Android Stagefright

Logan Bair

Department of Computer & Information Sciences   
College of Science & Mathematics   
Towson University

Lbair89@gmail.com

Mary Snyder

Department of Computer & Information Sciences  
College of Science & Mathematics  
Towson University

msnyde8@students.towson.edu

**ABSTRACT**

In this paper, we describe our approach for implementing the known exploit for the Android operating system (OS) Stagefright vulnerability on numerous releases of the Android OS ranging from versions 4.0-5.1.

**Categories and Subject Descriptors**

K.6.5 [Management of Computing Information Systems]: Security and Protection

**Keywords**

Security; Stagefright; Zimperium Research Labs (zLABS); Android Open Source Project (AOSP); Short Message Service (SMS); Multimedia Messaging Service (MMS); Java; C++; Memory Corruption; Buffer Overflow; Integer Overflow; Android Studio; Android Emulator; Genymotion; MPEG4

# INTRODUCTION

Security researchers at Zimperium Research Labs (zLABS) recently discovered an Android OS vulnerability. To exploit this vulnerability, the attacker merely needs access to the phone’s mobile number and can gain remote code execution privileges. The Android Open Source Project (AOSP) contains a media library named Stagefright at its core which is responsible for processing of all multimedia files (videos, audio, and documents).

The vulnerability found in Stagefright is especially dangerous because it requires no interaction from the device user. Multimedia files are pre-processed by Stagefright as they are received to insure they are ready for use/viewing when the device user needs them. This pre-processing causes any malicious code embedded in the multimedia file to be executed even before the multimedia file is accessed by the device user.

We will show which parts of the Android OS are affected by the vulnerability and the cause of the vulnerability. We will compare what we learned in our class lectures to the information we gather from our project research to show a real would example of the topics we have discussed. The outcome of our research project will be to show how the Stagefright vulnerability is executed in a virtual environment as well as on physical devices. If time remains, we hope to expand on the vulnerability making the exploit payload into a self-propagating worm, which would use the infected device’s available contact information to pass on the exploit payload.

# A DEEPER LOOK INTO THE STAGEFRIGHT VULNERABILITY

## Discovery

The VP of Platform Research and Exploitation at Zimperium zLABS, Joshua J. Drake discovered the vulnerability after delving into the inner workings of the Android OS. Zimperium zLABs released a video shortly after the discovery with a demonstration of the exploit in action. Zimperium zLABS continued their research into the vulnerability were able to create tools to test if devices were susceptible to the vulnerability in addition to reproduction of the exploit payload. Many carriers for the devices asked that Zimperium zLABS delay the release of the exploit shown in the video due to the gravity of the exploit and the potential damage, which they agreed to with one caveat: if an exploit appeared from another source, their exploit would be released for companies to use for testing and securing purposes.

## Cause and Effect

When dealing with multimedia files time is of the essence and speed of processing is a high priority. As such, the Stagefright library was not written in Java or a similar more memory-safe language, but instead was based in C++, which is more apt to suffer from memory corruption.

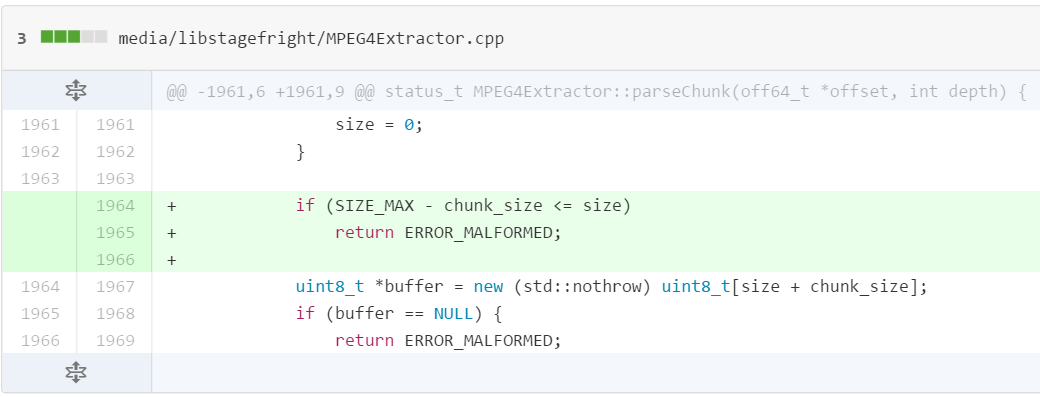
The possible memory corruption opens the window for an attacker to not only potentially install spyware onto Android devices, but even more lethal, open a shell on the device and have access to the device itself. Multiple remote code execution vulnerabilities have been identified through the Stagefright vulnerability exploits.

