

WhatsApp Buffer Overflow Vulnerability

Case Study

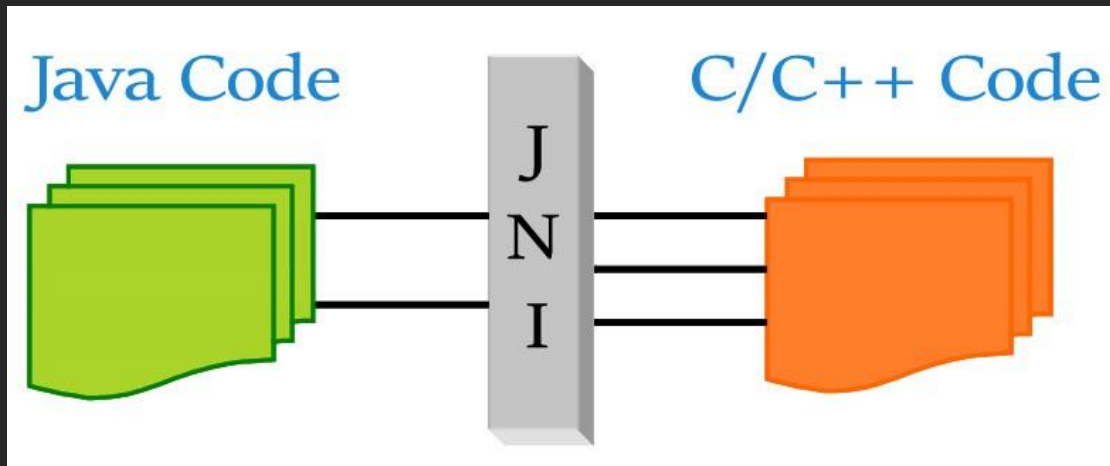
By Mohammed Muteb

Introduction

في سنة 2019 أعلنت Facebook إصلاح ثغرة Buffer overflow تحمل الرقم CVE-2019-3568 مصابة في Whatsapp's VoIP implementation يستطيع Attackers إستغلالها عبر تحميل malicious RTCP packets داخل VoIP call [1]. مما يعني بمقدور attackers إستغلالها بمجرد الإتصال بالضحية بدون الحاجة منه للرد على الإتصال (zero-click exploit), ستعتمد هذه المقالة في التحليل statically و dynamically على WhatsApp for Android حيث أن الإصدار المصاب [2] هو 2.19.133 أما ما هو أعلى من هذا الإصدار قد تم إصلاحه.


NDK

في ما مضى كان android app development يعتمد في تطويره على لغة Java مما يعني سوف تفتقر لمميزات لغات مثل C++/C كان Java developers يحتاجونها ولهذا ظهرت مجموعة من الأدوات تسمى اليوم بمصطلح مختصر ألا وهي NDK حيث ستسمح لتطبيقات Android على التطوير بلغات أخرى غير Java/Kotlin وهي C++/C وهذا سوف يساعد في performance وأيضا تساعد في تصعيب عمليات الهندسة العكسية ف لغات مثل C++/C تنتج binary-translated objects وليس bytecode-translated objects يسهل إعادتها لحالتها السابقة, والطريقة للوصول إلى control flow المكتوب بلغة C++/C هي عن طريق Java Native Interface ذات الاختصار (JNI) فهي وسيلة وصول إلى المكاتب المكتوبة بلغة C++/C [3]



Setup


كما هو وارد ومعروف أن الإصدارات القديمة من Whatsapp فور تشغيلها سيظهر إما خطأ، الأول بأن التاريخ غير دقيق والآخر بأن الإصدار جدا قديم وهو كذلك فعلا !



هذا الإصدار من واتساب أصبح قديماً، ولم يعد يمكن استخدامه بدءاً من تاريخ ٢٠١٩/٠٨/١٩. انقر على "تنزيل" للحصول على أحدث إصدار من متجر Play.

التاريخ المحدد في هاتفك هو ٢٠١٩/٠٨/١٨. إذا كان هذا التاريخ غير صحيح، يرجى تصحيح إعدادات التاريخ في هاتفك. ثم أعد تشغيل واتساب.

تنزيل



تاريخ الهاتف غير دقيق. اضبط الساعة ثم أعد المحاولة.

