr->nb_entries));
        return GF_ISOM_INVALID_FILE;
    }

    ptr->alloc_size = ptr->nb_entries;                                      // [74] store to 32-bit field
    ptr->sampleNumbers = (u32 *) gf_malloc( ptr->alloc_size * sizeof(u32));     // [74] multiply by u32 for allocation size
    if (ptr->sampleNumbers == NULL) return GF_OUT_OF_MEM;

    for (i = 0; i < ptr->nb_entries; i++) {
        ptr->sampleNumbers[i] = gf_bs_read_u32(bs);                         // [75] read 32-bit integers into undersized array
    }
    return GF_OK;
}
```

Crash Information

The provided proof-of-concept sets the number of entries to 0x40000000. When this count is multiplied by the size of a `u32` type, this will result in a zero-sized allocation being made. The loop will then overflow the zero-sized buffer by reading 32-bit integers from the atom into it.

```
=================================================================
==237==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf1200750 at pc 0xf50c328b bp 0xffbc2ca8 sp 0xffbc2ca0
WRITE of size 4 at 0xf1200750 thread T0
    #0 0xf50c328a in stss_box_read /root/src/isomedia/box_code_base.c:5480:25
    #1 0xf5251097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #2 0xf524bd5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #3 0xf52a1ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #4 0xf52a1ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #5 0xf52a0bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #6 0xf52b43f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #7 0xf52cd061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #8 0x512f6a in main /root/harness/parser.c:50:13
    #9 0xf3a2eee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)
    #10 0x4621e5 in _start (/root/harness/parser32.asan+0x4621e5)

0xf1200751 is located 0 bytes to the right of 1-byte region [0xf1200750,0xf1200751)
allocated by thread T0 here:
    #0 0x4dcb75 in malloc (/root/harness/parser32.asan+0x4dcb75)
    #1 0xf50c2b28 in gf_malloc /root/src/utils/alloc.c:150:9
    #2 0xf50c2b28 in stss_box_read /root/src/isomedia/box_code_base.c:5476:31
    #3 0xf5251097 in gf_isom_box_read /root/src/isomedia/box_funcs.c:1808:9
    #4 0xf524bd5d in gf_isom_box_parse_ex /root/src/isomedia/box_funcs.c:265:14
    #5 0xf52a1ced in gf_isom_parse_root_box /root/src/isomedia/box_funcs.c:38:8
    #6 0xf52a1ced in gf_isom_parse_movie_boxes_internal /root/src/isomedia/isom_intern.c:318:7
    #7 0xf52a0bc0 in gf_isom_parse_movie_boxes /root/src/isomedia/isom_intern.c:777:6
    #8 0xf52b43f7 in gf_isom_open_file /root/src/isomedia/isom_intern.c:897:19
    #9 0xf52cd061 in gf_isom_open /root/src/isomedia/isom_read.c:509:11
    #10 0x512f6a in main /root/harness/parser.c:50:13
    #11 0xf3a2eee4 in __libc_start_main (/lib32/libc.so.6+0x1eee4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/src/isomedia/box_code_base.c:5480:25 in stss_box_read
Shadow bytes around the buggy address:
  0x3e240090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e2400d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3e2400e0: fa fa fa fa fa fa fa fa fa[01]fa fa fa fd fa
  0x3e2400f0: fa fa 00 fa fa fa 00 04 fa fa fa fa fa fa fa fa
  0x3e240100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3e240130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==237==ABORTING
```

Timeline

2021-06-24 - Vendor Disclosure

2021-08-11 - Vendor patched

2021-08-16 - Public Release

CREDIT

Discovered by a member of Cisco Talos.