## Examination of Code

There were two major points identified in the Stagefright code, which were culprits for the potential vulnerability, both in the MPEG4Extractor code.

First, when handling MPEG4 the size and chunk size variables are not properly checked for size limitations. When either of these variables is larger than 232 an integer overflow occurs [3].

**Figure 1. Integer Overflow[3].**



Second, certain fields in the 3GPP video metadata are vulnerable, this time to buffer overflow attacks. The metadata processed may not be null terminated and may cause reading out of bounds [4].



**Figure 2. Buffer Overflow[4].**

## Means for Exploitation

There are various method to exploit the Stagefright vulnerability, with the main and most potentially dangerous being through a specially crafted media file delivered via Multimedia Messaging Service (MMS). In a statement by Zimperium zLABS, the devices were vulnerable in whatever way the file could be place onto the system. This would include the exploit payload being downloaded onto the device through e-mail, copied onto the device via SD card, or be placed on the device through proximity using NFC.

## Severity

The Stagefright attack does not require the victim to take any action for the exploit to be successful. Unlike other attacks where the user clicks a malicious link or the victim opens a suspicious document, this attack can even take place while the victim is not actively using their device. With the correctly crafted attack payload, the attacker can even remove any/all signs of the attack before the user even touches their device again or knows something malicious has occurred. This leaves the victim oblivious to the fact their device has been compromised. Experts estimate the Stagefright vulnerability effects 95% of all Android devices, an estimated 950 million.

# TECHNIQUES FOR REPRODUCTION

## Environment

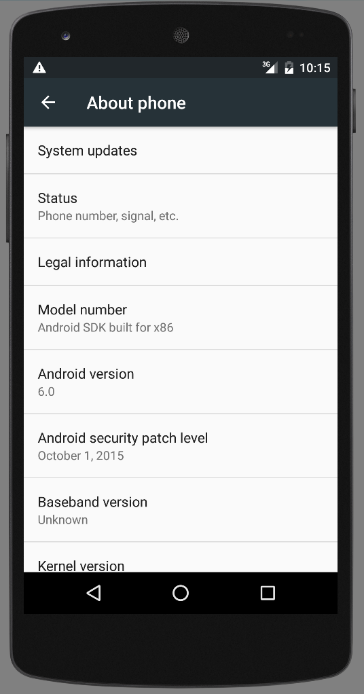
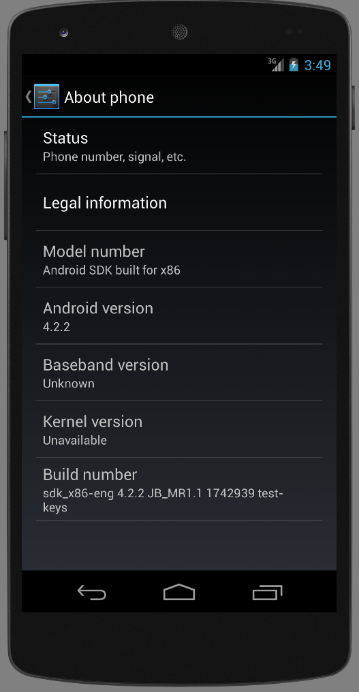
### Android x86 OS

We initially planned to use the Android x86 version of the OS for replicating the Stagefright exploit; however we were unable to find a way to emulate sending of text messages, specifically MMS messages with this version of the OS. Since MMS is one of the best and most likely way this exploit would be delivered to the public, we decided to find an environment that would allow us to deliver the exploit in this manner.

