Liam Nicoll

ASMboy’s segfault Writeup

Intro

I wanted to reverse engineer the segfault challenge created by ASMboy from crackmes.one, hyperlinked here.

Analysis

I ran `file a.out` to ensure the filetype of the binary:

```
$ file a.out

a.out: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, not stripped
```

This confirmed it was a 64-bit ELF binary with debugging symbols intact.

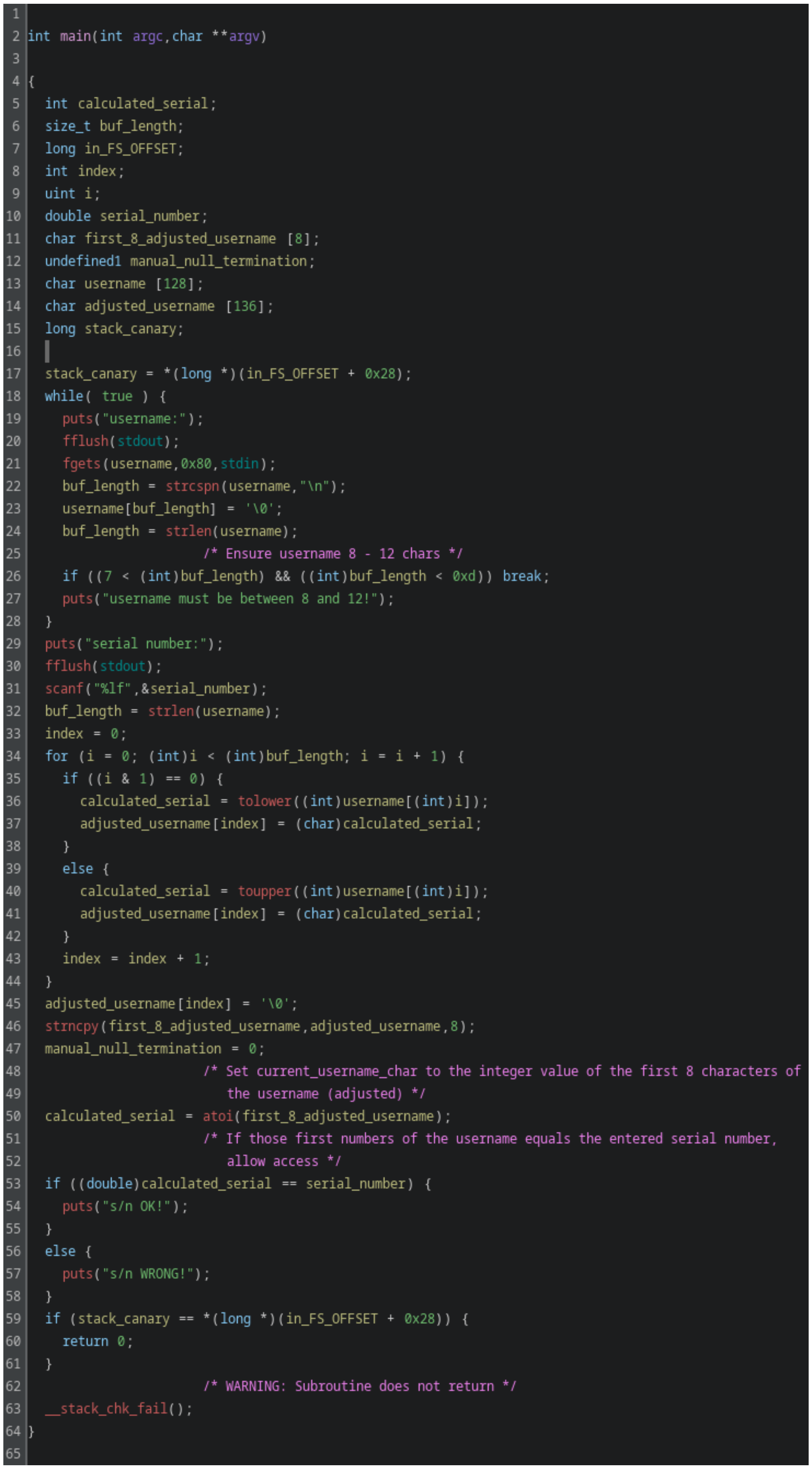
I then ran `strings a.out` to get an idea of what was involved in the program. Running `strings a.out` revealed several interesting artifacts:

```
$ strings a.out
...
username:
username must be between 8 and 12!
serial number:
s/n OK!
s/n WRONG!
...
```

This indicated the program follows a username/serial validation flow, with the username constrained to 8-12 characters. I did not observe a hardcoded serial number, which suggests that the serial number is going to be calculated based on the username.

