Stagefright: An Android Exploitation Case Study

ZIMPERIUM
MOBILE DEFENSE

Joshua "jduck" Drake
August 9th, 2016
WOOT '16

Agenda

- Introduction
- CVE-2015-3864
- Android Exploitation
- Metaphor
- Improving on Metaphor
- DEMO!
- Porting to a Samsung Device
- DEMO 2!
- Wrap-Up

Introduction

About the presenter and this research

About Joshua J. Drake aka jduck

Focused on vulnerability research and exploit development for the past 17 years

Current Affiliations:

- Found and initially disclosed "Stagefright"
- VP of Platform Research and Exploitation at Zimperium
- Lead Author of Android Hacker's Handbook
- Founder of the #droidsec research group

Previous Affiliations:

- Accuvant Labs (now Optiv), Rapid7 Metasploit, VeriSign iDefense Labs

Motivations

For the work in this talk:

1. Explore the difficulty of exploiting Stagefright bugs
2. Get more people involved in vulnerability research
    - See Vulnerabilities 101, Drake/Christey, DEF CON 24

Originally:

1. Improve the overall state of mobile security
2. Increase visibility of risky code in Android
3. Put the Droid Army to good use!

Led to improvements big and small, but still plenty work left.

1. See Vulnerabilities 101, Drake/Christey, DEF CON 24

Acknowledgments

This work was sponsored by Accuvant Labs (now Optiv) with continuing support from Zimperium.

OPTIV & ZIMPERIUM MOBILE DEFENSE

Special thanks go to Amir Etemadieh of Optiv / Exploiteers.

Additional thanks to Collin Mulliner, Mathew Solnik, Daniel Micay, Gil Dabah, Hanan Be'er

Thanks to USENIX, WOOT, and Natalie Silvanovich for the opportunity!

What is Stagefright?

- Android Multimedia Framework library
  - Written primarily in C++
  - Handles all video and audio files
    - Provides playback facilities
    - Extracts meta-data for the Gallery, etc.

- Now also the name of "a vulnerability" that made waves.[1]
  - An attacker could obtain elevated privileges on an affected Android device, unbeknownst to the victim, with only a single MMS.

1. https://en.wikipedia.org/wiki/Stagefright_%28bug%29

See my 2015 talk slides for more introductory information. (link at end)

CVE-2015-3864

The Vulnerability Exploited

CVE-2015-3864 I: OOPS

When I made my patch for CVE-2015-3824, I missed that *chunk_size* is 64-bit and can be above 2^32.

Using such a value, it was possible to bypass my check:

```
        if (SIZE_MAX - chunk_size <= size) {
            return ERROR_MALFORMED;
        }
```

How embarrassing :-/

CVE-2015-3864 II: Why exploit this one?

- More vulnerable devices as it was patched a month later.
- Exploiting this instead of 2015-3824 is a minor change:

```
-    chunk_size = 0xffffffff - num_write + num_alloc + 1
-    tx3g2 = (pack('>L', chunk_size)+'tx3g') + tx3g2data
+    chunk_size = 0x1ffffffff - num_write + num_alloc + 1
+    tx3g2 = (pack('>L', 1)+'tx3g') + (pack('>Q', chunk_size)+tx3g2data)
```

CVE-2015-3864 III: Root Cause

The vulnerability is an Integer Overflow when allocating *buffer* in the 'tx3g' handling within MPEG4 *parseChunk*.

```
status_t MPEG4Extractor::parseChunk(off64_t offset, int depth) {
...
  case FOURCC('t', 'x', '3', 'g'):
...
      if (!mLastTrack->meta->findData(
              kKeyTextFormatData, &type, &data, &size)) {
...
      uint8_t buffer = new (std::nothrow) uint8_t[size + chunk_size];
...
      mLastTrack->meta->setData(
              kKeyTextFormatData, 0, buffer, size + chunk_size);
```

The *size* value is **accumulated** in *MetaData*. A second 'tx3g' atom can make *size* + *chunk_size* wrap. How is *buffer* used?

From android-5.1.0_r4 (LMY47M) - frameworks/av / media/libstagefright/MPEG4Extractor.cpp

CVE-2015-3864 IV: Consequence

For each 'tx3g' chunk, the data is appended to a temporary buffer (*buffer*) and then saved into the *MetaData*.

```
        if (size > 0) {
            memcpy(buffer, data, size);
        }

        if ((size_t)(mDataSource->readAt(*offset, buffer + size,
                                          chunk_size))
```

We control all the variables, including the allocation size, overflow length, and contents!