التاريخ والوقت في هاتفك:
٢٠٢٢/١٠/٢٠ ١:٥٤ ص
(التوقيت العربي الرسمي)

ضبط التاريخ

الخطأ الأول سهل إصلاحه وهو عن طريق تغيير التاريخ من الهاتف نفسه إلى التاريخ الذي كان آنذاك الإصدار يعمل بدون مشاكل مما يعني سنة 2019 ومابعد شهر مايو، اما خطأ الإصدار قديم عندما نقوم بعمل هندسة عكسية لتطبيق WhatsApp الإصدار المصاب ونبحث عن 2.19.133 نكتشف وبالتحديد في Java class d.f.C1576cu أن WhatsApp يعتمد في البداية على SharedPreferences للتعرف على إصدار WhatsApp وإقرار ما إذا كان هذا الإصدار صالحاً للإستخدام أو أصبح قديم جداً

```
d.f.va.Db a3 = d.f.va.Db.a(c2674m.f19958c.getString("version", "0.0.0"));
if (a3 != null) {
    Log.d("app-init/async/version/previous " + a3);
    d.f.va.Db a4 = d.f.va.Db.a("2.19.133");
    if (a4 != null) {
        Log.d("app-init/async/version/current " + a4);
        c3256zI.m = a4.a(a3);
        if (c3256zI.m != 0) {
            c3256zI.d();
            a(application, a3, aVar, c2674m, bg, d2, tbVar, f2, ubVar, c1344fa);
            c2674m.g().putString("version", "2.19.133").putLong("client_version_upgrade_timestamp", System.currentTimeMillis()).putBoolean("client_version_upgraded", true).apply();
        }
    }
}
```

لكن هناك JNI call يتحقق من الإصدار أيضا ألا وهو `getJNICodeVersion` موجود في `com.whatsapp.util.WhatsAppLibLoader` class المسؤول عن تحميل native libraries في WhatsApp's JVM layer

```
public final boolean a() {
    byte[] bArr = new byte[3];
    try {
        testLibraryUsable(bArr);
        if (!Arrays.equals(new byte[]{31, 41, 59}, bArr)) {
            Log.w("whatsapplibloader/usable test array does not match");
            return false;
        }
        try {
            String jNICodeVersion = getJNICodeVersion();
            Log.i("whatsapplibloader/usable jniVersion: " + jNICodeVersion);
            if (!"2.19.133".equals(jNICodeVersion)) {
                Log.w("whatsapplibloader/usable version does not match. JAVA version: 2.19.133, JNI version: " + jNICodeVersion);
                return false;
            }
            try {
                Voip.getCallInfo();
                Log.i("whatsapplibloader/usable isLibraryUsable: True");
                return true;
            } catch (UnsatisfiedLinkError e2) {
                Log.w("whatsapplibloader/usable error while testing library usability getCallInfo", e2);
                return false;
            }
        } catch (UnsatisfiedLinkError e3) {
            Log.w("whatsapplibloader/usable error while testing library usability getJNICodeVersion", e3);
            return false;
        }
    } catch (UnsatisfiedLinkError e4) {
        Log.w("whatsapplibloader/usable error while testing library usability testLibraryUsable", e4);
        return false;
    }
}
```

يُتبقّى الآن معرفة المكتبة التي تحتوي على `getJNICodeVersion` JNI call لكن مهمة `WhatsAppLibLoader` ليست مقصوده على تحميل مكتبة واحدة بل جميع المكتبات كم نرى في دالة `b` من `util.WhatsAppLibLoader` class المسؤوله عن عملية library loading

```
public static void b(Context context, String[] strArr) {
    String str = Build.CPU_ABI;
    String str2 = "x86";
    if (str.startsWith("armeabi-v7")) {
        str2 = "armeabi-v7a";
    } else if (str.startsWith("arm64-v8a")) {
        str2 = "arm64-v8a";
    } else if (str.startsWith("x86_64")) {
        str2 = "x86_64";
    } else if (!str.startsWith(str2)) {
        throw new UnsatisfiedLinkError(a.b("can not find lib folder for ABI ", str));
    }
    a.d("whatsapplibloader/arch resolved to ", str2);
    try {
        ZipFile zipFile = new ZipFile(new ContextWrapper(context).getPackageCodePath());
        Map<String, File> a2 = a(context, zipFile, str2);
        a(a2, strArr);
        for (File file : b(a2, strArr)) {
            String absolutePath = file.getAbsolutePath();
            System.load(absolutePath);
            Log.i("whatsapplibloader/try-install loaded: " + absolutePath);
        }
        zipFile.close();
    } catch (IOException e2) {
        Log.e("whatsapplibloader/try-install ioerror", e2);
        throw new UnsatisfiedLinkError("IOException when install native library");
    }
}
```

