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 2.3.4-5.1.

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; MP4; MP3; Stagefright 2.0; Address Space Layout Randomization (ASLR)

1. 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 device user accesses the multimedia file.

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.

2. A DEEPER LOOK INTO THE STAGEFRIGHT VULNERABILITY

2.1 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.

2.2 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.

2.3 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 2^{32} an integer overflow occurs [3].

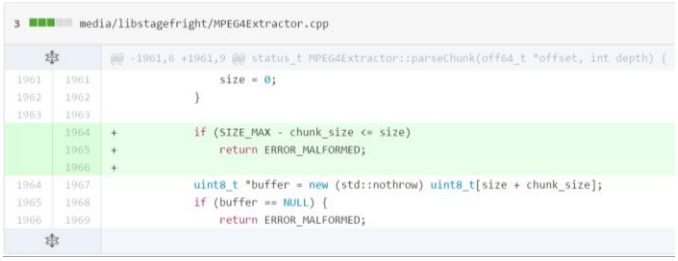


Figure 1. Integer Overflow[3].

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].

2.4 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, being placed on the device through proximity using NFC, or potentially placed on the device through network injection.

2.5 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.

2.6 Google's Response

The severity of this vulnerability was not lost on Zimperium zLABS. It drove them to report the vulnerability to Google, as well as to submit their own patches along with their report to help speed the turnaround process. Thankfully, Google recognized the gravity of the situation and was able to have a patch to their internal code within 48 hours. However, the process does not stop there. Once Google issues an official patch, typical updates to firmware by Google have still taken a fairly long time to reach actual users even from when a patch is issued. Third party provides add another potential wrinkle in getting out a quickly to users.

2.7 Stagefright 2.0

After the release of the information regarding the first Stagefright vulnerability, Zimperium zLABS continued to research the Stagefright library for any other potential weaknesses. They were focusing on other possible remote attacks and ran across another security issue, Stagefright 2.0.

2.7.1 New Vulnerability Details and Risks

This new vulnerability actually consists of two separate vulnerabilities and can be exploited through MP4 video file as before, as well as an MP3 audio file. The first part of the vulnerability is not contained in the Stagefright library itself, but a utility library. As such, it has been around since version 1.0 and may effect almost every Android device making it potentially even more dangerous than the original Stagefright vulnerability. While it is harder to exploit the vulnerability in the newer version of the Android OS (5.0 and later), it is possible through another vulnerability discovered in the Stagefright library.

2.7.2 Severity

The vulnerability within the utility library again deals with the processing of metadata contained in the files. This implies the user would not have to listen to the entire song or view the entire video for the exploit to be unleased. The malicious file would only need to be processed for it to start its dirty work. This also leaves the opportunity for the attacker to get in and out without detection as with the original Stagefright vulnerability.

2.7.3 Attack Vectors

With Google having released a patch for the MMS vulnerability, the attacker would this time have to use the web browser to get the exploit payload onto the system to be processed. While this would require the user to somehow download or allow the exploit payload to be place on the system, there are still ways not requiring user interaction for the exploit payload to affect the system. The exploit payload can be injected onto the device through network traffic if the attacker resides on the same network as the target device. In addition, the exploit payload could enter the system through third party applications that use the vulnerable utility library.

2.7.4 Google's Response to Version 2.0

Google was quick to respond to the information they were provided (by Zimperium zLABS) and worked to remedy the situation. They released patches to the Stagefright 2.0 vulnerabilities in their officially security fixes in October 2015. While this is potentially good news, the bad news is most devices, including Google's own Nexus line, did not receive the updates in the next monthly release. Since many devices have their firmware updates controlled by service providers, it may take a very long time before this fix actually makes it to all the devices affected by this new vulnerability.

