

A solution to “f-crackme”

A. S. “Aleksey” Ahmann*
<hackermaneia@riseup.net>

December 17, 2021

Contents

1	Introduction: the problem	2
2	A solution to the problem	2
2.1	Getting high: From decompiled source code to a high-level abstraction of f-crackme	3
2.2	“Eyeballing” the source code for clues	4
2.3	The proper length for <code>user</code>	6
2.4	Comparing the username and password inputs	6
2.5	Writing the keygen	7
3	Summary	8
3.1	Supplementary materials	8
A	Decompiler results	8
A.1	<code>main.c</code>	8
A.2	<code>checkUsername.c</code>	9
B	Python port of f-crackme	10
C	Keygen	11

*GitHub: @Alekseyyy; crackmes.one profile: @RelationalAlgebra

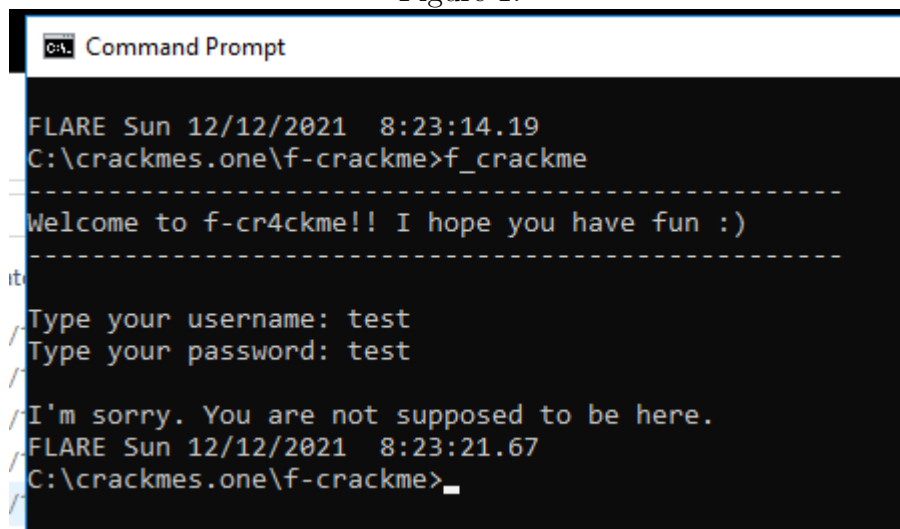
1 Introduction: the problem

*f-crackme*¹ is a simple crackme puzzle written by tk334.² It asks for a “username” and a “password,” and failure to enter in the proper parameters results in an error message. This writeup will discuss a reverse engineering method to avoid the error message and get the programme to execute *as if* the correct parameters have been supplied without the application of a hotpatch.

2 A solution to the problem

As the introduction points out, *f-crackme* asks for a “username” and a “password.” Failure to enter in the proper input results in an error message. A cursory inspection of the application (Figure 1) demonstrates this with the error message: “*I’m sorry. You are not supposed to be here.*”

Figure 1:



```
Command Prompt
FLARE Sun 12/12/2021 8:23:14.19
C:\crackmes.one\f-crackme>f_crackme
-----
Welcome to f-cr4ckme!! I hope you have fun :)
-----
it
Type your username: test
Type your password: test
I'm sorry. You are not supposed to be here.
FLARE Sun 12/12/2021 8:23:21.67
C:\crackmes.one\f-crackme>
```

The problem here is to devise a means as to bypass the error message. A necessary component to a correct solution to this problem is to circumvent the error message *without applying any hotpatches to the executable*.

¹Problem link (Retrieved on 12 Dec., 2021):
<https://crackmes.one/crackme/6194f35633c5d44c61906fe6>

²Their profile (Retrieved on 12 Dec., 2021):
<https://crackmes.one/user/tk334>

2.1 Getting high: From decompiled source code to a high-level abstraction of f-crackme

My initial reverse engineering method was to tinker with a variety of disassembly and debugging tools, and I finally decided on the following strategy: I will use the *Ghidra* reverse engineering framework to disassemble and decompile f-crackme, and then create a high-level representation,³ to the best of my ability, of the executable in the *Python* programming language.

