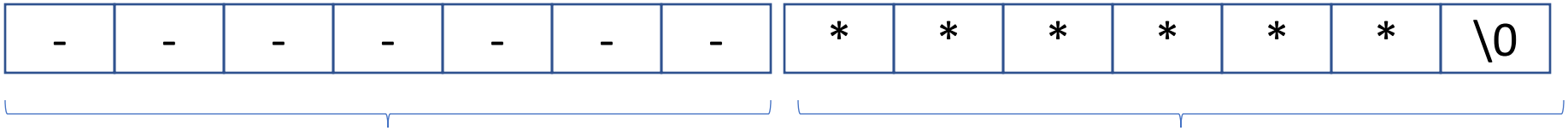
```
int password_verify(){
        // Assume password is of length 6
        char password_stored[7]; // one extra for \0
        char received_password[7];
        FILE *fp;

        // Program reads password from file
        fp = fopen("secret_file", "r");
        fscanf(fp, "%s", password_stored);
        fclose(fp);

        // Program receives user-input
        printf("Enter 6 letter password: ");
        scanf("%s", received_password);

        // Verify password char-by-char
        int i;
        for(i=0; i<6; i++){
                if(received_password[i] != password_stored[i]){
                        printf("Password not matched\n");
                        exit(-1);
                }
        }
        printf("Password matched! Welcome!\n");
        secret_function();
        return 0;
}
```

Here is an example of password verification.

If user-provided password matches with the stored password, then a secret function is called.

```
int password_verify(){
        // Assume password is of length 6
        char password_stored[7]; // one extra for \0
        char received_password[7];
        FILE *fp;
        . . .
}
```

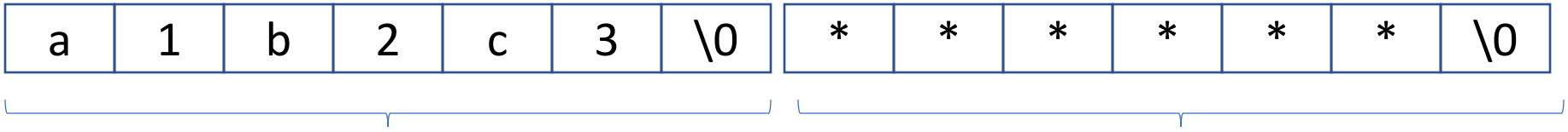
Memory allocation of two arrays in the Stack frame of password_verify()



* is used to represent secret char

```
int password_verify(){
        // Assume password is of length 6
        char password_stored[7]; // one extra for \0
        char received_password[7];
        FILE *fp;
        . . .
        // Program receives user-input
        printf("Enter 6 letter password: ");
        scanf("%s", received_password);
        . . .
}
```

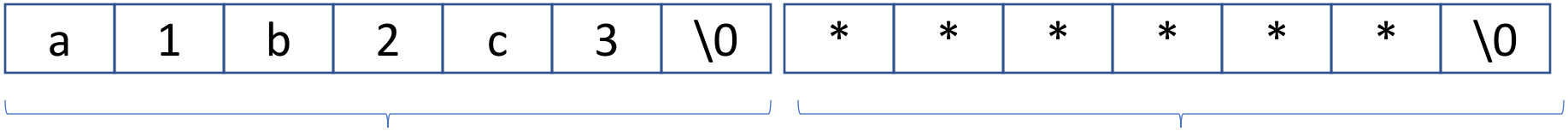
User provides a 6-char long string, say “a1b2c3” as a password



* is used to represent secret char

```
int password_verify(){
        // Assume password is of length 6
        char password_stored[7]; // one extra for \0
        char received_password[7];
        FILE *fp;
        . . .
        // Program receives user-input
        printf("Enter 6 letter password: ");
        scanf("%s", received_password);
        . . .
}
```

User provides a 6-char long string, say “a1b2c3” as a password



char received_password[7]

char password_stored[7]

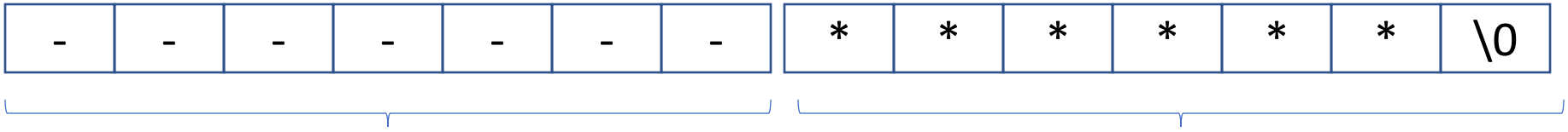
* is used to represent secret char

Password verification fails as soon as a mismatch is found

```
int password_verify(){
        // Assume password is of length 6
        char password_stored[7]; // one extra for \0
        char received_password[7];
        FILE *fp;
        . . .
        // Program receives user-input
        printf("Enter 6 letter password: ");
        scanf("%s", received_password);
        . . .
}
```

'Nasty' user enters a much longer string, say
"aaaaaaaaaaaaaaaaaa"

What happens next?



