# Stagefright: Vulnerabilities, Bug Fixing, Preventive Measures in Android

Walunj Swapnil K.
MCA Research Scholar,
Mumbai, India

Yadav Anil H.
MCA Research Scholar,
Mumbai, India

## ABSTRACT
This paper presents the threats to the Android by using the Loop Hole in the Operating System of Android. Stagefright is the group of software bugs that affect versions 2.2 ("Froyo") and newer of the Android operating system, allowing an attacker to perform arbitrary operations on the victim's device through remote code execution and privilege escalation. Stagefright is believed to be the worst Android vulnerability yet discovered. All devices running Android versions Froyo 2.2 to Lollipop 5.1.1 are affected, which are used by approximately 95% of all Android devices, by nearly 1 billion people. Hackers only need to know your phone number to infect your device.

## Keywords
loop hole; operating system; Froyo; infect

## 1. INTRODUCTION
In July 2015, security company Zimperium announced that it had discovered a "unicorn" of a vulnerability inside the Android operating system. More details were publicly disclosed at the BlackHat conference in early August — but not before headlines declaring that nearly a billion Android devices could potentially be taken over without their users even knowing it. The original vulnerability was found by Joshua Drake from Zimperium , affecting Android versions 1.0 and above.[4] "Stagefright" is the nickname given to a potential exploit that lives fairly deep inside the Android operating system itself. The gist is that a video sent via MMS (text message) could be theoretically used as an avenue of attack through the *libStageFright* mechanism (thus the "Stagefright" name), which helps Android process video files. Many text messaging apps — Google's Hangouts app was specifically mentioned — automatically process that video so it's ready for viewing as soon as you open the message, and so the attack theoretically could happen without you even knowing it.

Because *libStageFright* dates back to Android 2.2, hundreds of millions of phones contain this flawed library.

In July 2015, Evgeny Legerov, a Moscow-based security researcher, announced that he found at least two similar heap overflow zero-day vulnerabilities in the Stagefright library, claiming at the same time that the library has been already exploited for a while. Legerov also confirmed that the vulnerabilities he discovered become unexploitable by applying the patches Drake submitted to Google.[5][6]

## 2. THE VULNERABILITIES
There are various diverse bugs making up Stagefright, and these have their own particular CVE [Common Vulnerabilities and Exposures] numbers for following:

- CVE-2015-1538
- CVE-2015-1539
- CVE-2015-3824
- CVE-2015-3826
- CVE-2015-3827
- CVE-2015-3828
- CVE-2015-3829

Shockingly the patches which are accessible are not connected straightforwardly to each CVE (as they ought to be), so this will be somewhat untidy to clarify [1],[2].

### 2.1 [CVE-2015-1538]
In the MPEG4 taking care of code, the 3GPP metadata (the stuff that depicts the configuration and other additional information, when a video is in 3GPP arrangement) taking care of code is surrey. It didn't dismiss metadata, where the information was too much extensive, rather just checking on the off chance that it was too little. This implied it was workable for an assailant to create an "adjusted" or "ruined" document, which would have a more drawn out metadata divide than it ought to.

One of the enormous difficulties in composing code to handle "untrusted" information (i.e. from a client or from whatever other sort of place outside to your code) is taking care of variable-length information. Recordings are a flawless case of this. The product needs to allot memory powerfully, contingent upon what's going on.

For this situation, a support is made as a pointer to some memory, yet the length of the exhibit it focuses to was one component too short. The metadata was then perused into this exhibit, and it was conceivable to have the last section in this cluster not be "invalid" (or zero). It's essential the last thing in the cluster is zero, since that is the manner by which the product tells the exhibit is done. By having the capacity to make the last esteem non-zero (since the cluster was conceivably one component too little), the malignant code could be perused by another piece of code, and read in a lot of information. As opposed to stop toward the end of this esteem, it could continue perusing into other information it shouldn't read [4],[6].

```
 if (size < 4) {
     return ERROR_MALFORMED;
   }
   uint8_t *buffer = new (std::nothrow) uint8_t[size];
   if (buffer == NULL) {
     return ERROR_MALFORMED;
   }
if (isUTF8) {
        mFileMetaData->setCString(metadataKey, (const char
*)buffer + 6);
```

```
    }
```

changed to,

```
 if (size < 4 || size == SIZE_MAX) {
    return ERROR_MALFORMED;
  }
  uint8_t *buffer = new (std::nothrow) uint8_t[size + 1];
  if (buffer == NULL) {
    return ERROR_MALFORMED;
  }
if (isUTF8) {
        buffer[size] = 0;
      mFileMetaData->setCString(metadataKey, (const char *)buffer + 6);
    }[10]
```

## 2.2 [CVE-2015-1539]

The most limited conceivable "size" of the metadata ought to be 6 bytes, because of it being an UTF-16 string. The code takes the number esteem estimate, and subtracts 6 from it. On the off chance that this esteem was under 6, the subtraction would "sub-current" and wrap around, and we'd wind up with a huge number. Suppose you can just check from 0 to 65535, for instance. On the off chance that you take the number 4, and subtract 6, you can't go underneath zero. So, you do a reversal to 65535 and number from that point. That is what's going on here!