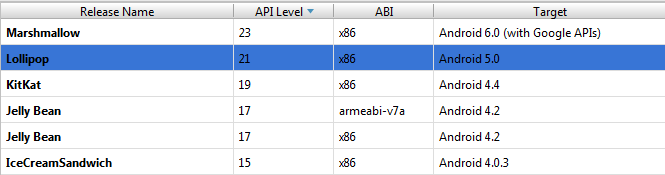
### Android Studio

After our issues with the Android x86 OS we tried Android Studio to spin up virtual instance of Android on PC. Android Studio allows the user to install and load any version of the Android OS on many different phone emulators. According to our research, the Android Emulator version allows for sending of Short Message Service (SMS) text messages through the command “sms send <senderPhoneNum> <textMsgBody>”.

Unfortunately, after playing around with the Android Emulator we discovered it would not meet our needs. While it is able to send SMS text messages, it is unable to send MMS messages. Since multimedia files are transmitted as MMS messages, not SMS messages, we would have to find another way to transmit the MMS message in this tool.



**Figure 3. Android Studio example.**



Before moving from Android Studio to yet another tool, we also looked at the possibility of using the Andoid app Google Hangouts to transmit the malicious MMS. Google Hangouts comes standard on the Android OS and allow for SMS/MMS integration; however, even our attempts using Google Hangouts in Android Studio were unsuccessful and we were forced to find another way to deliver the exploit to the device.

### VMs and Physical Devices

We finally employed the use of Genymotion VMs (versions 4.1-4.4 as well as less vulnerable versions 4.4-5.1), VMs in the Android SDK emulator (again on vulnerable versions 4.1-4.4 as well as 4.4-5.1), and even a physical Nexus 7 tablet running 4.1. After more research we discovered our issues with using Google Hangouts was due their disabling of MP4 messages, which was the type of file we were using as an exploit payload. However, Zimperium zLABS also divulged the phone was vulnerable in whatever way the file could get onto the system, so we were able to deliver the exploit payload through other means (methods discussed previously in 2.4 Means For Exploitation).

## Schedule

Our original schedule allowed for:

1. One to two weeks of research on the Stagefright vulnerability topic
2. One week to research and create the malicious MP4 for the Stagefright exploit
3. Two weeks to work with the exploit payload to successfully execute the Stagefright vulnerability exploit
4. Three weeks to test and execute the exploit on multiple version of the OS and create the self-propagating worm with any remaining time
5. Two weeks to compile our findings and results into a final report

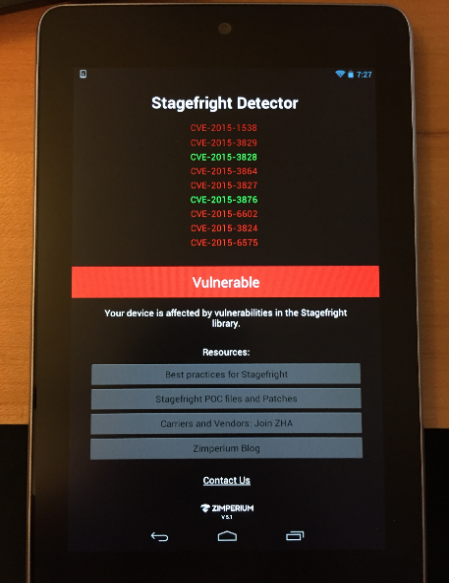
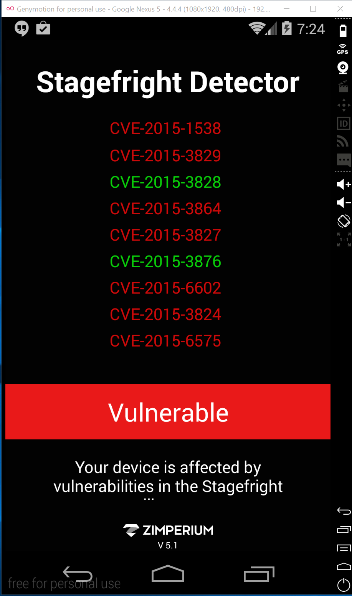
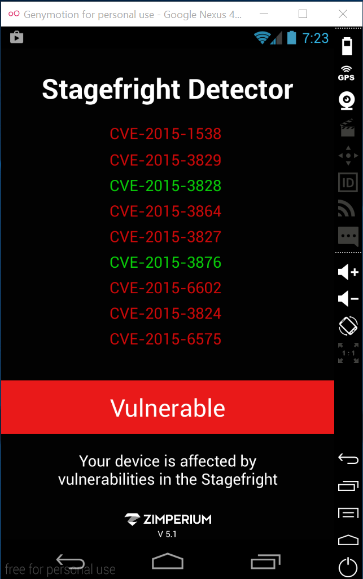
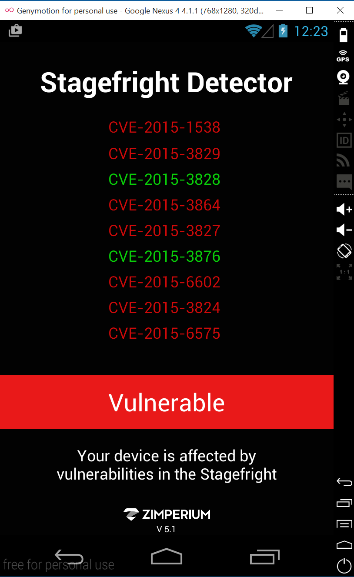
After issues with our original selection for the Android OS, we took more time than anticipated for the first two items on our schedule. To compensate for this we reduce the time to work with the execution of the exploit payload (#3) from two weeks to one week, reduced the time to test/execute (#4) from three weeks to two weeks, and reduce the time for our final report (#5) from two weeks to ten days.

