# Stagefright Vulnerability Analysis and Exploitability

## Executive Summary

The Android library responsible for the parsing of multimedia files, known as *libStageFright*, contained a vulnerable function that when exploited could lead to remote code execution and denial of service attacks. Additionally, because the vulnerability occurred in the parsing of the file and not necessarily the execution of it, a video sent via MMS (text message) could be theoretically used as one of the methods of attack.

## Vulnerability Details

The vulnerability in the libstagefright involved the parsing of MPEG­4 files, or more specifically the ‘tx3g’ atom, which is used to embed subtitles(timed-text) into multimedia files. The vulnerable code bellow does not properly restrict size addition, which allows remote attackers to execute arbitrary code or cause a denial of service (integer overflow and memory corruption) via a specially crafted MPEG-4 tx3g atom.

Vulnerable Code Snippet:

**case** FOURCC**(**'t'**,** 'x'**,** '3'**,** 'g'**):**

**{**

uint32\_t type**;**

const void **\***data**;**

size\_t size **=** 0**;**

**if** **(!**mLastTrack**->**meta**->**findData**(**

kKeyTextFormatData**,** **&**type**,** **&**data**,** **&**size**))** **{**

size **=** 0**;**

**}**

uint8\_t **\***buffer **=** **new** **(**std**::**nothrow**)** uint8\_t**[**size **+** chunk\_size**];**

**if** **(**buffer **==** **NULL)** **{**

**return** ERROR\_MALFORMED**;**

**}**

**if** **(**size **>** 0**)** **{**

memcpy**(**buffer**,** data**,** size**);**

**}**

// NOTE: EXPLOITABLE

**if** **((**size\_t**)(**mDataSource**->**readAt**(\***offset**,** buffer **+** size**,** chunk\_size**))**

**<** chunk\_size**)** **{**

**delete[]** buffer**;**

buffer **=** **NULL;**

**\***offset **+=** chunk\_size**;**

**return** ERROR\_IO**;**

**}**

mLastTrack**->**meta**->**setData**(**

kKeyTextFormatData**,** 0**,** buffer**,** size **+** chunk\_size**);**

**delete[]** buffer**;**

**\***offset **+=** chunk\_size**;**

**break;**

**}**

When handling the addition of ***size*** and ***chunk\_size***, there is no check to see if the result is larger than 2^32, causing an integer overflow to occur.

uint8\_t **\***buffer **=** **new** **(**std**::**nothrow**)** uint8\_t**[**size **+** chunk\_size**];**

The result is then truncated to fit the 32-bit value. When allocating memory based on that result, it creates buffer allocations with less than the necessary memory. When *chunk\_size* worth of data from the ‘tx3g’ atom is read into the buffer, which is allocated on the heap, it will overrun the buffer’s boundary and start writing into adjacent memory. This is all executed when *memcpy* is called here:

**if** **(**size **>** 0**)** **{**

memcpy**(**buffer**,** data**,** size**);**

**}**

The *memcpy* method copies data into the destination (first parameter) from the object pointing to the source of the data (second parameter) by counting the number of bytes to copy (third parameter). The memcpy method also returns the address of the destination. Because the third parameter will be larger than the size of the data buffer, it results in a buffer over-read. If the MP4 file is crafted in such a way that the data that is copied into the heap past the boundaries of the buffer is arbitrary code, the attacker may be able to exploit this vulnerability in addition to other existing vulnerabilities to execute the code.

## Exploitability

Assumptions:

* DEP is enabled
* ASLR is not enforced
* CFI is not enabled
* No integer overflow sanitization