بعد البحث نكتشف ان المكتبة المسؤوله عن تنفيذ JNI call `getJNICodeVersion` هي `libwhatsapp.so` مما يعني سوف نقوم بعمل هندسة عكسية للملف التنفيذي `libwhatsapp` المتواجد في `lib folder` وسنستخدم IDA (ليس حصرا) للهندسة العكسية, حيث سنرى مسمى JNI call هكذا `Java_com_whatsapp_util_WhatsAppLibLoader_getJNICodeVersion`

```

1 int __fastcall Java_com_whatsapp_util_WhatsAppLibLoader_getJNICodeVersion(int a1)
2 {
3 return (*(int (__fastcall **)(int, const char **))(_DWORD *)a1 + 668))(a1, "2.19.133");
4 }

```

يلاحظ البعض غرابة `pseudocode`. يحدث ذلك بسبب عدم تعريف JNI structure مما يجعل برامج الهندسة العكسية مثل IDA لا تعرف كيف تترجم `struct-purpose statements` بشكل مفهوم للمحلل بدون إضافتها يدويا بسبب أن JNI يعتمد وبشكل كبير على `structures` الخاصة به وعند إضافة `jni.h` header file في نافذة "Parse C Header File" سوف يتغير مظهر `pseudocode` قليلا ليتضح لنا الكود أكثر [4]

```

1 jstring __fastcall Java_com_whatsapp_util_WhatsAppLibLoader_getJNICodeVersion(JNIEnv *env, jclass clazz)
2 {
3 return (jstring)((int (__fastcall *) (JNIEnv *, const char **))(*env->NewStringUTF)(env, "2.19.133"));
4 }

```

ومن هنا نعرف ان `statement` الوحيد الموجود الذي كتبه developer في `getJNICodeVersion` JNI call كان مجرد

```
return env->NewStringUTF("2.19.133");
```

والآن بعد `static analysis` هذه المرحلة نتجه لمرحلة تخطي هذه `checks` عن طريق `hooking` بإستخدام `Frida` حيث أن `check` الأول يعتمد على `SharedPreferences` وبالتحديد دالة `putString` [5]

```

Java.perform(function() {
    var SP = Java.use("android.app.SharedPreferencesImpl$EditorImpl");
    SP.putString.overload('java.lang.String', 'java.lang.String').implementation =
function(name, value) {
    if(name == 'version'){
        var editor = this.putString(name, "2.22.22.81");
        console.log("[+] Whatsapp version is patched with the new one [+]");
    }else{
        var editor = this.putString(name, value);
    }
    return editor;
}
});

```

اما check الثاني يعتمد على JNI calls وكما ذكرت سابقا ان JNI structs تمثل البنية التحتية لكل JNI Statements [6] أي ان في حال hooking من Frida سنستعين بـ wrapper- Java.vm.getEnv() [7]

```
Java.perform(function() {
    var SP = Java.use("android.app.SharedPreferencesImpl$EditorImpl");
    SP.putString.overload('java.lang.String', 'java.lang.String').implementation =
function(name, value) {
    if(name == 'version'){
        var editor = this.putString(name, "2.22.22.81");
        console.log("[+] Whatsapp version is patched with the new one [+]");
    }else{
        var editor = this.putString(name, value);
    }
    return editor;
}
});

Interceptor.attach(Module.findExportByName('libwhatsapp.so',
'Java_com_whatsapp_util_WhatsAppLibLoader_getJNICodeVersion'), {
    onLeave: function (retval) {
        console.log("[+] Whatsapp version is patched with the new one natively [+]");
        retval.replace(Java.vm.getEnv().newStringUtf("2.22.22.81"));
    }
});
```