3. REPRODUCTION TECHNIQUES

3.1 Environment

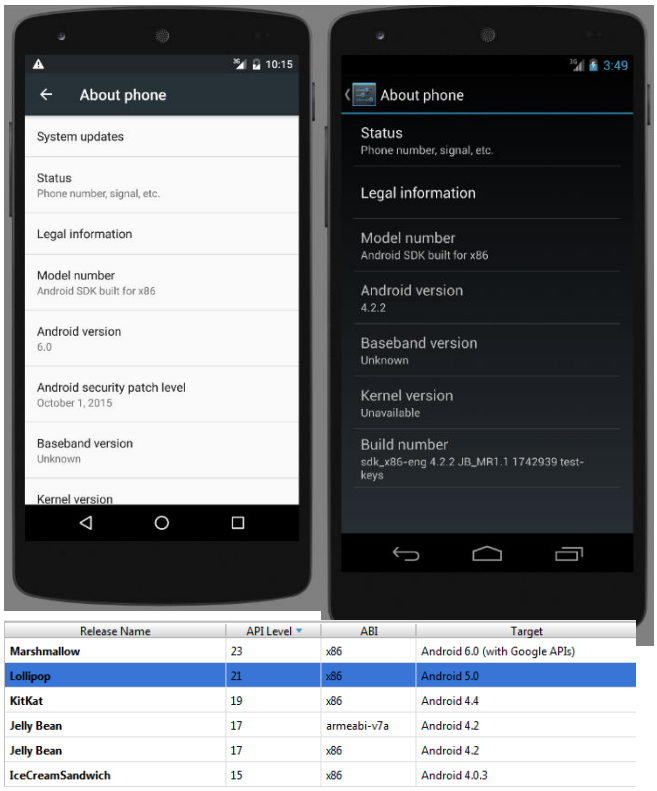
3.1.1 Android x86 OS

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.

3.1.2 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.



Release Name	API Level	ABI	Target
Marshmallow	23	x86	Android 6.0 (with Google APIs)
Lollipop	21	x86	Android 5.0
KitKat	19	x86	Android 4.4
Jelly Bean	17	armeabi-v7a	Android 4.2
Jelly Bean	17	x86	Android 4.2
IceCreamSandwich	15	x86	Android 4.0.3

Figure 3. Android Studio example.

Before moving from Android Studio to yet another tool, we also looked at the possibility of using the Android 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.

3.1.3 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).

3.2 Schedule

3.2.1 Original Plan

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

3.2.2 Changes After the First Setback

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.

3.2.3 Changes After the Second Setback

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.

3.3 Software Approach

3.3.1 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.

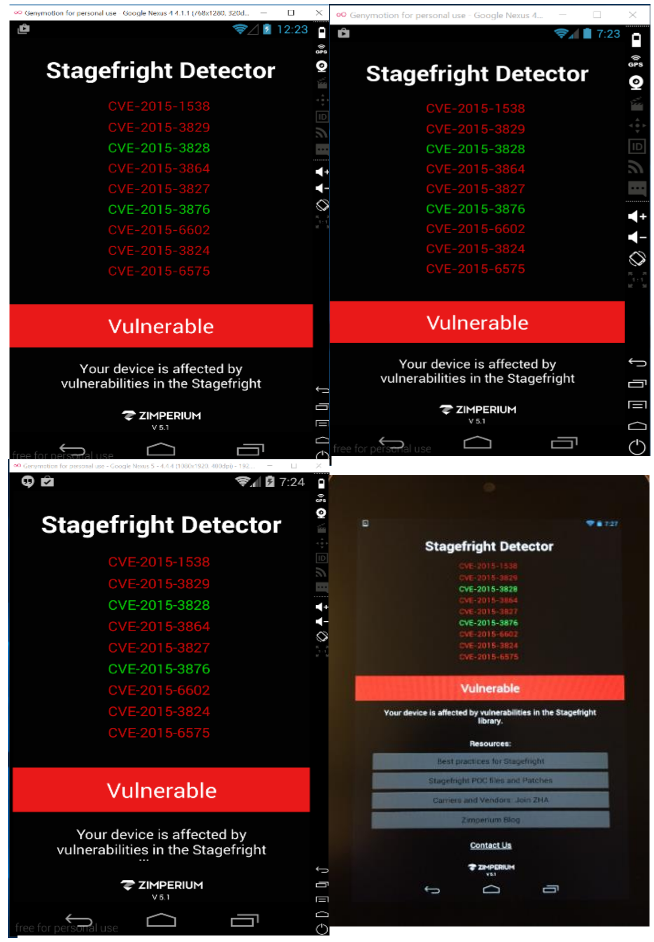


Figure 4. Detector tool examples[5].