In the event that a length of under 6 was gotten, it could prompt edges being erroneously decoded, since the byteswap procedure utilizes the variable len16, whose esteem is acquired from an estimation starting with (size-6). This could make a much greater swap operation happen than planned, which would change values in the casing information in a sudden way [9].

```
if (metadataKey > 0) {
    bool isUTF8 = true; // Common case
    char16_t *framedata = NULL;
    int len16 = 0; // Number of UTF-16 characters
    // smallest possible valid UTF-16 string w BOM: 0xfe 0xff 0x00 0x00
    if (size - 6 >= 4) {
      len16 = ((size - 6) / 2) - 1; // don't include 0x0000 terminator
      framedata = (char16_t *)(buffer + 6);
      if (0xfffe == *framedata) {
        // endianness marker (BOM) doesn't match host endianness
        for (int i = 0; i < len16; i++) {
          framedata[i] = bswap_16(framedata[i]);
        }
        // BOM is now swapped to 0xfeff, we will execute next block too
      }
```

has changed to,

```
if (metadataKey > 0) {
    bool isUTF8 = true; // Common case
    char16_t *framedata = NULL;
    int len16 = 0; // Number of UTF-16 characters
    // smallest possible valid UTF-16 string w BOM: 0xfe 0xff 0x00 0x00
    if (size < 6) {
      return ERROR_MALFORMED;
    }
    if (size - 6 >= 4) {
      len16 = ((size - 6) / 2) - 1; // don't include 0x0000 terminator
      framedata = (char16_t *)(buffer + 6);
      if (0xfffe == *framedata) {
        // endianness marker (BOM) doesn't match host endianness
        for (int i = 0; i < len16; i++) {
          framedata[i] = bswap_16(framedata[i]);
        }
        // BOM is now swapped to 0xfeff, we will execute next block too
      }[10]
```

## 2.3 [CVE-2015-3824]

A biggie! This one is frightful. There's the correct inverse of this last issue – a whole number flood, where a whole number can get "too huge". In the event that you achieve 65535 (for instance) and can't tally any higher, you would move around, and go to 0 next!

On the off chance that you are dispensing memory in view of this (which is the thing that Stagefright is doing!), you would wind up with very little memory allotted in the exhibit. At the point when information was put into this, it would conceivably overwrite random information with information the vindictive record maker controlled.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);
        }
```

has changed to,

```
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;
        }
        if (SIZE_MAX - chunk_size <= size)
            return ERROR_MALFORMED;
        uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
        if (buffer == NULL) {
            return ERROR_MALFORMED;
        }
        if (size > 0) {
            memcpy(buffer, data, size);
        }
```

## 2.4 [CVE-2015-3826]

Another terrible one! Fundamentally the same as the last one – another whole number flood, where a cluster (utilized as a support) would be made too little. This would permit disconnected memory to be overwritten, which is again terrible. Somebody who can compose information into memory can degenerate other information that is disconnected, and conceivably utilize this to have some code they control be controlled by your telephone.

```
case FOURCC('c', 'o', 'v', 'r'):
    {
        *offset += chunk_size;


        if (mFileMetaData != NULL) {
            ALOGV("chunk_data_size = %lld and data_offset = %lld",
                    chunk_data_size, data_offset);
            sp<ABuffer> buffer = new ABuffer(chunk_data_size + 1);
            if (mDataSource->readAt(
                data_offset, buffer->data(), chunk_data_size) != (ssize_t)chunk_data_size) {
                return ERROR_IO;
            }
            const int kSkipBytesOfDataBox = 16;
            mFileMetaData->setData(
                kKeyAlbumArt, MetaData::TYPE_NONE,
                buffer->data() + kSkipBytesOfDataBox, chunk_data_size - kSkipBytesOfDataBox);
        }
```

changed to,