From the exploit:

```
    big_num = 0x1fffffff - tx3g_1.length + 1 + vector_alloc_size
    tx3g_2 = get_atom('tx3g', more_data, big_num)
```

Exploitation

Is Android secure yet?

Exploitation I: Crucial Components

Certain system properties are critical to understand for exploitation.

- Heap implementation details - libc
- ASLR Quality (entropy) - kernel

It is difficult (some argue impossible) to eliminate all vulnerabilities in a code base.

Hardening critical system components **can** preventing successful attacks.

Other system-wide mitigations exist and can help too...

Exploitation II: Newer Heap is Weaker

Android is switching to *jemalloc* instead of *dlmalloc*

This new heap implementation is weaker in two ways.

1. Less entropy in heap addresses
    - Easier to guess where your data is in memory
2. No more in-line meta-data
    - *dlmalloc* checks for corrupt meta-data
    - Detected corruption leads to a crash

As a result, exploiting Lollipop/Marshmallow/Nougat is easier than older versions.

Exploitation III: Mitigation Summary

Mitigation	Applicability
SELinux	N/A 1
Stack Cookies	N/A
FORTIFY_SOURCE	N/A
ASLR	only Android >= 4.1
NX	bypass with ROP
GCC new[] mitigation	N/A 2

1. Only comes into play on some devices and only after achieving arbitrary code execution.
2. Only affects some of the vulnerabilities. It still leads to DoS.

Exploitation IV: ASLR

I previously developed 3 exploits.

1. CVE-2015-1538 via MMS on Android 4.0.4
2. CVE-2015-3824 via MediaScanner on Android 5.1
3. CVE-2015-3824 via Browser on Android 5.1
    - It was discussed at ISSW 2016. Ask me for slides later...

I was able to overcome ASLR in each case.

However, ASLR impacts speed and reliability of the exploit.

Enter...

Metaphor

Novel Work on Exploiting CVE-2015-3864

Metaphor I: Key Takeaways

Metaphor proved it was possible to leak information from mediaserver to the browser.

- JavaScript can read the "duration" attribute of an HTML5 video element.
- Using CVE-2015-3864, an attacker can corrupt a pointer that's used to obtain the "duration".

Metaphor II: Details

- MetaData + Vector trickery…
  - Uses MetaData items / Vector storage
  - In memory, the Vector looks like this:

```
  addr           tag       type        len         data
- 0xb4c27bb0:    "heig"    "in32"      0x00000004  0x00000300
+ 0xb4c27bb0:    "dura"    "in64"      0x00000008  0xb4010530
  0xb4c27bc0:    "hvcc"    "hvcc"      0x00000078  0xb4d2e200
  0xb4c27bd0:    "inpS"    "in32"      0x00000004  0x00120000
  0xb4c27be0:    "mime"    "cstr"      0x0000000e  0xb4d2c3a0
  0xb4c27bf0:    "text"    0x00000000  0x00000090  0xb4c27ce0
  0xb4c27c00:    "widt"    "in32"      0x00000004  0x00000400
```

After overwriting the "heig" item with the "dura" tag, type, and length of 8, JavaScript can read from the pointer using the HTML5 video element's "duration" attribute.

Metaphor III: Problems

1. Paper stated multiple devices/firmware versions supported, but exploit only supported one.
2. The leak method had annoying Limitations.
    - Value is rounded and converted to *double*.
        - Can only leak 64-bit values with high 32-bits < 0x1ff
        - Some precision is lost during conversion :-/
3. The technique to utilize the leak required many requests.
    - Testing showed it worked worse than my 3rd exploit.

Improving on Metaphor

What else can we do?

What about height and width?

As mentioned in the Metaphor paper, *videoHeight* and *videoWidth* are also derived from MetaData items read from the file.

- *videoHeight* comes from MetaData items of type 'heig'
- *videoWidth* comes from MetaData items of type 'widt'

These can be populated by including an 'mp4v' atom in the MP4 file. In memory, they look like this:

```
addr           tag       type        len         data
0xb4d14350:    "heig"    "in32"      0x00000004  0x00000300
0xb4d14360:    "hvcc"    "hvcc"      0x00000078  0xb4c6fa00
0xb4d14370:    "inpS"    "in32"      0x00000004  0x00120000
0xb4d14380:    "mime"    "cstr"      0x0000000e  0xb4e42800
0xb4d14390:    "text"    0x00000000  0x00000090  0xb4d14660
0xb4d143a0:    "widt"    "in32"      0x00000004  0x00000400
```

Another Leak Technique!