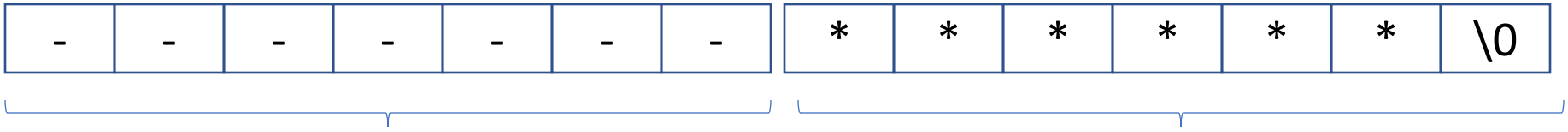
* is used to represent secret char

```
int password_verify(){
        // Assume password is of length 6
        char password_stored[7]; // one extra for \0
        char received_password[7];
        FILE *fp;
        . . .
        // Program receives user-input
        printf("Enter 6 letter password: ");
        scanf("%s", received_password);
        . . .
}
```

scanf() doesn't check length of input string. It writes everything into the buffer.

'Nasty' user enters a much longer string, say "aaaaaaaaaaaaaaaaaa"

What happens next?



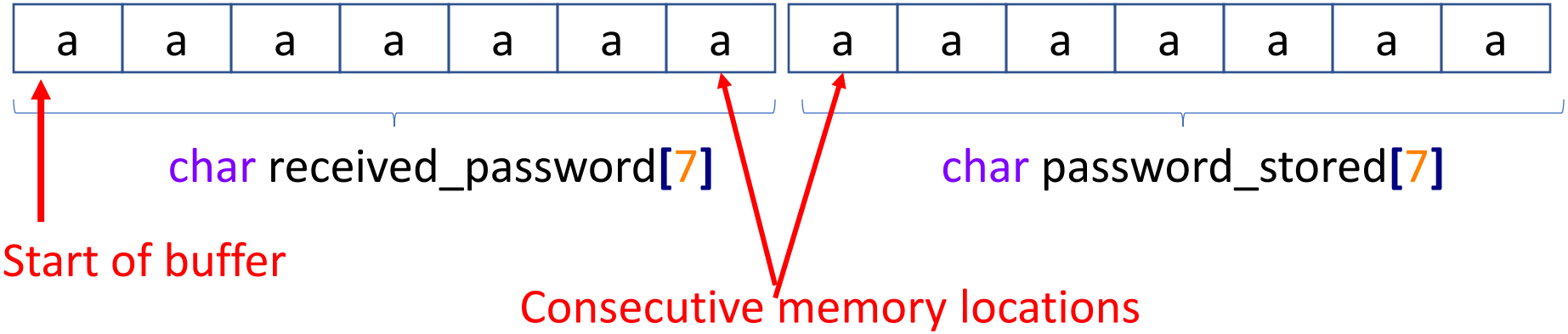
* is used to represent secret char

```
int password_verify(){
        // Assume password is of length 6
        char password_stored[7]; // one extra for \0
        char received_password[7];
        FILE *fp;
        . . .
        // Program receives user-input
        printf("Enter 6 letter password: ");
        scanf("%s", received_password);
        . . .
}
```

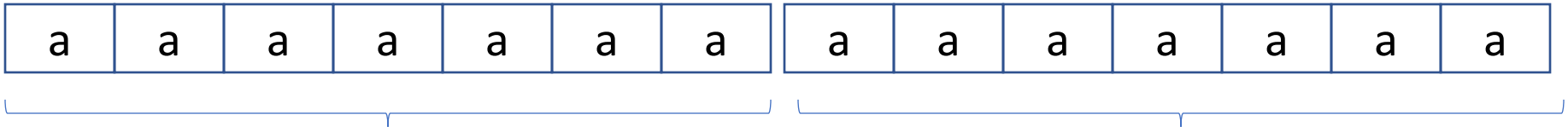
scanf() doesn't check length of input string. It writes everything into the buffer.

'Nasty' user enters a much longer string, say "aaaaaaaaaaaaaaaaaa"

Correct password is overwritten



```
int password_verify(){
        . . .
        // Verify password char-by-char
        for(i=0; i<6; i++){
                if(received_password[i] != password_stored[i]){
                        printf("Password not matched\n");
                        exit(-1);
                }
        }
        printf("Password matched! Welcome!\n");
        secret_function();
        . . .
}
```



Now, they match

Attacker wins in calling secret_function();

This bug is known as the ‘Buffer overflow bug’

How to prevent buffer overflow

```
int password_verify(){
   . . .
   // Program receives user-input
   printf("Enter 6 letter password: ");
   //scanf("%s", received_password);
   fgets(received_password, sizeof(received_password), stdin);
   . . .
}
```

This is how you read a string and prevent buffer overflows in C:

fgets(buffer, sizeof(buffer), stdin);

fget() ensures that only sizeof(buffer)-bytes are read from standatd input.