StageFright Attack

By Sarah Larbi, Victoria Thomson, Konstantino Sparakis, and Megan Horan

Abstract

Stagefright was a hack initially discovered and reported to Google by a security researcher, Joshua Drake, in April 2015. News of the attack surfaced in the mainstream media later that summer, describing it as “the worst android hack ever,”[3] affecting nearly 950 million devices. Headlines ominously depicted the attack, implying that no Android was safe. Stagefright was a library within the Android media server library. It existed in several versions of Android’s operating system and allowed an attacker to execute remote code on a user’s device, potentially without detection. In our report, we detail what the Stagefright vulnerability was, as well as demonstrate how Joshua Drake was able to exploit it.

Superficial Issues

Joshua Drake, security researcher at Zimperium, discovered a set of vulnerabilities in all versions of Android’s operating system greater than 2.2 (“Froyo”). He reported the bug to Google as well as incorporated a related bug fix into its internal source code repositories two days after the report. [1] The bug was announced publicly in July 2015 and the full disclosure took place on August 5th at the Black Hat USA conference. When the bug was announced publicly in July, the hack was reported in the mainstream media as potentially affecting up to 950 million devices[2]. Reportedly the scariest part of the hack was that, contrasting to a spear-phishing attack, all vulnerable devices could potentially be attacked via a simple text message that required zero interaction on user side.

Technical Overview

Underlying technical issues surrounded the C++ media processing library described above, StageFright. Libstagefright is located in the media server.

Drake started his search by running several tests on the media server to see where there were errors/crashes. His tests resulted in discovering 5 different memory corruptions. These became CVE-2015-1538 and CVE-2015-1539, which centered around problems pertaining to buffer and overflow issues. These types of issues have been proven exploitable in the past and provide the main source of exploitation in this attack, and will be elaborated on in the next section.

Drake then started digging deeper into the media server framework and libstagefright library. He began by finding all the ways he could send media files to himself. Through this methodology he discovered eleven different attack vectors. These vectors included Mobile Network MMS, Client Side, and more [7].

The attack vector he chose to exploit in this specific attack was the MMS vector. This attack proved to be particularly powerful because if the media files received via text message contained a header, Android would automatically download the attachment. This means that upon receipt of an MMS containing an audio/video file, libstagefright would automatically begin processing the malicious code before the user even opens the message. Further, the media library executes continuously from the moment you receive the file, to when you view the notification, to when you tilt your screen on the text message page.

The MMS Attack

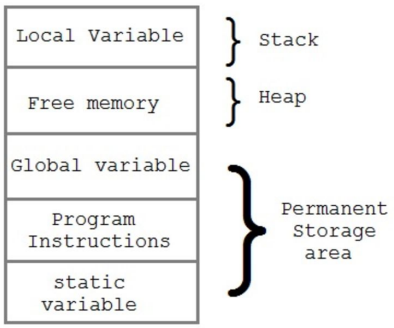
Now that the malicious MMS received, the attack began to attack the media server and eventually escalate privileges.

libstagefright had overflow issues. This is what allowed an attacker to execute malicious code in media files. There were multiple exploitable overflows across various attack vectors, but this attack focuses on the MPEG4 files. In the code for processing MPEG files, it contained an integer overflow vulnerable function “parseChunk(offset, depth + 1)” [7]. Overflow issues are elaborated on in the next section.

```
671     switch(chunk_type) {
672         case FOURCC('m', 'o', 'o', 'v'):
673         case FOURCC('t', 'r', 'a', 'k'):
...
724             while (*offset < stop_offset) {
725                 status_t err = parseChunk(offset, depth + 1);
```

