Tutorial 4: Integers & Format String

presented by

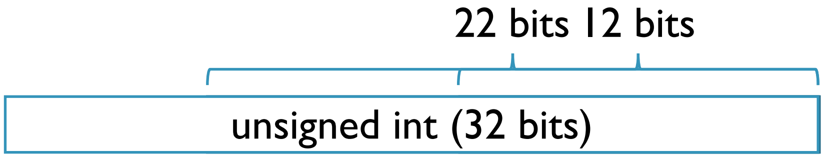
Li Yi
Assistant Professor
SCSE

N4-02b-64
yi_li@ntu.edu.sg

COPYRIGHT STATEMENT

- All course materials, including but not limited to, lecture slides, handout and recordings, are for your own educational purposes only. **All the contents of the materials are protected by copyright, trademark or other forms of proprietary rights.**
- All rights, title and interest in the materials are owned by, licensed to or controlled by the University, unless otherwise expressly stated. **The materials shall not be uploaded, reproduced, distributed, republished or transmitted in any form or by any means, in whole or in part, without written approval from the University.**
- You are also not allowed to take any photograph, film, audio record or other means of capturing images or voice of any contents during lecture(s) and/or tutorial(s) and reproduce, distribute and/or transmit any form or by any means, in whole or in part, without the written permission from the University.
- Appropriate action(s) will be taken against you including but not limited to disciplinary proceeding and/or legal action if you are found to have committed any of the above or infringed the University's copyright.

Working with Integers



22 bits 12 bits

unsigned int (32 bits)

Spot the problem in this ASLR routine, and its impact

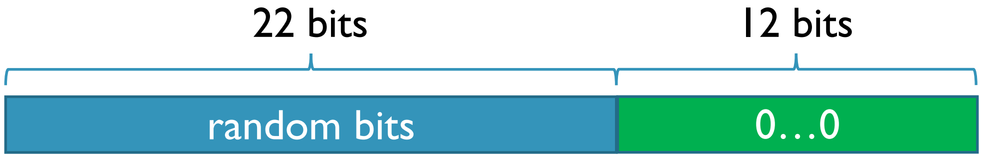
```
static unsigned long randomize_stack_top(unsigned long stack_top)
{
   unsigned int random_variable = 0;
   if ((current->flags & PF_RANDOMIZE) &&
       !(current->personality & ADDR_NO_RANDOMIZE))
      {
        random_variable = get_random_int() & STACK_RND_MASK;
        random_variable <<= PAGE_SHIFT;
      }
#ifdef CONFIG_STACK_GROWSUP
   return PAGE_ALIGN(stack_top) + random_variable;
#else
   return PAGE_ALIGN(stack_top) - random_variable;
#endif
}
```

0x3FFFFF

12 on x86_64

http://hmarco.org/bugs/linux-ASLR-integer-overflow.html

CVE-2015-1593



- Code switches from unsigned long to unsigned int

```
static unsigned long randomize_stack_top(unsigned long stack_top)
{
   unsigned int random_variable = 0;
   if ((current->flags & PF_RANDOMIZE) &&
       !(current->personality & ADDR_NO_RANDOMIZE))
      {
        random_variable = get_random_int() & STACK_RND_MASK;
        random_variable <<= PAGE_SHIFT;
      }
#ifdef CONFIG_STACK_GROWSUP
   return PAGE_ALIGN(stack_top) + random_variable;
#else
   return PAGE_ALIGN(stack_top) - random_variable;
#endif
}
```