3.3.2 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 not to use the script on the test Android devices, but to 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.

```
root@kali:~/Desktop/Exploit-Android-Stagefright-master# python mp4.py
usage: mp4.py [-h] [-c CBHOST] [-p CBPORT] [-s SPRAY_ADDR] [-r ROP_PIVOT]
              [-o OUTPUT_FILE]

optional arguments:
  -h, --help            show this help message and exit
  -c CBHOST, -connectback-host CBHOST
  -p CBPORT, -connectback-port CBPORT
  -s SPRAY_ADDR, -spray-address SPRAY_ADDR
  -r ROP_PIVOT, -rop-pivot ROP_PIVOT
  -o OUTPUT_FILE, -output-file OUTPUT_FILE
```

Figure 5. Exploit payload generation.

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

4. Results

None of the attempted methods were successful in completing the entire exploit as described including opening a shell on the systems; however we were able to get partial result with a few of the different devices.

4.1 VM (Android version 4.0 and later)

On the virtual systems, we were not able to see a shell with access to the system, but we did received errors stating the system "Can't play this video."

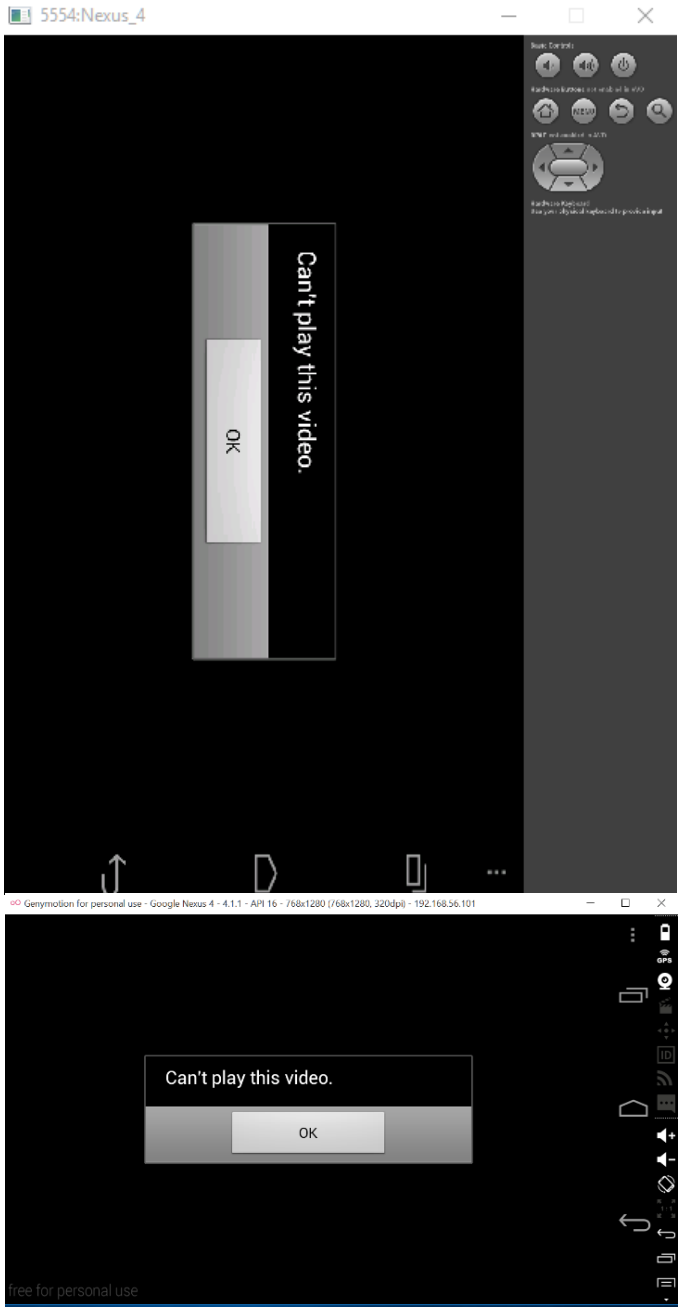


Figure 6. Nexus 4 error message examples.