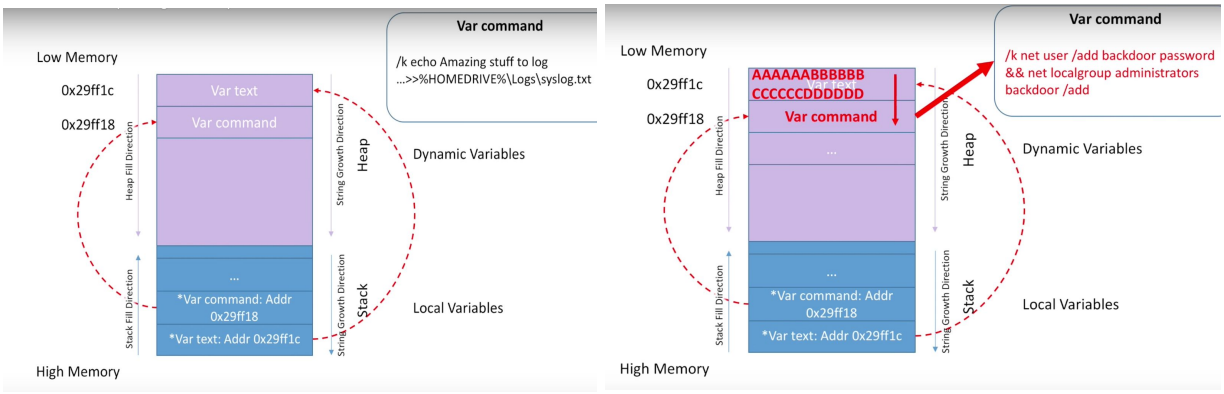
How Does the Buffer Overflow Work? [12.]

Buffer overflows are an entire family of attacks. certain parts Androids, such as the media library, are written in C++ . The improper use of memory pointers is what allows for these stack overflow vulnerabilities which was the basis for the Stagefright attack. The integer overflows, pointed out in detail earlier with references to CVE’s, are a sort of heap overflow attack.



Every computer process has memory allocated to it in a specific fashion, as pointed out by [12.]. This memory space will always range a sequence of memory address. All variables, computer instructions and things related to this program will be located within this memory space. For the sake of this attack, we will concentrate on the heap and stack and ignore static variable and program instructions.

In the diagram below on the left, we see that in the Stack we have a pointer, that points to the memory location of our variable text, which is located in the heap. Let's say text = “AAAAA...”. The memory location for text is at *Var text=“0x29ff1c”.Var command, points to “0x29ff18” in this example.



[11.]

This attack works when the actual size of text, exceeds the allocated size of text. Then it keeps writing over whatever is next. So what the attacker does is send in a variable text = “AAAA...” + Malicious(){does something bad;}. There is still one issue here: the malicious function needs to override perfectly where the function command() is in the memory.. But to an attacker's

advantage there exists a CPU instruction called a “nop”. The CPU will skip over this until it reaches it’s next valid instruction. So an attacker adds this in between the text and their malicious code, in something called a Nop Slide: text = “AAAA...” + “nop/nop/nop/nop….” + “Malicious(){does something bad;}”. So as long as one of the nops is positioned properly, when the process thinks it is calling command(), if it hits a nop it will keep executing nop which is nothing until it reaches the instructions of Malicious(), executing the malicious code. Enter into Drake’s Stagefright attack.

Given that most modern heaps have meta data and other architectures would complicate such an attack. This explanation is generally oversimplified, yet buffer overflow is the premise of the Stagefright attack. The media file containing the malicious code, once it overflows the buffer it will execute this flavor of attack. Once the malicious code is running, the attacker can escalate his codes’ privileges over the device device.This gives an attacker the ability to conduct a REMOTE, SILENT, and PRIVILEGED code execution due to the nature of the media server.[7]

Preventing the Attack: Provider Level

Prevention attempt: (ASLR)

In order to prevent the buffer overflow, at the operating system level, Google has failed to provide a strong ASLR solution. ASLR randomizes the memory addresses of all items in the heap and stack, which an attacker, in order to execute their attack, needs to know the exact addresses of objects in memory.