0x3FFFFF
mask 22 bits

shift by 12; 2 bits dropped: (22+12)–32 = 2

Entropy is reduced by 4: 2^{30} → 2^{28}

Stagefright

Stagefright Vulnerability

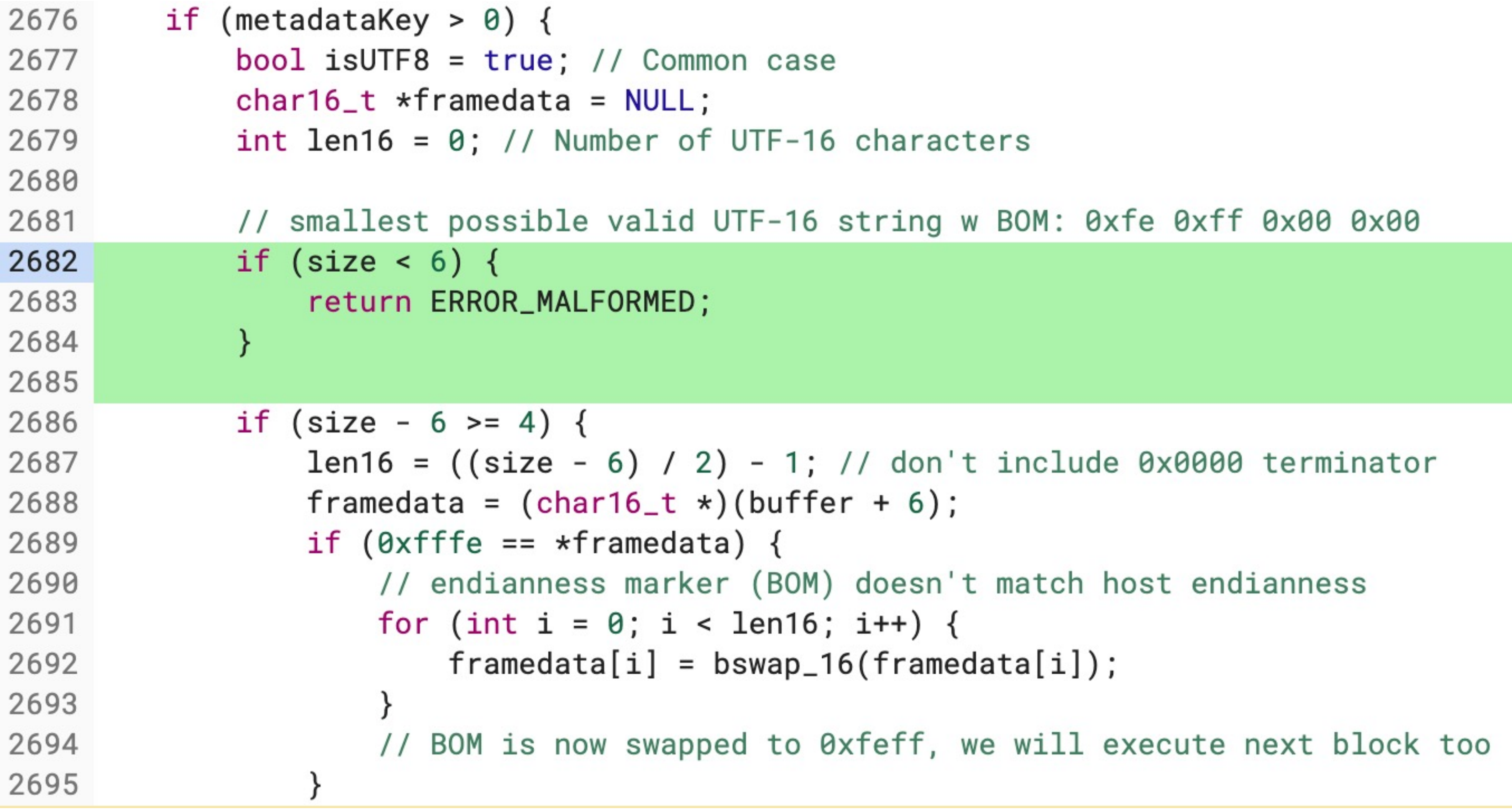
- Stagefright: Android multimedia framework library, e.g., for handling MMS messages, MP4 videos
  - Lots of low level operations on composite data structures
  - Runs with system permissions on many Android phones
- Buffer overflow vulnerability in Stagefright allows an attacker to run arbitrary code with either the "media" or "system" permissions
  - Buffer overflow is possible because of a flawed check on the length of a data structure

Stagefright Vulnerabilities

- CVE-2015-1538 #1 -- MP4 'stsc' Integer Overflow
- CVE-2015-1538 #2 -- MP4 'ctts' Integer Overflow
- CVE-2015-1538 #3 -- MP4 'stts' Integer Overflow
- CVE-2015-1538 #4 -- MP4 'stss' Integer Overflow
- CVE-2015-1539 ------ MP4 'esds' Integer Underflow
- CVE-2015-3824 ------ MP4 'tx3g' Integer Overflow
- CVE-2015-3826 ------ MP4 3GPP Buffer Overread
- CVE-2015-3827 ------ MP4 'covr' Integer Underflow
- CVE-2015-3828 ------ MP4 3GPP Integer Underflow
- CVE-2015-3829 ------ MP4 'covr' Integer Overflow
- ..and a whole slew of stability fixes