4.2 Nexus 7 (Android version 4.1)

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.

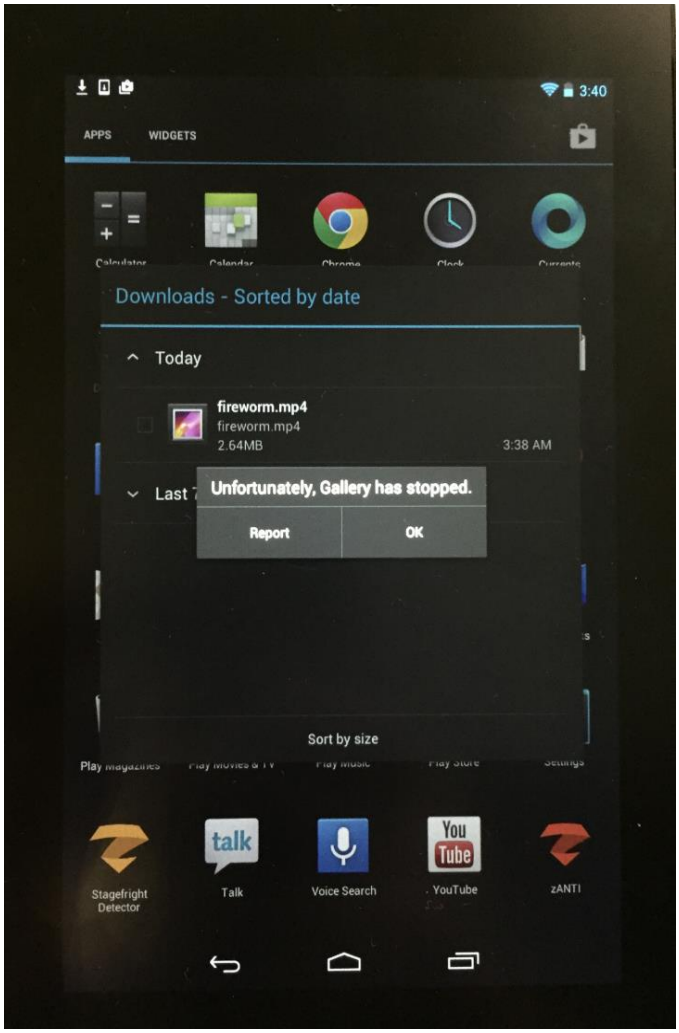


Figure 7. Nexus 7 device error message example.

4.3 Nexus S (Android version 2.3.4)

We again tried the exploit this time on a much older version of the Android OS with the Nexus S running 2.3.4. We were able crash the VM completely, but still were not able to generate a shell. We will discuss reasons for this more in the following section (4.4).

4.4 Reasons Behind Partial Success

4.4.1 Platform Specifics

We believe the reason the exploit was not successful in completely replicating the attack 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.

4.4.2 Address Space Layout Randomization (ASLR)

In addition, address space layout randomization (ASLR) was introduced in Android 4 and up. It is in use on the Android devices since the service is spawned 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.

4.4.3 Older Versions

Because of the potential for ASLR to thwart our attempts to reproduce the exploit we had testing on version 2.3.4 with the Nexus S. Literature on the topic suggests there are only 256 potential locations in which the attacker could find the malicious code. If the attacker is unable to locate the malicious code, the system will crash, which is what happened in our case. This tells us we were to getting the exploit to be successful, but device specific tweaks were probably needed to get the exploit payload to succeed.

5. CONCLUSION

5.1 Lessons Learned

One of the biggest lessons we learned in this research is how much work goes into not only discovering a vulnerability, but also developing exploits for those vulnerabilities. With all the different types of devices on the market and different version of software such as Android OS, what might work to exploit one particular device may not work on another device. This may be due to upgrades since the older version, difference in the hardware itself, or even potential patches the devices may or may not have received for other vulnerabilities.