```
case FOURCC('c', 'o', 'v', 'r'):
    {
        *offset += chunk_size;
        if (mFileMetaData != NULL) {
            ALOGV("chunk_data_size = %lld and data_offset = %lld",
                    chunk_data_size, data_offset);
            if (chunk_data_size >= SIZE_MAX - 1) {
                return ERROR_MALFORMED;
            }
            sp<ABuffer> buffer = new ABuffer(chunk_data_size + 1);
            if (mDataSource->readAt(
                data_offset, buffer->data(), chunk_data_size) != (ssize_t)chunk_data_size) {
                return ERROR_IO;
            }
            const int kSkipBytesOfDataBox = 16;
            mFileMetaData->setData(
                kKeyAlbumArt, MetaData::TYPE_NONE,
                buffer->data() + kSkipBytesOfDataBox, chunk_data_size - kSkipBytesOfDataBox);
        }
```

[10]

## 2.5 [CVE-2015-3827]

Very like these last ones. A variable is utilized when skirting some memory, and this could be made negative amid a subtraction (like above). This would bring about an extensive "skip" length, flooding a cushion, offering access to memory that shouldn't be gotten to.

```
const int kSkipBytesOfDataBox = 16;
            mFileMetaData->setData(
                kKeyAlbumArt, MetaData::TYPE_NONE,
                buffer->data() + kSkipBytesOfDataBox, chunk_data_size - kSkipBytesOfDataBox);
        }
```

changed to,