Partially overwriting MetaData items allows more primitives:

- Change the size of an item (read or write too much)
- Change an element's type (type confusion)

Allows arbitrary write by "updating" MetaData items, and most importantly **leaking allocated pointers**!

```
  addr         tag     type        len         data
- 0xb4d14350:  "heig"  "in32"      0x00000004  0x00000300
- 0xb4d14360:  "hvcc"  "hvcc"      0x00000078  0xb4c6fa00
- 0xb4d14370:  "inpS"  "in32"      0x00000004  0x00120000
- 0xb4d14380:  "mime"  "cstr"      0x0000000e  0xb4e42800
+ 0xb4d14350:  "abc1"  "in32"      0x00000004  0x00007a67
+ 0xb4d14360:  "abc2"  "in32"      0x00000004  0x00007a68
+ 0xb4d14370:  "abc3"  "in32"      0x00000004  0x00007a69
+ 0xb4d14380:  "heig"  "in32"      0x00000004  0xb4e42800
  0xb4d14390:  "text"  0x00000000  0x00000090  0xb4d14660
  0xb4d143a0:  "widt"  "in32"      0x00000004  0x00000400
```

An Interesting Property

My previous experiences told me that this won't be reliable because the buffer would be freed after finishing processing the file.

I decided to give it a try anyway and was surprised.

The buffer lives across multiple requests!

- Browser keeps a connection to the MetadataRetriever.
- MPEG4Extractor data is only freed on disconnect.

So we can leak a pointer and be sure it will be alive on our next request! AWESOME!!

Getting a Code Pointer

The next step in any good modern exploit is to leak a vtable pointer.

MetaData items have been useful so far, can they help here?

Short answer is no.

- Code pointers are never put in MetaData items
- MetaData items don't point to C++ objects

We can't use them directly :-/

Getting a Code Pointer II

After looking around, the most common C++ virtual objects are specializations of *VectorImpl*.

- The first two fields are: vtable pointer and *mStorage*
- *mStorage* is initialized whenever the Vector is pre-sized or any item is added.

This means we can only use the original Metaphor technique if the Vector is empty.

A cursory look for Vectors did not look promising.

Time to look deeper to find out if any such Vectors exist!

Getting a Code Pointer III

After looking at many objects, I found *ONE*.

It turns out *SampleIterator* objects are created without initializing the Vector within. The plan:

1. Allocate a MetaData item with the same size as a *SampleIterator* object.
2. Leak that pointer back to ourselves via *videoHeight*
3. Use the original Metaphor technique to read from an offset of that pointer to get the vtable pointer.

We can use this 2-stage leak to build our ROP chain from libstagefright itself.

Putting it All Together

ASLR still poses a small problem.

We need to point to some memory we control to hijack the execution flow.

We use a large heap spray, which turns out to be very predictable.

This was used with much success in my third exploit too.

The result is a highly reliable and fast exploit.

Key Exploit Details

This exploit has been implemented as a Metasploit module.

- Currently 29 supported targets
  - Includes all vulnerable 5.x Nexus devices
  - Automatically selects a target based on user-agent, contains the precise firmware version!!

    ```
    Mozilla/5.0 (Linux; Android 5.1; Nexus 6 Build/LMY47M)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.84
    Mobile Safari/537.36
    ```

  - This particular exploit is currently limited to Android 5.x devices using jemalloc -- future work
- Only 3 web requests needed

Android Device Diversity

Diversity in the Android ecosystem complicates research, but is not a significant barrier to exploitation.

Exploiting a device usually requires porting/testing per-device-model.

However, automation makes crafting a device-specific exploit rather simple.