https://gerrit.omnirom.org/c/android_frameworks_av/+/14227

[CVE-2015-1539] Input too Small



```
    if (metadataKey > 0) {
        bool isUTF8 = true; // Common case
        char16_t *framedata = NULL;
        int len16 = 0; // Number of UTF-16 characters

        // smallest possible valid UTF-16 string w BOM: 0xfe 0xff 0x00 0x00
        if (size < 6) {
            return ERROR_MALFORMED;
        }

        if (size - 6 >= 4) {
            len16 = ((size - 6) / 2) - 1; // don't include 0x0000 terminator
            framedata = (char16_t *)(buffer + 6);
            if (0xfffe == *framedata) {
                // endianness marker (BOM) doesn't match host endianness
                for (int i = 0; i < len16; i++) {
                    framedata[i] = bswap_16(framedata[i]);
                }
                // BOM is now swapped to 0xfeff, we will execute next block too
            }
```

http://drippler.com/drip/stagefright-explained-exploit-changed-android

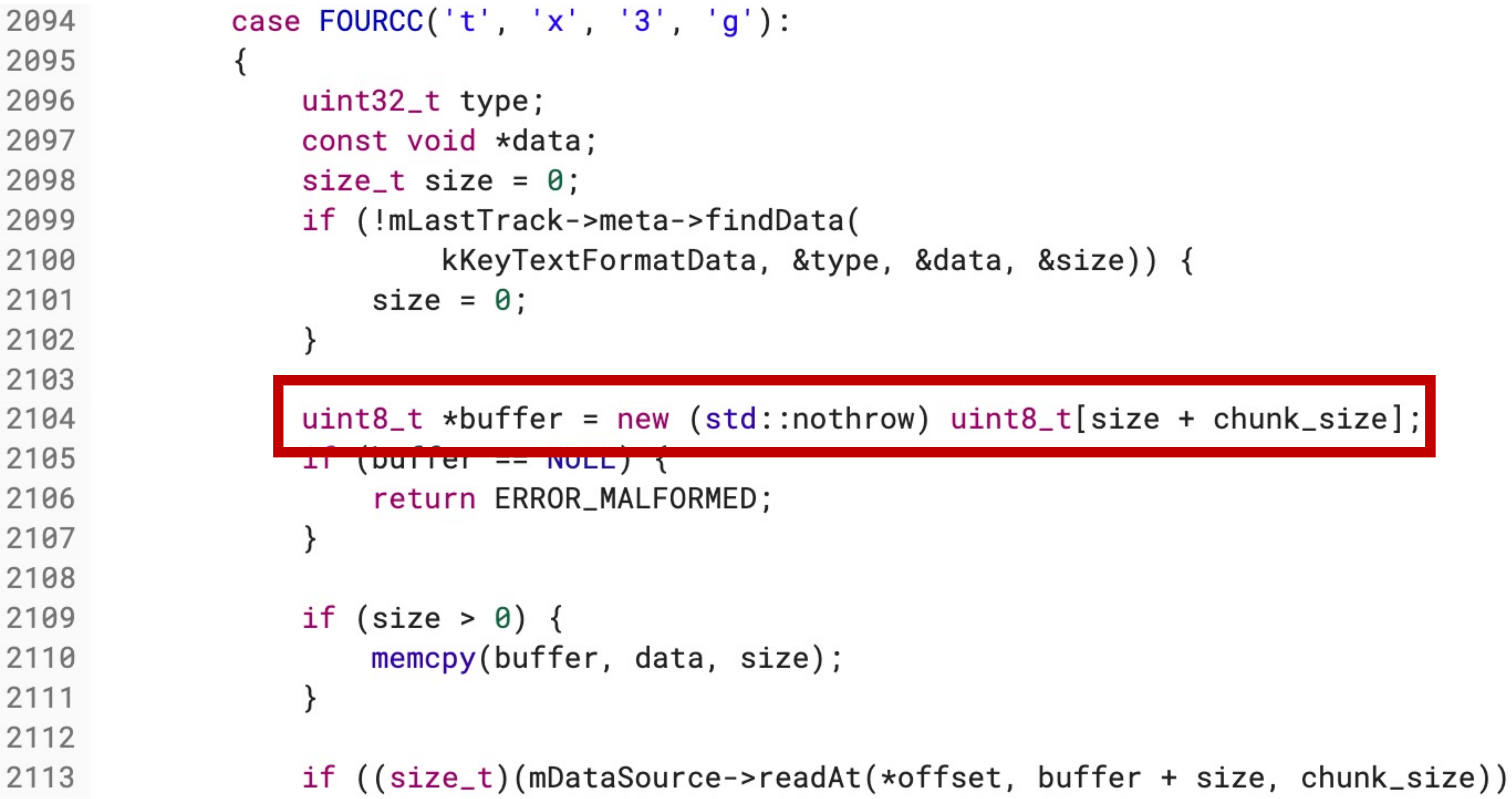
[CVE-2015-1539] Input too Small

- Metadata given as a UTF-16 string
- Shortest possible size of metadata is 6 bytes
- Code that handles metadata subtracts 6 from **`size`**
- For **`size < 6`**, the subtraction underflows and wraps around
  - Result is a very large number
- Frames could then be incorrectly decoded as **`byteswap`** uses a variable whose value is calculated using **`size - 6`**