When we encountered our second change in environment due to limitations of the chosen product, we unfortunately had to abandon the idea of the self-propagating worm (#4) and had to overlap the remaining time to work on both getting the vulnerability to execute (#4) and the final report (#5) simultaneously to make our deadline.

## Software Approach

### Confirmation of Vulnerability

We started by running the application released by Zimperium zLABS to test each version of Android that we were using to reproduce the exploit to see if they were indeed vulnerable to the Stagefright attack.



**Figure 4. Detector tool examples [5].**

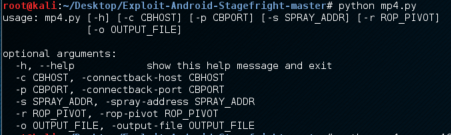
### Payload Generation

After confirming the devices were indeed vulnerable, we proceeded to download and run the script provided by Drake from Zimperium zLABS through Exploit-DB [6] used to target a specific part of the Stagefright vulnerability, cve-2015-1538. The script is coded in python and for ease of development we decided to not use the script on the test Android devices, but use a separate Kali Linux machine. Once the exploit payload was created it could then be moved to the test devices.

The script did run successfully out of the box, as downloading the source file from Exploit-DB changed the formatting for some parts of the code. Our changes were as follows:

* Add the following to the top of the script: “#coding: utf-8 a”
* Replace all ‘ symbols
* Replace all “ symbols
* Change the name of the file from 38124.py to pm4.py

After making the above changes, we were successful in generating the exploit payload using the command “python mp4.py –c 192.168.1.123 –p 4444” where the ip (192.168.1.123) and port (4444) are specific to the device/application. Running the script generates an MP4 file containing the exploit payload delivered to the phone.

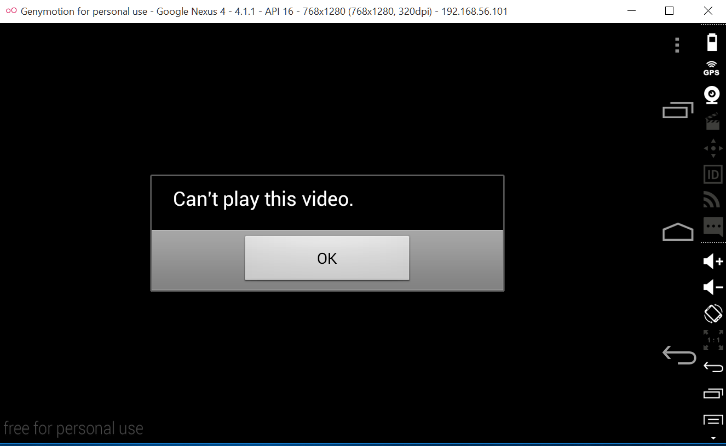


**Figure 2. Exploit payload generation.**

After the MP4 was generated, we ran the command “netcat –l –p 4444” to listen for any created shells.