- Extract and use key details from each firmware version

BTW, Android browsers are very revealing:

```
Mozilla/5.0 (Linux; Android 5.1; Nexus 6 Build/LMY47M)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.84
  Mobile Safari/537.36
```

Mandated by Android CDD

From the Android 1.6 (and later) Compatibility Definitions Document:

```
...
The user agent string reported by the WebView MUST be in this format:

Mozilla/5.0 (Linux; Android $(VERSION); $(MODEL) Build/$(BUILD)$(WEBVIEW))
AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 $(CHROMIUM_VER)
Mobile Safari/537.36

The value of the $(VERSION) string MUST be the same as the value
for android.os.Build.VERSION.RELEASE.

The $(WEBVIEW) string MAY be omitted, but if included MUST be "; wv"
to note that this is a webview

The value of the $(MODEL) string MUST be the same as the value
for android.os.Build.MODEL.

The value of the $(BUILD) string MUST be the same as the value
for android.os.Build.ID.
```

LIVE DEMO!

Let's see it in action!

See also: https://asciinema.org/a/8jlbdq006wsnkqewvcaf05wva

Porting to a Samsung Device

Because Android is Samsung, right?

Porting to a Samsung Device I

Chosen device - Samsung Galaxy S5 Verizon (SM-G900V)

- Because it was available cheaply...

Porting was not easy.

- Required reverse engineering because Samsung modifies lots and does not release the modified source.
- Debugging is harder because there are no symbols
  - Makes GDB crashy and half-useless.
  - Used IDA Pro android_server

Porting to SM-G900V II

Initial tests did not even reach the vulnerable code!

- Samsung devices actually have **2 different** meta-data processing libraries
  - Proprietary "libsavsmeta.so"
  - The traditional libstagefright

Reaching the libstagefright code required changing the delivery method slightly.

Instead of using *URL.createObjectURL*, we use *FileReader* with *readAsDataURL*.

Porting to SM-G900V III

Samsung modified libstagefright to add additional processing...

- Closed source --> binary diffing required :-/
- One particular change is very interesting!!
  - Usually, errors processing meta-data causes an error processing the file itself
  - Samsung changed "trak" processing to return "OK" on error

This allows repeatedly triggering vulnerabilities in MPEG4Extractor with only one file!!

Porting to SM-G900V IV

Otherwise, only minor adjustments needed.

- Some object sizes changed
- Adds several meta-data items
  - Introduces complexity in Vector overwrite (due to ordering)

In the end, I got it working and even managed to make one module work for both.

LIVE DEMO 2!

Let's see it in action against Samsung!

Just like before, except you get "system" group privileges and can shell without disabling SELinux...

Wrap-Up

Future Work and Conclusions

Future Work

Exploit TODO:

- Support dlmalloc devices (i.e. Android 4.4 and older)
- Support additional devices (esp. non-Nexus)
- Eliminate the need for a heap spray
- Put all exploit logic into the JavaScript only

Android TODO:

- Research Android system_server
  - Shares address space layout with apps!!
  - No ASLR thanks to Zygote!
- Mitigate *android::RefBase* technique

Conclusions I

Exploiting via the browser is awesome

- Auto-targeting thanks to the Android CDD
- Infoleaks workable
- Can hide traffic in https
- Vulnerabilities reachable even if MMS vector patched
- Getting people to click links is easy

Browser based attacks on android devices should be a high concern!

Conclusions II

- Fragmentation is a thorn, but less-so for exploit-dev
  - I'm considering pitching a talk on scaling Android exploit development
- The faster the patching, the better
- Android N raises the bar, but...
  - Adoption remains a problem
  - jemalloc remains still weak (PLZ HARDEN!!)

Releasing the Exploit

The tentative exploit release date is September 2016

Reach out to me if you'd like to contribute!

Thanks for your time!

Any questions?

My 2015 Talk Slides: http://j.mp/stagefright-slides

Prefer to ask offline? Contact me:

Joshua J. Drake
jdrake@zimperium.com
jduck @ Twitter/IRC
www.droidsec.org

the real end. really.

Older Slides....

These are still useful, just no time to talk

Android Exploitability

What stands in the way?

Address Space Layout Randomization

ASLR is the ONLY challenge, and it is not that hard.

I managed to fully bypass ASLR on ICS and Lollipop.

- Information leakage issues
- Heap spraying
  - Address space is usually only 32-bits
  - On 64-bit devices, *mediaserver* remains 32-bit :-/
- Other virtual memory tricks
- Bruteforce or statistical guessing

These tricks are simple but very effective.

CVE-2015-3824 III: What to smash?

Experimentation yielded some interesting crashes.

- The most interesting involved a smashed *mDataSource*.
- Used for a virtual function call just after the overflow!

```
1905         if ((size_t)(mDataSource->readAt(*offset, buffer + size,
                                                  chunk_size))
```

We control the values or contents of almost all of the parameters to the function too!