Initially, Android version Gingerbread did not have ASLR. Then ASLR was added in the next version, Ice Cream Sandwich, which was also exploitable. Google Claims that at version 4.1 “Kitkat” they have implemented a fully working ASLR, but as sources point out [5], this ASLR is weak and does not impede attackers sufficiently. Android ASLR only provides 2^8 entropy or randomization, which means an attacker has a 1/256 probability that they will guess the memory location of their malicious payload properly.

Further research has also shown that there are ways to bypass ASLR in Android due to leaking of information regarding memory allocation of processes. This allows for side channel attacks allowing you to bypass ASLR [10]. ASLR can still be somewhat considered a good protection system for other computers, but more and more research are pointing to the fact that this is not enough and more of a temporary solution [6]. If Google could strengthen their ASLR this would be key to preventing these attacks that are in the buffer overflow family.

Preventing the Attack: User Level

- Avoid downloading media files (mp3, mp4, mpeg...) from sketchy websites or torrents. [9.]
- Avoid installing android apps from third party app stores, as the vulnerabilities can be exploited from a malicious app. [9.]
- Disable auto-retrieve mms functions / block texts from unknown numbers
- Download Stagefright Detector apps found in google play.
- Update to Android Nougat (Version 7.0/7.1)
    - Although patches were released to fix the specifically pointed out vulnerabilities initially found in Stagefright, researchers have been continuously finding other related attacks within the media libraries that can have the same effect.
    - Google has announced that in Android Nougat [8.], they have completely redesigned the media library to avoid these types of attacks altogether. Updating to version 7.0 or high of

android is the only real solution to avoid the entire family of attacks that may still be lurking in the old media libraries such as stagefright.

Discussion of incentives/motivation for the attack.

At Black Hat USA 2015, he describes his primary motivation for the project to be his interest in “improving overall mobile security”[4].

The stagefright library was compelling for Drake. Prior to the discovery of the Stagefright bug set, there had been multiple public mentions of instability/crashes**.** Many users were reporting battery death, phone reboots, and a slew of other errors. Intel fuzz tested the media library on these Android devices and discovered nearly 12 terabytes of crashes within the server[3]**.** All of this pointed back to ‘corrupt media files’ which led to Joshua Drake looking further into the media server library to find what was going wrong.

“This work was done without any outside collaboration.”

Resources

1. https://en.wikipedia.org/wiki/Stagefright_(bug)
2. https://www.forbes.com/sites/thomasbrewster/2015/07/27/android-text-attacks/#674525573a50
3. https://www.lifewire.com/protect-yourself-from-the-worst-android-hack-ever-2487129
4.https://video.search.yahoo.com/yhs/search;_ylt=A0LEVr0Q7sZYy20AQ.snnIlQ?p=joshua+drake+black+hat&fr=yhs-mozilla-002&fr2=piv-web&hspart=mozilla&hsimp=yhs-002#id=&vid=&action=close
5. https://arstechnica.com/security/2015/09/googles-own-researchers-challenge-key-android-security-talking-point/
6. http://www.tomshardware.com/news/aslr-apocalypse-anc-attack-cpus,33665.html
7. https://www.blackhat.com/docs/us-15/materials/us-15-Drake-Stagefright-Scary-Code-In-The-Heart-Of-Android.pdf
8. http://www.theverge.com/2016/9/6/12816386/android-nougat-stagefright-security-update-mediaserver
9.http://www.techradar.com/news/phone-and-communications/mobile-phones/how-to-keep-your-android-device-safe-from-stagefright-2-0-1305897
10.https://www.blackhat.com/docs/us-15/materials/us-15-Gong-Fuzzing-Android-System-Services-By-Binder-Call-To-Escalate-Privilege-wp.pdf
11. https://www.youtube.com/watch?v=rtkRYxbt-r8
12. https://latestcomputing.blogspot.com/2014/08/memory-stack-vs-heap.html