After using strings, I attempted to run the program. As expected, it asked for a username, and when given a username that is between 8 and 12 characters, it asks for a serial number. With the inputted username of testuser and inputted serial number of 123, it said the serial number was incorrect.

I analyzed the binary in Ghidra and renamed the variables for clarity (see Figure 1). You can separate the binary into three distinct sections: input and validation, the username transformation algorithm, and the serial validation.



```

int main(int argc,char **argv)

{
  int calculated_serial;
  size_t buf_length;
  long in_FS_OFFSET;
  int index;
  uint i;
  double serial_number;
  char first_8_adjusted_username [8];
  undefined1 manual_null_termination;
  char username [128];
  char adjusted_username [136];
  long stack_canary;
  
  stack_canary = *(long *)(in_FS_OFFSET + 0x28);
  while( true ) {
    puts("username:");
    fflush(stdout);
    fgets(username,0x80,stdin);
    buf_length = strcspn(username,"\n");
    username[buf_length] = '\0';
    buf_length = strlen(username);
                    /* Ensure username 8 - 12 chars */
    if ((7 < (int)buf_length) && ((int)buf_length < 0xd)) break;
    puts("username must be between 8 and 12!");
  }
  puts("serial number:");
  fflush(stdout);
  scanf("%lf",&serial_number);
  buf_length = strlen(username);
  index = 0;
  for (i = 0; (int)i < (int)buf_length; i = i + 1) {
    if ((i & 1) == 0) {
      calculated_serial = tolower((int)username[(int)i]);
      adjusted_username[index] = (char)calculated_serial;
    }
    else {
      calculated_serial = toupper((int)username[(int)i]);
      adjusted_username[index] = (char)calculated_serial;
    }
    index = index + 1;
  }
  adjusted_username[index] = '\0';
  strncpy(first_8_adjusted_username,adjusted_username,8);
  manual_null_termination = 0;
                    /* Set current_username_char to the integer value of the first 8 characters of
                       the username (adjusted) */
  calculated_serial = atoi(first_8_adjusted_username);
                    /* If those first numbers of the username equals the entered serial number,
                       allow access */
  if ((double)calculated_serial == serial_number) {
    puts("s/n OK!");
  }
  else {
    puts("s/n WRONG!");
  }
  if (stack_canary == *(long *)(in_FS_OFFSET + 0x28)) {
    return 0;
  }
                    /* WARNING: Subroutine does not return */
  __stack_chk_fail();
}

```

Figure 1: Full main function with renamed variables

The core algorithm works as follows:

```
// Get username (must be 8-12 chars)
while (true) {
    printf("username:\n");
    fgets(username, 128, stdin);
    remove_newline(username);

    if (strlen(username) >= 8 && strlen(username) <= 12)
        break;
    printf("username must be between 8 and 12!\n");
}
```

The above section has the user enter a username until the user provides input between 8 and 12 characters long, inclusive.

```
// Transform username: even positions -> lowercase, odd -> uppercase
for (i = 0; i < strlen(username); i++) {
    if (i % 2 == 0)
        transformed_username[i] = tolower(username[i]);
    else
        transformed_username[i] = toupper(username[i]);
}
```

The above section changes the inputted username to alternate the cases. Characters at even indices will be changed to lowercase, and characters at odd indices will be changed to uppercase. For example, if the user inputs “TestUser” as the username, the main function would transform the username to “tEsTuSeR”.

```
// Get serial number input
printf("serial number:\n");
scanf("%lf", &input_serial);

// Calculate expected serial from first 8 chars
strncpy(serial_buffer, transformed_username, 8);
calculated_serial = atoi(serial_buffer);

// Validate
if (calculated_serial == input_serial)
    printf("s/n OK!\n");
else
```

```
    printf("s/n WRONG!\n");
}
```

This section of code copies the first 8 characters of the transformed username to a buffer, and then calculates the serial by running atoi() on that buffer. This means if the user enters integers as the username, the corresponding serial number will be calculated as the first 8 digits of that integer, or until the function reaches non-integers. If the user does not enter a username that begins with an integer, the calculated serial will be 0.

Conclusion

The segfault crackme is a simple username/serial number authentication program. The program transforms the username by alternating character case (lowercase for even positions, uppercase for odd positions), then extracts the first 8 characters of the transformed result and converts them to an integer using atoi(); this integer is the valid serial number. For numeric usernames (like "12345678"), the transformation has no effect since digits are unchanged by case conversion, making the serial number simply the first 8 digits of the username. Despite its name, the program appears to be well-protected against segmentation faults and doesn't crash under normal or malicious input.