CVE-2015-3824 IV: Heap Feng Shui

For a reliable exploit, we need *buffer* before *mDataSource* consistently.

Luckily, *jemalloc* also makes heap feng shui easier too. See the paper on it for more details.[1]

- 'covr' #1 - alloc chunk near size of an *MPEG4DataSource*
- 'stbl' - alloc an *MPEG4DataSource* and set to *mDataSource*
- 'covr' #2 - free first 'covr', making a free hole
- 'tx3g' #1 - alloc chunk w/overflow data/size
- 'tx3g' #2 - alloc *buffer* into hole, overflow it

1. Exploiting the jemalloc Memory Allocator, Patroklos Argyroudis and Chariton Karamitas

CVE-2015-3824 V: Heap Spray!

If all goes well, we smahed *mDataSource* and control:

- all member variables
- the virtual function table pointer

To reduce guessing, we point it to a heap spray!

- Same strategy as CVE-2015-1538 exploit
  - As before, a single large chunk
  - *jemalloc* still falls back to *mmap*

- Used a 16MB 'avcC' chunk
  - ~99% predictable in testing!

CVE-2015-3824 VI: Stack Pivot I

We have control of *pc*, but where do we point it?

- Android 4.1+ no longer have predictable *linker*
- Guessing *libc* is apparently 1 in 256 (< 1%)

Daniel Micay (of CopperheadOS) recommended abusing a library loaded by *dlopen*

- Address space would already be stabilized
- Chose the largest library
  - libWVStreamControlAPI_L1.so - 2.4MB
- Could spray more constrain more? (untested)

Result: ~38% success rate per single attempt

CVE-2015-3824 VI: Stack Pivot II

The stack pivot is significantly more complicated.

- Executes in three (!!) stages instead of one.

```
# ldr r2, [r0, #8] ; ldr r3, [r2, #0x28] ; blx r3
mds_pivot1 = mod_base + 0x179202+1

# ldm.w r2!, {r8, sb, sl, fp, ip, sp, pc}
mds_pivot2 = mod_base + 0xc8558+1

# pop {r4, r5, r6, pc}
mds_adjust = mod_base + 0xdbd78+1
```

This complicated dance launches the ROP chain.

CVE-2015-3824 VII: ROP Chain

ROP Stager to make shellcode RWX and run it, like the -1538 exploit, but different...

- Also based on the Widevine streaming library
- Works around ASLR by resolving the *libc* base
- Resolves *mprotect* from the *socket* GOT entry

```
rop += struct.pack('<L', mod_base + ropdict['pop0123'])
rop += struct.pack('<L', mod_base + ropdict['socket_got'])
rop += struct.pack('<L', ropdict['libc_socket'])
rop += struct.pack('<L', ropdict['libc_mprotect'])
scratch_addr = spray_addr + 0xfe0
rop += struct.pack('<L', scratch_addr)  # r3, libc base

# Modify the address to point to mprotect in libc
rop += struct.pack('<L', mod_base + ropdict['deref_r0'])
rop += struct.pack('<L', mod_base + ropdict['subr0r1'])
# Save the libc base address in scratch memory
rop += struct.pack('<L', mod_base + ropdict['strr0r3'])
```

CVE-2015-3824 VIII: Payload

Nexus devices on 5.x+ have SELinux in *enforcing* mode.

- *mediaserver* policy does not allow *execve* :-/
  - no shell for you!
  - Not the case on all Android devices (ahem Samsung)

I developed a kernel exploit (CVE-2015-3636) as a payload!

- Wrote in C first
- Translated to assembly from objdump output
  - That sucked, use gcc -S instead!
- Sets SELinux to permissive mode

Remote kernel FTW! Demo?

Conclusions

What are the key takeaways?

Final Conclusions

1. Take care when changing heap implementations.
   - Changes here can weaken your security posture.
2. Thinking outside the box can make your exploit better!
   - Controlling the environment can influence your target!
3. Diversity is a thorn, but can be dealt with
   - Android Browser user agents are very helpful!
4. Mitigations are not a silver bullet
   - Especially when multiple attempts are possible
5. Vendors using Android need to
   - Be more proactive in finding / fixing flaws
   - Be more aggressive in deploying fixes
6. The Android code base needs more attention. BBMFTW!