Secure Programming Coursework

Jure Sternad (s2450797)

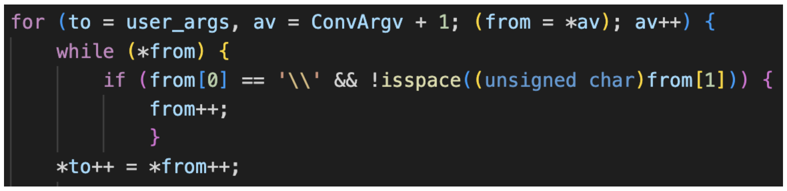
November 25, 2022

1 Secure Coding

1. There is a program vulnerable.c, a toy program for parsing commandline arguments. It contains a major insecure coding practice. Identify the insecure coding practice in the program vulnerable, with reference to its CWE. Provide a short script called exploit that demonstrates that this program is problematic.

Answer

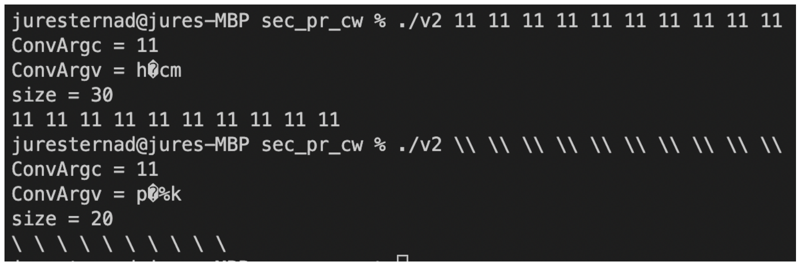
CWE-122: Heap-based Buffer Overflow: The vulnerability occurs in the following snippet.



```c
for (to = user_args, av = ConvArgv + 1; (from = *av); av++) {
    while (*from) {
        if (from[0] == '\\' && !isspace((unsigned char)from[1])) {
            from++;
            }
    *to++ = *from++;
```

Since the size of "\\" is read as 1, and in the `while` loop, only `from` is increased by 1, if the input is "\\", an attacker can exploit that. He can write such a sequence of "\\", which increases `from` enough to move the `*to` pointer past the allocated memory, resulting in a heap overflow.

I have helped myself with the following example.



```
juresternad@jures-MBP sec_pr_cw % ./v2 11 11 11 11 11 11 11 11 11 11
ConvArgc = 11
ConvArgv = h�cm
size = 30
11 11 11 11 11 11 11 11 11 11
juresternad@jures-MBP sec_pr_cw % ./v2 \\ \\ \\ \\ \\ \\ \\ \\ \\ \\
ConvArgc = 11
ConvArgv = p�%k
size = 20
\ \ \ \ \ \ \ \ \ \
```

3. Assuming that the insecure coding practice in vulnerable.c existed in a larger, feature-rich program, describe how it may be exploited. Make explicit reference to what the impact of an attack may be. Support your speculation with reference to at least one CVE(s) with the same CWE as you identified in Q2.1.

Answer

Heap overflows aim to abuse the functions that manage the heap memory or to alter the data on the heap. An attacker can usually make an input that is not correctly validated and is copied to a block on the heap. By that, an attacker can write past the bounds of that block into the next block. If he can craft an input that frees the next block, he can write directly into the arbitrary address in the memory and execute the arbitrary code.

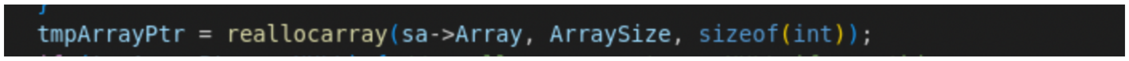
An example is **CVE-2009-2523 - License Logging Server Heap Overflow Vulnerability**, where a heap overflow was caused by improper validation of the length of the string passed in. An attacker would craft an RPC message requesting a service from a program on another computer in the network that runs the vulnerable component (License Logging service). By that, he could gain complete control of the system.

4. There is another program vulnerable2.c, a toy program for printing random arrays according to some conditions. It contains a major insecure coding practice.