# Results

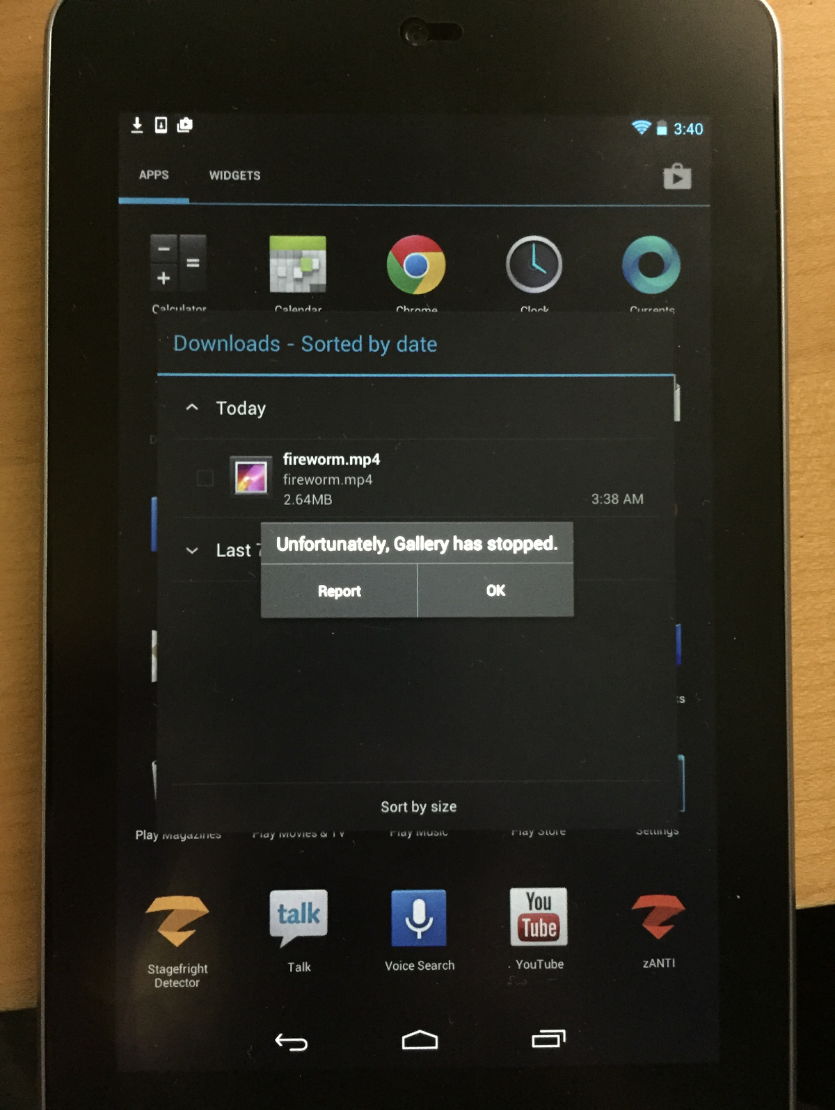
None of the attempted methods have yet been successful at opening a shell on the systems; however, on the virtual systems we received errors stating the system “Can’t play this video” and the exploit on the Nexus S running 2.3.7 completed crashed the VM (analogous to the phone restarting).



**Figure 5. Nexus 4 error message examples.**

On the physical Nexus 7 device, our results differed slightly from the VMs. Attempting to play the malicious MP4 video many times caused the same error as the VMs (“Can’t play this video”); however, on occasion the Gallery application would crash. This was a promising sign that exploit payload was interacting with the OS and in some way causing malicious things to occur.

**Figure 6. Nexus 7 device error message example.**



## Reasons Behind Partial Success

We believe the reason the exploit was not successful in completely replicating what was described by the Zimperium zLABS were details in the constructing of the malicious MP4 specific to each device. Memory addresses can be unique across the platforms and devices. This could be true in the virtual instances of the platform as well.