Having Ghidra perform an automatic analysis of the binary, I have worked out the entry point of f-crackme to be `main`.⁴ Rather than discussing the nitty-gritty details of the C decompile output, I want to abstract and simplify it into a more formal means as shown in Algorithms 1 and 2.

Algorithm 1 The main logic of f-crackme (Simplified)

```
1: procedure MAIN
2:   print_welcome()
3:   user ← stdin
4:   pass ← stdin
5:   local_c ← checkUsername(user)
6:   local_68, local_60 ← None
7:   if (local_c = 0) then
8:     local_60 ← "\x79"
9:     local_68 ← "\x74\x65\x69\x63\x66\x73\x66\x40"[:-1] + local_60 + input_user
10:  else
11:    if local_c ≠ 1 then
12:      end procedure
13:    end if
14:    local_68 = input_user + "\x2e\x72\x4d"[:-1]
15:  end if
16:  if local_68 = input_pass then
17:    Output "Cracked"
18:  else
19:    Output "Not Cracked"
20:  end if
21: end procedure
```

Algorithm 1 is the main logic of f-crackme. It inputs a username and password and stores them in `user` and `pass` respectively, and calls the

³See Appendix B for my high level representation of the code

⁴See Appendix A.1 for decompile dumps of the function

Algorithm 2 f-crackme’s checkUsername() procedure

```
1: procedure CHECKUSERNAME(username)
2:    $x \leftarrow \text{len}(\text{username})$ 
3:   if  $(x < 2) \vee (x > 7)$  then
4:     if  $(x < 8)$  then
5:       return 2
6:     end if
7:     return 1
8:   end if
9:   return 0
10: end procedure
```

checkUsername function (Algorithm 2), which can be expressed by the following piecewise function:

$$\delta(x) = \begin{cases} 2 & \text{if } [(x < 2) \vee (x > 7)] \wedge \neg(x < 8) \\ 1 & \text{if } [(x < 2) \vee (x > 7)] \wedge (x < 8) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where x is the length of the `user` string.

From lines 3 – 5 in Algorithm 2, and line 6 and line 10 of Algorithm 1, I can conclude that the I need to pick a `user` input that is not of length “8” because such an input will cause the programme to exit. Lines 3 – 5 will return a 2 if $[(x < 2) \vee (x > 7)] \wedge \neg(x < 8)$, lines 10 and 11 of Algorithm 1 instructs the programme to exit if `local_c` \neq 1. I would like to avoid the programme exiting, so I need to work out an acceptable input such that $\text{len}(\text{user}) \neq 8$.

2.2 “Eyeballing” the source code for clues

Through a combination of background knowledge of the technical aspects of reverse engineering, tinkering, knowledge of hacking and popular culture, and luck, I was able to work out from my (imperfect) Python reconstruction a keygen formula.

I learnt that the “password” consists of `user` with “@fsociety” appended to it. This snippet from A.1 led me to, somewhat by accident, discover this fact:

```
[ ... snip ... ]
```

```

if (local_c == 0) {
    local_68 = 0x746569636f736640;
    local_60 = 0x79;
    strcat(local_28, (char *)&local_68);
    strcpy((char *)&local_68, local_28);
}

else {
    if (local_c != 1) {
        /* WARNING: Subroutine does not return */
        exit(0);
    }

    local_68 = CONCAT44(local_68._4_4_, 0x2e724d);
    strcat((char *)&local_68, local_28);

}

[ ... snip ... ]

```

As one may be able to infer from this, f-crackme will exit if the `local_c` is not set to either 0 or 1. If the `local_c` is indeed set to 0, I observe that the value `local_68`, which is set to `0x746569636f736640`, is appended to `local_60`, which is set to `0x69`. It uses the `strcat` and `strcpy` functions to do so, and with a rudimentary programming background, I was able to infer that these variables were strings because `strcat` and `strcpy` deals with string operations.