مما يرشدنا ذلك إلى أهمية معرفة التعامل مع structures وتعريفها أثناء الهندسة العكسية لكن الكود السابق سوف يظهر خطأ شائع الحدوث ألا وهو Error: expected a pointer بسبب ان مكتبة libwhatsapp لم يتم تحميلها بعد لأن frida بدأ عمله قبل ان تكون المكتبات loaded في الذاكرة وهذه timing issue

```
C:\Users\HP\Desktop>frida -U -f com.whatsapp -l whatsapp.js --no-pause

Frida 15.0.13 - A world-class dynamic instrumentation toolkit

Commands:
  help          -> Displays the help system
  object?       -> Display information about 'object'
  exit/quit     -> Exit

More info at https://frida.re/docs/home/

Connected to SM N935F (id=192.168.26.57:5555)
Spawned `com.whatsapp`. Resuming main thread!
Error: expected a pointer
    at value (frida/runtime/core.js:316)
    at <eval> (/whatsapp.js:21)
[SM N935F::com.whatsapp ]-> [+] Whatsapp version is patched with the new one [+]
[SM N935F::com.whatsapp ]->
[SM N935F::com.whatsapp ]-> Module.findExportByName('libwhatsapp.so', 'Java_com_whatsapp_util_WhatsAppLibLoader_getJNICodeVersion')
"0x78f1475750"
[SM N935F::com.whatsapp ]-> exit

Thank you for using Frida!
```

لذلك سوف نستغل linker namespace [8] حيث كل عملية loading سوف تحتاج المكتبات dependencies ولهذا سوف نستخدم دالة android_get_exported_namespace [9] من مكتبة libdl.so أحد مكتبات bionic [10] حيث بعد إنتهاء عملها يستطيع Frida العمل

```
// bypass WhatsApp expiration date/version error
Java.perform(function() {
    var SP = Java.use("android.app.SharedPreferencesImpl$EditorImpl");
    SP.putString.overload('java.lang.String', 'java.lang.String').implementation =
function(name, value) {
    if(name == 'version'){
        var editor = this.putString(name, "2.22.22.81");
        console.log("[+] Whatsapp version is patched with the new one [+]");
    }else{
        var editor = this.putString(name, value);
    }
    return editor;
}
});

Interceptor.attach(Module.findExportByName('libdl.so',
'android_get_exported_namespace'), {
    onLeave: function () {
        Interceptor.attach(Module.findExportByName('libwhatsapp.so',
'Java_com_whatsapp_util_WhatsAppLibLoader_getJNICodeVersion'), {
            onLeave: function (retval) {
                console.log("[+] Whatsapp version is patched with the new one natively [+]");
                retval.replace(Java.vm.getEnv().newStringUtf("2.22.22.81"));
            }
        });
    }
});
```

وبهكذا سوف نتخطى جميع التحقيقات التي وضعتها whatsapp وقد تعيق dynamic analysis

```
C:\Users\HP\Desktop>frida -U -f com.whatsapp -l whatsapp.js --no-pause
Frida 15.0.13 - A world-class dynamic instrumentation toolkit
Commands:
  help           -> Displays the help system
  object?       -> Display information about 'object'
  exit/quit     -> Exit
  . . . . .
  More info at https://frida.re/docs/home/
  . . . . .
  . . . . . Connected to SM N935F (id=192.168.26.57:5555)
Spawned 'com.whatsapp'. Resuming main thread!
[SM N935F:com.whatsapp ]> [+] Whatsapp version is patched with the new one [+]
[+] Whatsapp version is patched with the new one natively [+]
```

أهلاً بك في واتساب

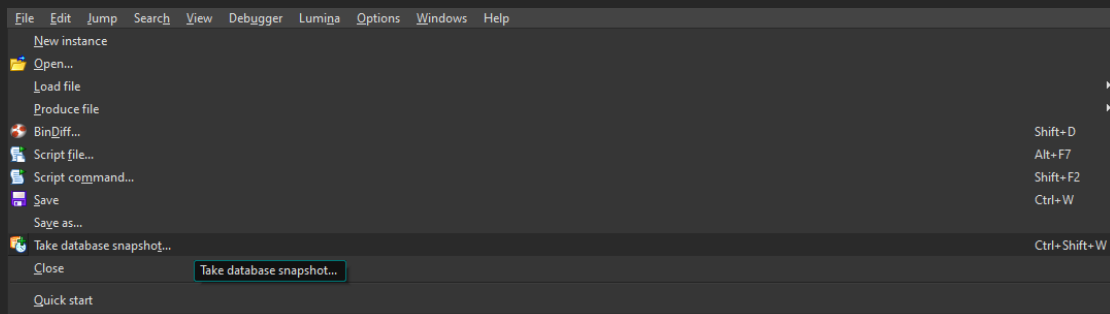


اقرأ سياسة الخصوصية (انقر على "الموافقة والمساعدة") لتبذل شروط الخدمة.