In addition, ASLR was introduced in Android 4 and up. It is in use on the Android devices since the service is pawned from Zygote. Using the attack vectors we attempted (file copy, e-mail, SD card emulation) it is very difficult to be successful in a brute force the attempt and still gain control of the device before it has a chance to detect an issue and force reboot itself. Using MMS, an attacker could continue sending the message in rapid succession in hopes that at least one of the malicious messages would get through to the device.

# CONCLUSION

Blah blah blah de blah.

# ACKNOWLEDGMENTS

We are grateful Zimperium zLABS shared their knowledge with the general public on not only the details of the Stagefright vulnerability itself, but also for providing tools to help identify systems with the vulnerability and scripts to test the exploit.

We would also like to thank Kyle Digeorgio and his project partner for sharing their successes and failures in their research also on the Stagefight vulnerability.

# REFERENCES

1. Zimperium Mobile Security Blog. “Experts Found A Unicorn In The Heart Of Android” N.p., <https://blog.zimperium.com/experts-found-a-unicorn-in-the-heart-of-android/> Web. 05 Nov. 2015.
2. YouTube: Blackhat “Stagefright: Scary Code in the Heart of Android” N.p., <https://www.youtube.com/watch?v=71YP65UANP0> Web. Nov. 2015
3. GitHub: CyanogenMod/android\_frameworks\_av. “Fix integer overflow when handling MPEG4 tx3g atom” N.p., <https://github.com/CyanogenMod/android\_frameworks\_av/commit/5fd0cb515aba32d4dd961c16449bd5c8f4c37a6c> Web. 05 Nov. 2015
4. GitHub: CyanogenMod/android\_frameworks\_av. “Prevent reading past the end of the buffer in 3GPP” N.p., <https://github.com/CyanogenMod/android\_frameworks\_av/commit/57db9b42418b434751f609ac7e5539367e9f01a6> Web. 05 Nov. 2015
5. Zimperium Mobile Security Blog. “Stagefright: Vulnerability Details, Stagefright Detector tool released” N.p., https://blog.zimperium.com/stagefright-vulnerability-details-stagefright-detector-tool-released/> Web. 05 Nov. 2015
6. Exploit Database. “Android Stagefright – Remote Code Execution” N.p., <https://www.exploit-db.com/exploits/38124/> Web. 05 Nov. 2015.
7. Trend Micro: TrendLabs Security Intelligence Blog. “MMS Not the Only Attack Vector for Stagefright” N.p., < http://blog.trendmicro.com/trendlabs-security-intelligence/mms-not-the-only-attack-vector-for-stagefright/> Web. 05 Nov. 2015.
8. Hacker 9: The Unruly Security Channel. “Hacking any android smartphone using ‘Stagefright’ vulnerability” N.p., <http://www.hacker9.com/hack-android-smartphone-vulnerability.html> Web. 05 Nov. 2015.
9. The Hacker News: Security in a serious way. “How to Hack Millions of Android Phones Using Stagefright Bug, Without Sending MMS” N.p., <http://thehackernews.com/2015/07/how-to-hack-android-phone.html> Web. 05 Nov. 2015.
10. Exploit Database. “Android libstagefright – Integer Overflow Remote Code Execusion” N.p., <https://www.exploit-db.com/exploits/38226/> Web. 05 Nov. 2015.
11. How-To Geek. “How to Install Android in VirtualBox” N.p., <http://www.howtogeek.com/164570/how-to-install-android-in-virtualbox/> Web. 05 Nov. 2015.
12. Developer.android.com. “Android Studio Overview” N.p., <http://developer.android.com/tools/studio/index.html> Web. 05 Nov. 2015.
13. Developer.android.com. “Using the Emulator” N.p., <http://developer.android.com/tools/devices/emulator.html> Web. 05 Nov. 2015.
14. Genymotion. N.p., <https://www.genymotion.com/#!/> Web. 05 Nov. 2015.
15. AndroidCentral. “The ‘Stagefright’ exploit: What you need to know” N.p., < http://www.androidcentral.com/stagefright> Web. 05 Nov. 2015.

Columns on Last Page Should Be Made As Close As Possible to Equal Length