I asked myself: “what is `local_60` set to?” I discovered that it is a string called `@fsociety` with the following method:

I represented the `0x746569636f736640` and `0x69` parts of the code as a string in Python3 shell as so:

```

>>> "\x74\x65\x69\x63\x6f\x73\x66\x40"
'teicosf@'
>>> "\x79"
'y'

```

It seemed like this was “@fsociety” spelt backwards. So then I tried:

```

>>> "\x74\x65\x69\x63\x6f\x73\x66\x40"[::-1] + "\x79"
'@fsociety'

```

This `@fsociety` string is appended to `local_28`, which is the input of the “username,” and ultimately stored in the `local_68` variable. But, the `local_c` needs to be set to 0 in order for this code to be executed (and to avoid an exit). How will I go about doing this?

2.3 The proper length for user

I noticed from the `if (local_c == 0)` part of the conditional⁵ that `checkUsername()` should return a value set to 0 in order for the `@fsociety` to be appended to `local_68`, and in order for the programme to execute properly. I can infer from Algorithm 2 and Equation 1 that in order to get a value of 0, I must find an x such that:

$$\neg[(x < 2) \vee (x > 7)] \equiv \{x \mid 2 \leq x \leq 7\}$$

where x is the length of the username.

This is an intuitive expression of the condition I am faced with and I do not intend to construct a rigorous proof for it. My (casual) justification for it is that a value of x that is < 2 or > 7 will return a 2. Likewise, a value where x is between 2 and 7 will return a 0 or a 1.

Knowing this, I can now proceed to analyse the other most crucial bit of the code.

2.4 Comparing the username and password inputs

From the following code:

```
printf("Type your password: ");
scanf("%30s",local_48);
local_10 = strcmp((char *)&local_68,local_48);

if (local_10 == 0) {
    printf("\nHello, friend. You successfully cr4cked me :)");
}
else {
    printf("\nI\'m sorry. You are not supposed to be here.");
}
```

[... snip ...]

⁵From Section 2.2

I can infer that the “password” will be stored in some variable called `local_48`, that the `strcmp` function will compare `local_48` (the “password”) to see if it’s identical to the `local_68`, and to print out the message “Hello, friend. You successfully cr4cked me :)” to notify the reverse engineer that they have indeed worked out the proper “username” and “password” combination.

I have learnt from Section 2.2 that the string `@fsociety` is appended to the username and stored in `local_68`. So, I can conclude that the “password,” or the `local_68`, is the username with a length between 3 and 7 with “fsociety” appended to it.

Knowing this, I can now proceed to build a keygen.

2.5 Writing the keygen

The keygen follows a simple procedure as outlined in Algorithm 3:

Algorithm 3 The core logic of the keygen (Simplified)

```
1: procedure MAIN
2:   user  $\leftarrow$  ""
3:    $x_f \leftarrow$  random number between 3 and 7.
4:   for 2 in range( $x_f + 1$ ) do
5:     user  $\leftarrow$  user + random lowercase letter
6:   end for
7:   pass  $\leftarrow$  user + "@fsociety"
8:   Print out user and pass
9: end procedure
```

So to write the keygen, I will start by importing necessary modules:

```
import string
import random
```

Then proceed by implementing the main logic. I used Python’s `String.join` method to build a string from a `list` of random lowercase letters in `string.ascii_lowercase`, drawn with the `random` module’s `choice` method. Finally, the generated username and passwords are printed out.

The implementation of the core logic is as follows:

```
def main():
    user = "".join([random.choice(list(string.ascii_lowercase)) \
                     for k in range(2, random.randint(4, 8))])
```

```
print("Username: " + user)
print("Password: " + user + "@fsociety\n")
```

And of course giving credit where credit is due ;-)

```
def message():
    print("Aleksey's Keygen for f-crackme!")
    print("Here's what I came up with :D\n")
```

3 Summary

The final keygen can be seen in Appendix C and the supplementary materials. In terms of selfish personal growth, I learnt a few things from this, like to consider the endianness of how a string may be stored (I was lucky because I got the Mr. Robot cultural reference).

This writeup can also serve as a case-study for anyone interested in software cracking, as opposed to merely for “selfish personal growth.” Hopefully it can be used by others as inspiration when it comes developing strategies in reverse code engineering.