```
const int kSkipBytesOfDataBox = 16;
```

```
if (chunk_data_size <= kSkipBytesOfDataBox) {
   return ERROR_MALFORMED;
}
mFileMetaData->setData(
   kKeyAlbumArt, MetaData::TYPE_NONE,
   buffer->data() + kSkipBytesOfDataBox, chunk_data_size - kSkipBytesOfDataBox);
}[10]
```

There are additionally a few (conceivably) related fixes that hope to have made it into [Android] 5.1 also.

This adds checks to stop issues with a past security alter to include limits checks, which can itself be flooded. In C, numbers that can be spoken to as a marked int are put away as a marked int. Else they stay unaltered amid operations. In these checks, a few whole numbers could have been made marked (as opposed to unsigned), which would lessen their most extreme esteem later on, and take into account a flood to occur.

Some more whole number undercurrents (where numbers are too low, and after that subtraction is completed on those numbers, permitting them to go negative). This again prompts a vast number, as opposed to a little one, and that causes an indistinguishable issue from above.

Lastly, another number flood. Same as some time recently, it's going to be utilized somewhere else, and it could flood.

## 3. GOOGLE FIXES STAGE FRIGHT BUG

There are some basic security fixes out.

Google has altered 12 vulnerabilities influencing Android variants 4.4.4 through 6.0.1, including five appraised as "basic" – the assignment for the most noticeably bad sort of security bug.

The most genuine defencelessness in this group is a remote code execution (RCE) bug, assigned CVE-2015-6636, in Android's mediaserver part.

Mediaserver is regularly used to render remotely-provided sight and sound substance, so Google is cautioning that an aggressor could misuse the bug to run malware covered up in booby-caught media documents conveyed by means of various strategies, including email, web perusing and MMS.

Mediaserver is a "centre part of the working framework," with access to video and sound streams too having run-time benefits that outsider applications don't.

On the off chance that this sounds natural, that is likely in light of the fact that Google has now fixed 30 vulnerabilities in mediaserver since month to month Android security redesigns started in August 2015, as per InfoWorld's Fahmida Y. Rashid.

This mediaserver bug is additionally like the real powerlessness known as "Stagefright" that influenced up to 95% of Android gadgets, which could have permitted law breakers to embed malware correspondingly.

Luckily, to relieve the bug, Google has rolled out improvements to the default Android informing applications, Google Hangouts and Messenger, so that they "[no longer] consequently pass media to procedures, for example, mediaserver."

Google said it made the security redesign accessible to accomplices on 7 December 2015 "or prior."

Google and Samsung have been speedier at getting security settles out since Stagefright, yet shockingly, bearers haven't pushed out overhauls for each kind of Android gadget influenced by this most recent arrangement of vulnerabilities.

Sprint and Verizon have overhauled their Nexus 5 and 6 gadgets, as indicated by Softpedia, which additionally reports that other Android gadgets are required to get the redesigns soon, including BlackBerry PRIV, Samsung Galaxy S6, Galaxy Note 5 and "some Motorola and HTC cell phones."

When you see a warning that the redesign is prepared on your gadget, you ought to acknowledge it and move up to the most recent form of Android "wherever conceivable," Google suggests.

Until you can apply the security overhaul, be exceptionally careful about downloading or playing media records.

Try not to acknowledge media messages from obscure senders, and ensure the setting to Automatically recover MMS messages in both Hangouts and Messenger is killed.

## 4. PREVENTIVE MEASURES

To the extent we know, Android antivirus applications won't spare you from Stagefright assaults. They don't really have enough framework authorizations to block MMS messages and meddling with framework segments. Google additionally can't overhaul the Google Play Services part in Androidto alter this bug, an interwoven arrangement Google regularly utilizes when security openings appear.

To truly keep yourself from being traded off, you have to keep your informing application of decision from downloading and propelling MMS messages. When all is said in done, this implies impairing the "MMS auto-recovery" setting in its settings. When you get a MMS message, it won't consequently download — you'll need to download it by tapping a placeholder or something comparable. You won't be at hazard unless you download the MMS.

You shouldn't do this. On the off chance that the MMS is from somebody you don't have the foggiest idea, unquestionably overlook it. On the off chance that the MMS is from a companion, it would be conceivable their Phone has been bargained if a worm begins to take off. It's most secure to never download MMS messages if your telephone is helpless.

To cripple MMS message auto-recovery, take after the proper strides for your informing application.

- Messaging (incorporated with Android): Open Messaging, tap the menu catch, and tap Settings. Look down to the "Interactive media (MMS) messages" segment and uncheck auto-recover."
- Messenger (by Google): Open Messenger, tap the menu, tap Settings, tap Advanced, and cripple AUTO recover."
- Hangouts (by Google): Open Hangouts, tap the menu, and explore to Settings > SMS. Uncheck "Auto recover SMS" under Advanced. (On the off chance that you don't see SMS choices here, your telephone isn't utilizing Hangouts for SMS. Impair the setting in the SMS application you use.)

- Messages (by Samsung): Open Messages and explore to More > Settings > More settings. Tap Multimedia messages and impair the "Auto recover" choice. This setting might be in an alternate spot on various Samsung gadgets, which utilize diverse renditions of the Messages application.

It's difficult to manufactured a total rundown here. Simply open up the application you use to send SMS messages (instant messages) and search for a choice that will incapacitate "auto recover" or download" of MMS messages.

- Cautioning: If you download a MMS message, you're still helpless. Also, as the Stagefright defencelessness isn't only a MMS message issue, this won't totally shield you from each sort of assault.

Don't ignore updates from Android - when you receive a notification about an update, accept it, and upgrade to the latest version of Android.

Avoid opening video and audio files you receive via text or email. Delete all messages you get, without opening it first, from any sender you do not recognize.

This recommend users disable “auto retrieve MMS” within their default messaging app’s settings, as a precautionary measure for the moment. [9]

## 5. CONCLUSION

Stagefright is a truly risky danger to all Android gadgets. The main thing you have to do is to know about the danger. Stagefright has been only a reminder towards Android and its issue of discontinuity and additionally redesigns. as of now it is obscure if the powerlessness has yet been abused in "nature." Some telephones have as of now been fixed. It highlights how there is no reasonable system by method for which such basic fixes can be taken off in a convenient way to various gadgets. There are applications which can test whether your gadget is at danger of Stagefright. You can essentially run the tests and after that choose whether to go for the strategies talked about above. While OEM's are attempting to take off patches for gadgets, the unforgiving truth is that the vast majority of these fixes will be constrained to late leaders as it were. Other non-leads and more established gadgets, a great deal less from littler OEM's will proceed on being presented to the like of Stagefright.

## 6. REFERENCES

[1] https://en.wikipedia.org/wiki/Stagefright_(bug)

[2] https://www.avast.com/faq.php?article=AVKB230

[3] http://www.androidcentral.com/stagefright

[4] https://www.exploit-db.com/docs/39527.pdf

[5] "How to Protect from StageFright Vulnerability". zimperium.com.

[6] Thomas Fox-Brewster (July 30, 2015). "Russian 'Zero Day' Hunter Has Android Stagefright Bugs Primed For One-Text Hacks". Forbes.

[7] http://blog.trendmicro.com/trendlabs-security-intelligence/android-security-update-includes-fix-for-stagefright-vulnerabilities-discovered-by-trend-micro/

[8] https://www.engadget.com/2016/03/19/reliable-stagefright-android-exploit/

[9] https://blog.avast.com/2016/01/07/android-security-updates-roll-out-to-fight-stagefright-type-bug/

[10] http://www.xda-developers.com/stagefright-explained-the-exploit-that-changed-android/