- Stagefright was not the first to fall into this type of trap!

https://gerrit.omnirom.org/c/android_frameworks_av/+/14228

Integer Overflow [CVE-2015-3824]



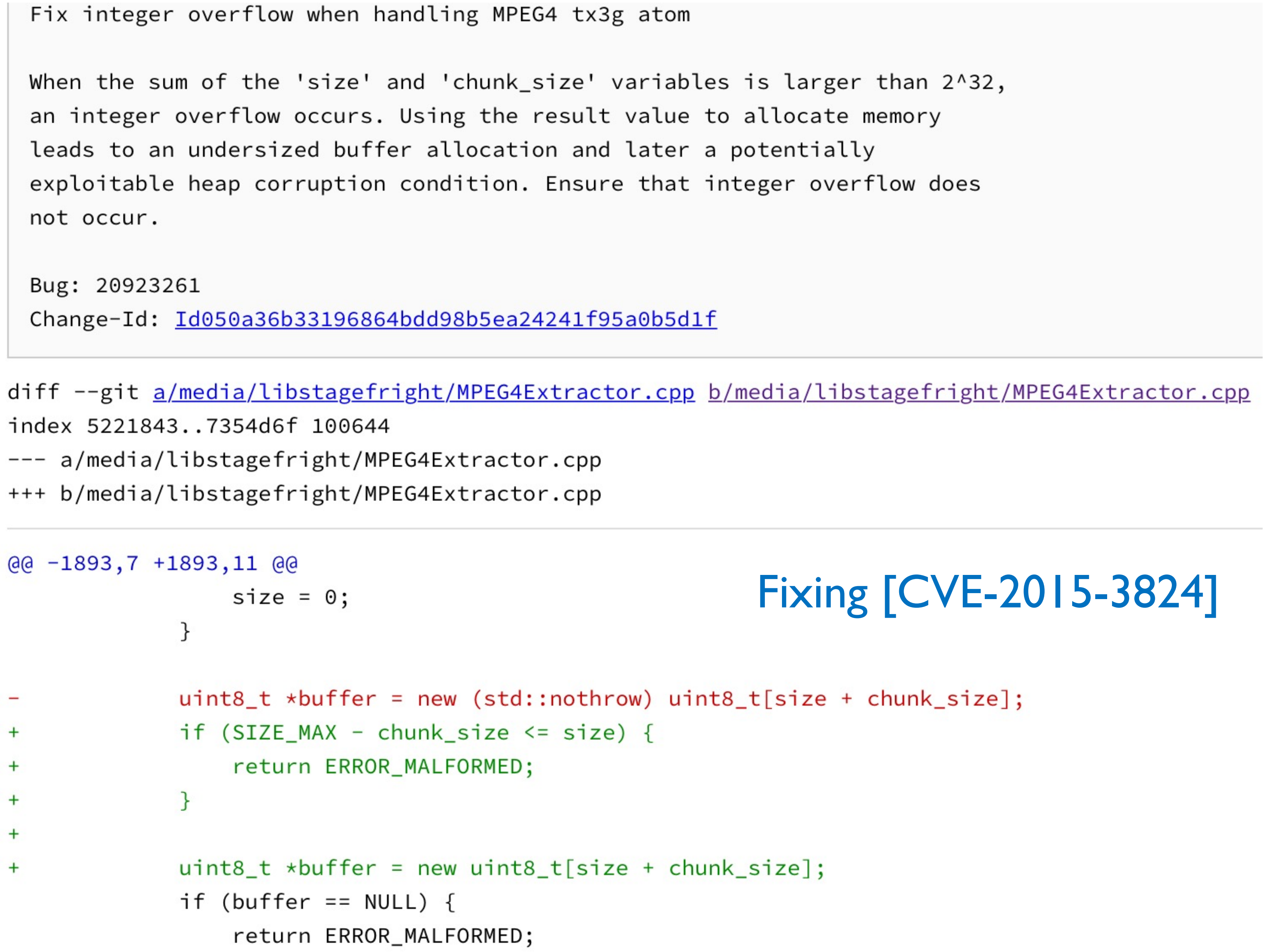
```
case FOURCC('t', 'x', '3', 'g'):
{
    uint32_t type;
    const void *data;
    size_t size = 0;
    if (!mLastTrack->meta->findData(
            kKeyTextFormatData, &type, &data, &size)) {
        size = 0;
    }

    uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
    if (buffer == NULL) {
        return ERROR_MALFORMED;
    }

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

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

http://drippler.com/drip/stagefright-explained-exploit-changed-android

Integer Overflow [CVE-2015-3824]

- Allocating memory of size: `size + chunk_size`
- Can overflow if the sum is big (larger than `2^32`)
- End up with far too little memory allocated in the array
- Potentially lead to exploitable heap corruption condition

Fixing [CVE-2015-3824]



