Nanyang Technological University

AY2021-2022 Semester 2

CE/CZ4067 SOFTWARE SECURITY

Tutorial 4: Integers and Format String

1. Recall the `randomize_stack_top` function in Linux kernel from Tutorial 3. An older version of the function (Fig. 1) contains integer overflow bugs.

```
1  static unsigned long randomize_stack_top(unsigned long stack_top)
2  {
3    unsigned int random_variable = 0;
4
5    if ((current->flags & PF_RANDOMIZE) &&
6    !(current->personality & ADDR_NO_RANDOMIZE)) {
7      // STACK_RND_MASK is 0x3fffff on x86_64
8      random_variable = get_random_int() & STACK_RND_MASK;
9      random_variable <<= PAGE_SHIFT; // 12 on x86_64
10   }
11   return PAGE_ALIGN(stack_top) + random_variable;
12   return PAGE_ALIGN(stack_top) - random_variable;
13 }
```

Figure 1: The `randomize_stack_top` function with integer overflow bugs.

   (a) Point out which line(s) may contain integer overflow bugs and how to fix them?

   (b) How does the bug affect the entropy of the stack randomization?

2. Stagefright is the name given to a group of software bugs that affect versions 2.2 "Froyo" of the Android operating system. The name is taken from the affected library, which among other things, is used to unpack MMS messages [1]. Exploitation of the bug allows an attacker to perform arbitrary operations on the victim's device through remote code execution and privilege escalation [2].

   The discovered bugs have been provided with multiple Common Vulnerabilities and Exposures (CVE) identifiers, CVE-2015-1538, CVE-2015-1539, CVE-2015-3824, CVE-2015-3826, CVE-2015-3827, CVE-2015-3828, CVE-2015-3829 and CVE-2015-3864 (the latter one has been assigned separately from the others), which are collectively referred to as the Stagefright bug [1, 3].

(a) Figure 2 shows the code snippet related to CVE-2015-1539. Which line(s) of code may contain an integer overflow/underflow bug? What might be the consequences of this bug? How do you fix it?

```
1  if (metadataKey > 0) {
2    bool isUTF8 = true; // Common case
3    char16_t *framedata = NULL;
4    int len16 = 0; // Number of UTF-16 characters
5
6    // smallest possible valid UTF-16 string w BOM: 0xfe 0xff 0x00 0x00
7    if (size - 6 >= 4) {
8      len16 = ((size - 6) / 2) - 1; // don't include 0x0000 terminator
9      framedata = (char16_t *)(buffer + 6);
10     if (0xfffe == *framedata) {
11       // endianness marker (BOM) doesn't match host endianness
12       for (int i = 0; i < len16; i++) {
13         framedata[i] = bswap_16(framedata[i]);
14       }
15       // BOM is now swapped to 0xfeff, we will execute next block too
16     }
```

Figure 2: Code snippet from `media/libstagefright/MPEG4Extractor.cpp`.

(b) Figure 3 shows the code snippet related to CVE-2015-3824. Line 10 contains an integer overflow bug. When does the overflow happen? What might be the consequences of this bug? How do you fix it?

```
1  uint32_t type;
2  const void *data;
3  size_t size = 0;
4  if (!mLastTrack->meta->findData(
5      kKeyTextFormatData, &type, &data, &size)) {
6    size = 0;
7  }
8
9  // chunk_size is uint64_t
10 uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
11 if (buffer == NULL) {
12   return ERROR_MALFORMED;
13 }
14
15 if (size > 0) {
16   memcpy(buffer, data, size);
17 }
18
19 if ((size_t)(mDataSource->readAt(*offset, buffer + size, chunk_size))
```

Figure 3: Code snippet from `media/libstagefright/MPEG4Extractor.cpp`.

3. Compile and run the program shown in Fig. 4 with ASLR disabled.

```
1  void vuln(char *user_input){
2    char buf[128];
3
4    strcpy(buf, user_input);
5    printf(buf);
6    printf("\n");
7  }
8
9  int main(int argc, char **argv) {
10   char *secret = (char *) malloc(5);
11   strcpy(secret, "4067");
12   printf("secret is at: %p\n", secret);
13
14   vuln(argv[1]);
15 }
```

Figure 4: A simple C program with the format string vulnerability.

(a) Assume the executable is named `format`. Run `./format` to determine the memory address of "`secret`".

(b) What is the output of the following command? Locate the constant string in the output.

```
./format "AAAA$(python -c 'print "%08x "*20')"
```

(c) Craft an input to the executable to read from the secret address.

References

[1] Stagefright: Everything you need to know about Google's Android megabug. https://fortune.com/2015/07/28/stagefright-google-android-security/, November 2021.

[2] Zimperium. How to protect from StageFright vulnerability. https://blog.zimperium.com/how-to-protect-from-stagefright-vulnerability/, November 2021.

[3] Zimperium. Stagefright: Vulnerability details, Stagefright detector tool released. https://blog.zimperium.com/stagefright-vulnerability-details-stagefright-detector-tool-released/, November 2021.