3.1 Supplementary materials

GitHub repo: <https://github.com/Alekseyyy/SNHU/tree/main/sundries/wargames/crackmes.one/writeups/6194f35633c5d44c61906fe6>

Appendixes

A Decompiler results

These are the outputs of the Ghidra decompiler regarding interesting functions which can be used as a basis for constructing a keygen.

A.1 main.c

```
int main(int _Argc, char **_Argv, char **_Env)
{
    undefined8 uVar1;
    undefined8 local_68;
    undefined2 local_60;
    char local_48 [32];
```



```

char local_28 [24];
int local_10;
int local_c;

__main();
message();
printf("Type your username: ");
scanf("%20s",local_28);
uVar1 = checkUsername(local_28);
local_c = (int)uVar1;
if (local_c == 0) {
    local_68 = 0x746569636f736640;
    local_60 = 0x79;
    strcat(local_28,(char *)&local_68);
    strcpy((char *)&local_68,local_28);
}
else {
    if (local_c != 1) {
        /* WARNING: Subroutine does not return */
        exit(0);
    }
    local_68 = CONCAT44(local_68._4_4_,0x2e724d);
    strcat((char *)&local_68,local_28);
}
printf("Type your password: ");
scanf("%30s",local_48);
local_10 = strcmp((char *)&local_68,local_48);
if (local_10 == 0) {
    printf("\nHello, friend. You successfully cr4cked me :)");
}
else {
    printf("\nI\'m sorry. You are not supposed to be here.");
}
return 0;
}

```

A.2 checkUsername.c

```

undefined8 checkUsername(char *param_1)
{
    int iVar1;

```

```

size_t sVar2;
undefined8 uVar3;

sVar2 = strlen(param_1);
iVar1 = (int)sVar2;
if ((iVar1 < 2) || (7 < iVar1)) {
    if (iVar1 < 8) {
        uVar3 = 2;
    }
    else {
        uVar3 = 1;
    }
}
else {
    uVar3 = 0;
}
return uVar3;
}

```

B Python port of f-crackme

Please note that this script does not perfectly represent the original f-crackme binary. It was written just to help me work out the “username” and “password” combination, and said combination may not work with this script.

```

# High-level, albeit erroneous, representation of the
# f-crackme puzzle hosted on the crackmes.one website

# By A. S. "Aleksey" Ahmann <hackermaneia@riseup.net>
# - GitHub: github.com/Alekseyyy
# - Crackmes profile: crackmes.one/user/RelationalAlgebra

import sys

class f_crackme:

    def main(self):
        input_user = input("Type your username: ") #local_28
        input_pass = input("Type your password: ") #local_48
        local_c = self.checkUsername(input_user)

```

```

local_68 = None
local_60 = None

if local_c == 0:
    local_60 = "\x79"
    local_68 = "\x74\x65\x69\x63\x6f\x73\x66\x40"[:-1] + \
        local_60 + input_user
else:
    if local_c != 1:
        sys.exit(0)
    local_68 = input_user + "\x2e\x72\x4d"[:-1]

if local_68 == input_pass:
    print("Cracked")
else:
    print("Not Cracked")

def checkUsername(self, x):
    x_len = len(x)

    if (x_len < 2) or (x_len > 7):
        if (x_len < 8):
            return 2
        return 1
    return 0

if __name__ == "__main__":
    main_logic = f_crackme()
    main_logic.main()

```

C Keygen

This is the final keygen script that I came up for this puzzle:

```

# Keygen script for f-crackme (crackmes.one puzzle)

# By A. S. "Aleksey" Ahmann <hackermaneia@riseup.net>
# - GitHub: github.com/Alekseyyy
# - Crackmes profile: crackmes.one/user/RelationalAlgebra

```

```
import string
import random

def message():
    print("Aleksey's Keygen for f-crackme!")
    print("Here's what I came up with :D\n")

def main():
    message()
    user = "".join([random.choice(list(string.ascii_lowercase)) \
        for k in range(2, random.randint(4, 9))])
    print("Username: " + user)
    print("Password: " + user + "@fsociety\n")

if __name__ == "__main__":
    main()
```