```
Fix integer overflow when handling MPEG4 tx3g atom

When the sum of the 'size' and 'chunk_size' variables is larger than 2^32,
an integer overflow occurs. Using the result value to allocate memory
leads to an undersized buffer allocation and later a potentially
exploitable heap corruption condition. Ensure that integer overflow does
not occur.

Bug: 20923261
Change-Id: Id050a36b33196864bdd98b5ea24241f95a0b5d1f
```

```
diff --git a/media/libstagefright/MPEG4Extractor.cpp b/media/libstagefright/MPEG4Extractor.cpp
index 5221843..7354d6f 100644
--- a/media/libstagefright/MPEG4Extractor.cpp
+++ b/media/libstagefright/MPEG4Extractor.cpp
```

```
@@ -1893,7 +1893,11 @@
                 size = 0;
             }
 
-            uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
+            if (SIZE_MAX - chunk_size <= size) {
+                return ERROR_MALFORMED;
+            }
+
+            uint8_t *buffer = new uint8_t[size + chunk_size];
             if (buffer == NULL) {
                 return ERROR_MALFORMED;
```

https://android.googlesource.com/platform/frameworks/av/+/463a6f807e187828442949d1924e143cf07778c6 https://www.usenix.org/sites/default/files/conference/protected-files/woot16_slides_drake.pdf

[CVE-2015-3864] Fixing Fixes …

- "When I made my patch for CVE-2015-3824, I missed that **`chunk_size`** is 64-bit and can be above 2^{32}"
- With such a value, the check could be bypassed:

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

- Know your units of measurement!

https://nvd.nist.gov/vuln/detail/CVE-2015-3864 https://www.blackhat.com/docs/us-15/materials/us-15-Drake-Stagefright-Scary-Code-In-The-Heart-Of-Android.pdf

Stagefright – Size Check

```
mTimeToSampleCount = U32_AT(&header[4]);

uint64_t allocSize = mTimeToSampleCount * 2 *
sizeof(uint32_t);
if (allocSize > SIZE_MAX) {
    return ERROR_OUT_OF_RANGE;
}

mTimeToSample = new uint32_t[mTimeToSampleCount * 2];

size_t size = sizeof(uint32_t) * mTimeToSampleCount *
2;
```

32-bit Integer Arithmetic

- In C, the product of two 32-bit integers is a 32-bit integer; the upper 32 bits of the result are lost
  - All factors in the calculation of `allocSize` are of type `uint32_t`
  - `SIZE_MAX` = 2^{32}
- Buffer overflows would not be detected!
- Flawed check makes memory corruption in the heap possible; send malformed MP4 video to overwrite memory locations that give control to the attacker

Stagefright – Summary

- A very powerful set of attacks
  - Large number of devices affected
  - No user interaction needed
  - Attacker just sends an MMS to a phone
- Root cause of the attacks: flaws in integer operations
  - Further technical details need to be explored to turn this vulnerability into an exploit
- Mitigation: ASLR makes this attack more difficult
- Pointer ahead in the course: search for vulnerabilities was conducted with fuzzing

Format String Demo

Read Secret using Format String

```
void vuln(char *user_input){
  char buf[128];

  strcpy(buf, user_input);
  printf(buf);
  printf("\n");
}

int main(int argc, char **argv) {
  char *secret = (char *) malloc(5);
  strcpy(secret, "4067");
  printf("secret is at: %p\n", secret);

  vuln(argv[1]);
}
```

- Disable ASLR (on Linux)
  - `echo 0 | sudo tee /proc/sys/kernel/randomize_va_space`
- Locate constant string on the stack
  - `./format “AAAA$(python -c 'print “%08x ”*20')”`
- Print secret
  - `./format $(python -c 'print “\x60\x81\x55\x56” + “%4$s”')`

Take Aways

- Dangers of abstraction: one may slip up when using 32-bit and 64-bit integers at the same time
  - Programming with a 32-bit language on a 64-bit machine?
- Predictable memory allocation is bad for security
- Next lecture / tutorial: what to leave behind in freed memory?