5.2 Future Development

If it were possible to continue research and development on the Stagefright vulnerability exploit it would be first and foremost be great to get the original Stagefright exploit to work. With the work were able to accomplish in the time allotted, the results we received seemed to indicate we were on the cusp of having a successful exploit.

Once that is accomplished, it would be a great exercise to patch the devices with the Google provided fix and see if we are still able to reproduce the exploit. There may be cases where some of the older versions of the devices may still be vulnerable to some of the attack vectors or may produce other errors from our original exploit payload.

Third, our original idea of creating a worm to self-propagate the exploit payload onto the original vulnerable devices would show another level of how dangerous this vulnerability could have been. If the vulnerability had been discovered by people with malicious intent instead of Zimperium zLABS (which was very cooperative in doing all it could to help reduce the impact of the vulnerability they found) the potential harm not only to the individual devices, but also to Google could have been of large magnitude.

Finally, it would be interesting to work with the new Stagefright 2.0 vulnerability. Analyzing the similarities and

difference between the exploit payloads and potential attack vectors could help in research to protect against other similar issues that potentially have not been discovered yet.

6. 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 own research into/on the Stagefight vulnerability.

7. 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 Oct. 2015.

[2] YouTube: Blackhat “Stagefright: Scary Code in the Heart of Android” N.p., <https://www.youtube.com/watch?v=71YP65UANP0> Web. Oct. 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 Oct. 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 Oct. 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 Oct. 2015

[6] Exploit Database. “Android Stagefright – Remote Code Execution” N.p., <https://www.exploit-db.com/exploits/38124/> Web. 05 Oct. 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. 15 Oct. 2015.

[8] Naked Security by Sophos. “Google issues Android patches for Stagefright 2 (for some users)” N.p., < https://nakedsecurity.sophos.com/2015/10/06/google-issues-android-patches-for-stagefright-2-for-some-users/> Web. 15 Oct. 2015

[9] 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 Oct. 2015.

[10] 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 Oct. 2015.

[11] Exploit Database. “Android libstagefright – Integer Overflow Remote Code Execusion” N.p., <https://www.exploit-db.com/exploits/38226/> Web. 05 Oct. 2015.

[12] How-To Geek. “How to Install Android in VirtualBox” N.p., <http://www.howtogeek.com/164570/how-to-install-android-in-virtualbox/> Web. 05 Oct. 2015.

[13] Developer.android.com. “Android Studio Overview” N.p., <http://developer.android.com/tools/studio/index.html > Web. 15 Oct. 2015.

[14] Developer.android.com. “Using the Emulator” N.p., <http://developer.android.com/tools/devices/emulator.html> Web. 15 Oct. 2015.

[15] YouTube: ujjwal dwivedi “Hacking Android Using Stagefright Exploit” N.p., <https://blog.zimperium.com/zimperium-zlabs-is-raising-the-volume-new-vulnerability-processing-mp3mp4-media/> Web. 15 Oct. 2015

[16] Genymotion. N.p., <https://www.genymotion.com/#!/> Web. 15 Oct. 2015.

[17] Stackoverflow: “How to install Google Play Services in Genymotion VM (with no drag and drop support” N.p., http://stackoverflow.com/questions/20121883/how-to-install-google-play-services-in-a-genymotion-vm-with-no-drag-and-drop-su> Web. 15 Oct. 2015.

[18] AndroidCentral. “The ‘Stagefright’ exploit: What you need to know” N.p., < http://www.androidcentral.com/stagefright> Web. 05 Oct. 2015.

[19] CheatSheet. “What You Need to Know About Android’s Stagefright Vulnerability” N.p., <http://www.cheatsheet.com/gear-style/what-you-need-to-know-about-androids-stagefright-vulnerability.html/?a=viewall> Web. 15 Oct. 2015

[20] Zimperium Mobile Security Blog. “Zimperium zLabs is Raiding the Volume: New Vulnerability Processing MP3/MP4 Media” N.p., https://blog.zimperium.com/zimperium-zlabs-is-raising-the-volume-new-vulnerability-processing-mp3mp4-media/> Web. 15 Oct. 2015