الموافقة والمساعدة

Root Cause

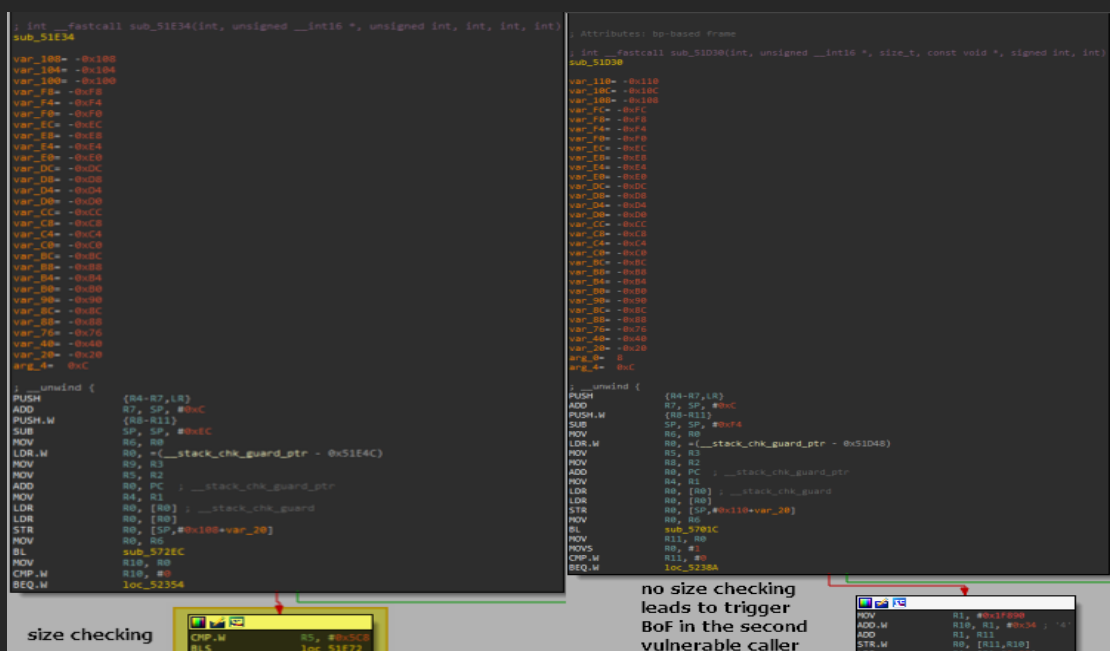
في هذه المرحلة سنعتمد على binary diffing [11] حيث أنها سوف تقارن لنا التغييرات البرمجية التي حصلت بين vulnerable native library و patched native library ولا وهي libwhatsapp وسنستخدم من أجل binary diffing أداة BinDiff على IDA [12], حيث سوف نرى التغييرات البرمجية التي حصلت في الدالتين sub_51D30 و sub_52F00 [13] قبل patching لذلك سوف نأخذ database snapshot للملف التنفيذي libwhatsapp.so بعد patching



ثم نستخدم BinDiff للقيام بـ binary diffing للدوال المذكورة سابقا, في sub_51D30 نرى أن patched native function هي sub_51E34 وكليةما يعتبرون RTCP handler function [14]

Similarity	Confid	Change	EA Primary	Name Primary	EA Secondary	Name Secondary
0.87	0.97	GI-JE--	00051D30	sub_51D30	00051E34	sub_00051E34