Identify the insecure coding practice in the program vulnerable2, with reference to its CWE, describing its root cause in detail. Provide a short script called exploit2 that demonstrates that this program is problematic.

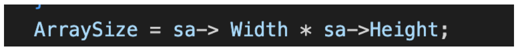
Answer

CWE-415: Double free: The problem can arise in the following line of code.



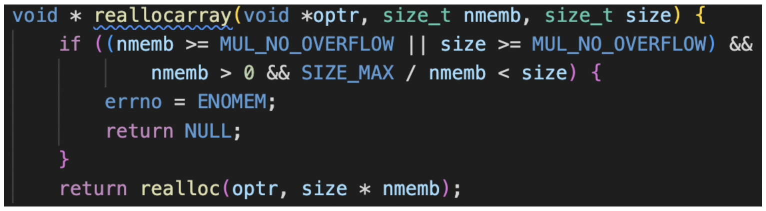
```
tmpArrayPtr = reallocarray(sa->Array, ArraySize, sizeof(int));
```

Since the user can input 0 as either height or weight (or both), the variable `ArraySize` can be set to 0.



```
ArraySize = sa-> Width * sa->Height;
```

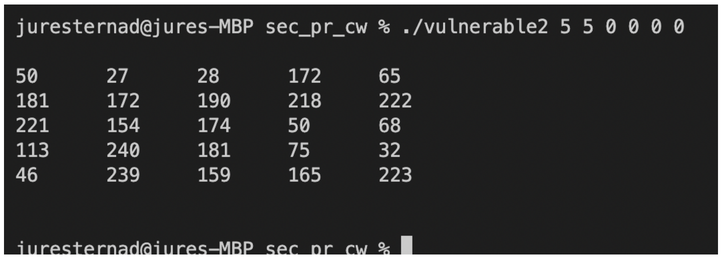
If the `ArraySize` is 0, the `reallocarray` acts as a `free()` function.



```
void * reallocarray(void *optr, size_t nmemb, size_t size) {
    if ((nmemb >= MUL_NO_OVERFLOW || size >= MUL_NO_OVERFLOW) &&
            nmemb > 0 && SIZE_MAX / nmemb < size) {
        errno = ENOMEM;
        return NULL;
    }
    return realloc(optr, size * nmemb);
```

An attacker can call realloc() with a new size of 0 and the same buffer will be freed again. Then, he can write to the same buffer. An input of a positive height and weight of the first array, followed by two array dimensions consisting of at least one 0, will result in `reallocarray` double freeing the same block of memory.

Interestingly, that was not the case on my MacBook with a pro m1 chip.



```
juresternad@jures-MBP sec_pr_cw % ./vulnerable2 5 5 0 0 0 0

50      27      28      172     65
181     172     190     218     222
221     154     174     50      68
113     240     181     75      32
46      239     159     165     223


juresternad@jures-MBP sec_pr_cw %
```

6. Assuming that the insecure coding practice in vulnerable2.c existed in a larger, feature-rich program, describe how it may be exploited. Make explicit reference to what the impact of an attack may be. Support your speculation with detailed reference to at least one CVE(s) with the same CWE as you identified in Q2.4.

Answer

The double-free vulnerability can cause the corruption of memory management data structures. When the function `free()` frees the buffer and sets the pointers of the adjacent buffers to point to each other, it allows the space of the removed buffer to be reused. A double-free fault occurs when an already free buffer is unlinked from the linked list. If the memory block is entered into the free buffer linked list twice, it can be later returned from an allocation request twice- resulting in storing different objects in the same block twice. An attacker can exploit that by writing the arbitrary code past the end of the allocated memory, which is later executed by root privileges.

An example can be seen in the **CVE-2019-8635: Apple macOS Double Free Vulnerability.** A vulnerability, which allowed a local attacker to execute code, occurred in the AMD component. The cause was the improper validation of the data supplied by the user, which could result in a double free flaw. That allowed an attacker to write arbitrary code past the allocated memory and expand privileges to the kernel level.