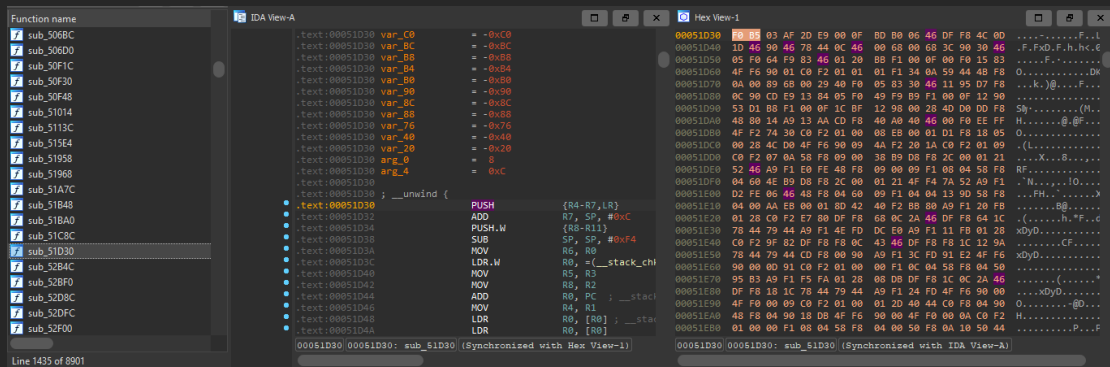
بعد patching أصبحت sub_51E34 تقوم بعمل size checking من طرفها لتجنب أي احتمالات حدوث buffer overflow كما يظهر في الصورة, على اليمين vulnerable native function وعلى اليسار patched native function



حيث يقوم دور size checking بالتحقق من حجم argument أن لا يتعدى 1480 بايت كم هو ظاهر (0x5c8), دالة RTCP handler تعتبر initializing function للـ RTCP module الذي نقوم بتحليله الآن [14] لأنه يتم إستدعاءها مباشرة قبل أن يرد الضحية على الإتصال الذي أتاه من Attacker أثناء exploitation مروراً إلى callee function ألا وهو sub_52F00 vulnerable function

Similarity	Confid	Change	EA Primary	Name Primary	EA Secondary	Name Secondary
0.91	0.99	GI-----	00052F00	sub_52F00	00052D0C	sub_00052D0C

ومعرفة execution path هنا أنت من تحليل dynamically للدالة المذكورة حيث في البداية يجب معرفة أين توجد initializing function بحساب offset اللازم للوصول إليها ويمكن ذلك manually عن طريق IDA/[15] Ghidra مثلًا أو programmatically عن طريق إستخراج prologue opcodes للدالة والبحث عنها بمساعدة Memory.scan's frida method [16] و returnAddress property [17]



ومن binary diffing نرى ان sub_52D0C هي patched native function وتحليلاً لدالة sub_52D0C نرى انها تقوم بعملية size checking [13] لحجم arguments قبل تمريره إلى memcpy بعكس sub_52F00 vulnerable function التي لا تقوم بذلك

```

if ( v9 )
{
    v12 = (void (__fastcall *)(int, int, size_t, int, int))a1[4729];
    v13 = (*(_DWORD *)a2 >> 4) & 1;
    if ( v12 )
    {
        v14 = a1[4727];
        v15 = sub_2000CC(*(_DWORD *)a2 + 4);
        v12(v14, a2, a3, v13, v15);
        sub_3ABC4(a2, a3, v22);
        v16 = 12;
        if ( !v13 )
        {
            v16 = 5;
            sub_4FA90(a1, v16, v22, 4);
        }
        else if ( a5 && (*(_DWORD *)a2 & 0xFE00) == 51200 )
        {
            memcpy(a1 + 32225, (const void *)a2, a3);
            a1[32289] = a3;
        }
    }
}

v14 = 4 * v13 + 4;
v15 = (v12 >> 4) & 1;
if ( v14 <= a3 )
{
    v20 = (void (__fastcall *)(int, int *, size_t, int, int))a1[4650];
    if ( v20 )
    {
        v21 = a1[4648];
        v22 = sub_1FDD08(a2[1]);
        v20(v21, a2, a3, v15, v22);
        sub_3A5B4(a2, a3, v25);
        v23 = 12;
        if ( !v15 )
        {
            v23 = 5;
            sub_4F2DC(a1, v23, v25, 4);
        }
        else if ( a3 <= 0x5C8 && a5 && (v12 & 0xFE00) == 51200 )
        {
            memcpy(a1 + 32137, a2, a3);
            a1[32507] = a3;
        }
    }
}

```

مما يسبب في نهاية المطاف ثغرة buffer overflow [18] في WhatsApp for Android 2.19.133

FAQ

- Is this bug reproducible ?
 - No, It's not reproducible because whatsapp already fixed the cause in their end (whatsapp server) [13], the blogpost is just a case study of this security bug from reverse engineering perspective.
- linker namespace trick didn't works for me!
 - OS version incompatibility, use android_dlopen_ext [19]
- Is it only for Android ?
 - No, the vulnerability infected many platforms [1] such as WA for iOS [20]

References

- [1] Facebook, "CVE-2019-3568," [Online]. Available: <https://www.facebook.com/security/advisories/cve-2019-3568>.
- [2] APKMirror, "APKMirror," 2019. [Online]. Available: <https://www.apkmirror.com/apk/whatsapp-inc/whatsapp/whatsapp-2-19-133-release/whatsapp-messenger-2-19-133-android-apk-download/>.
- [3] A. Developers, "Get started with the NDK | Android NDK | Android Developers," [Online]. Available: <https://developer.android.com/ndk/guides>.
- [4] OpenJDK, "JDK," [Online]. Available: <https://github.com/openjdk/jdk/blob/master/src/java.base/share/native/include/jni.h>.
- [5] A. Developers, "SharedPreferences.Editor | Android Developers," [Online]. Available: [https://developer.android.com/reference/android/content/SharedPreferences.Editor#putString\(java.lang.String,%20java.lang.String\)](https://developer.android.com/reference/android/content/SharedPreferences.Editor#putString(java.lang.String,%20java.lang.String)).
- [6] A. Developers, "JNI tips | Android Developers," [Online]. Available: <https://developer.android.com/training/articles/perf-jni#javavm-and-jnienv>.
- [7] Frida, "JavaScript API | Frida • A world-class dynamic instrumentation framework," [Online]. Available: <https://frida.re/docs/javascript-api/#java>.
- [8] Android Open Source Project, "Linker Namespace," [Online]. Available: <https://source.android.com/docs/core/architecture/vndk/linker-namespace>.
- [9] Android Code Search, "libdl_android.cpp - Android Code Search," [Online]. Available: https://cs.android.com/android/platform/superproject/+/master:bionic/libdl/libdl_android.cpp;l=114-116.

- [10] Google Source, "Bionic," [Online]. Available:
] <https://android.googlesource.com/platform/bionic/+refs/heads/master/README.md#what-are-the-big-pieces-of-bionic>.
- [11] Orange Cyber Defense, "What is binary diffing ?," [Online]. Available:
] <https://www.orangecyberdefense.com/be/insights/blog/research/introduction-to-binary-diffing-part-2>.
- [12] Zynamics, "BinDiff Manual," [Online]. Available: <https://www.zynamics.com/bindiff/manual/index.html#N201AE>.
]
- [13] M. Stone, "What's Up with WhatsApp," 2019. [Online]. Available:
] <https://raw.githubusercontent.com/maddiestone/ConPresentations/master/Jailbreak2019.WhatsUpWithWhatsApp.pdf>.
- [14] Check Point Research, "The NSO WhatsApp Vulnerability – This is How It Happened," 2019. Available:
[<https://research.checkpoint.com/2019/the-nso-whatsapp-vulnerability-this-is-how-it-happened/>
- [15] dev747368, "Twitter," [Online]. Available: <https://twitter.com/dev747368/status/1347360276476293125>.
]
- [16] Frida, "JavaScript API | Frida • A world-class dynamic instrumentation framework," [Online]. Available:
] <https://frida.re/docs/javascript-api/#memory>.
- [17] Frida, "JavaScript API | Frida • A world-class dynamic instrumentation framework." Available:
[<https://frida.re/docs/javascript-api/#interceptor>.
- [18] p. -. stackoverflow, "How to prevent memcpy buffer overflow ." Available:
[<https://stackoverflow.com/a/12210468/18753528>.
- [19] jdh5202, "frida cheatsheet," [Online]. Available: https://jdh5202.tistory.com/908#code_1648721148969.
]
- [20] C. Tamir, "WhatsApp Buffer Overflow Vulnerability: Under the Scope," 2019. [Online]. Available:
] <https://www.zimperium.com/blog/whatsapp-buffer-overflow-vulnerability-under-the-scope/>.