Tool Developer Qualification Course

Reverse Engineering project

Bomb

Joshua Abraham

Taught by by
Dr. Joseph Santmeyer

November 17, 2017

Contents

1 Summary **2**

2 Stage 1 Analysis **3**

3 Stage 2 Analysis **4**

4 Stage 3 Analysis **5**

5 Works Cited **7**

1 Summary

The Fibonacci project has been a challenging experience. Lack of high-level loops, conditionals, etc. made implementing a program that calculates the Fibonacci sequence hard. Having to calculate up to the 500^{th} number (an absurdly large value) made it even harder. However, this project was a resounding success. All requirements were accomplished, with minimal C function calls. The program is fast, easy to read, and well documented. I learned a great deal during development; one of the most practical lessons was being able to translate high-level C into assembly almost line by line. If I had more time, I would spend it on removing my call to *printf* entirely and use 100% assembly.

1. Stage 1: swordfish
2. Stage 2: jabraham
3. Stage 3: 1872
4. Stage 4: 107 214 428 856 1713
5. Stage 5: 1172
6. Stage 6: 48 a 48
7. Stage 7: (press enter)
8. Stage 8: (run ./stag8.sh in the same directory as the binary, then press enter).

2 Stage 1 Analysis

Stage 1 is straightforward. Below is an abbreviated listing of the relevant disassembled code:

```
mov     esi, 0x4018f8    ; "swordfish"
mov     rdi, rax         ; user input
call    strcmp
test    eax, eax
setz    al
leave
```

This stage calls strcmp with user input and the string "swordfish". If the return value is zero (meaning the strings are identical), this stage returns non-zero and the program proceeds.

3 Stage 2 Analysis

Stage 2 starts with a call to getenv. This function takes one parameter and returns the value of a specified environment variable.

```
mov     edi, 0x401902     ; "USER"
call    getenv
mov     QWORD PTR [rip+0x201264], rax
```

The user name is stored in a global variable that is used throughout the binary at rip+0x201264. In my case the user name is "jabraham". This and the saved user input are passed to strcmp, similarly to Stage 1.

```
mov     rax, [rbp-0x8]
mov     rsi, rdx          ; "jabraham"
mov     rdi, rax          ; "jabraham"
call    strcmp
test    eax, eax
setz    al
leave
```

After the call to strcmp, the program checks the return value in eax and sets al if the zero flag is set. The stage is passed if the user supplies their own username.

4 Stage 3 Analysis

Analysis:

Stage 3 takes user input and saves it in [rbp-0x28]. The global variable holding the username is moved into [rbp-0x8].

```
push    rbp
mov     rbp, rsp
sub     rsp, 30h
mov     [rbp-0x28], rdi
mov     rax, fs:28h
mov     [rbp-0x08], rax
xor     eax, eax
mov     rcx, cs:src
mov     rax, [rbp-0x28]
```

Once these variables are set, they are passed to the C function strncat. It takes three parameters: the destination, the source, and the maximum number of bytes to concatenate.

```
mov     edx, 0x80       ; number
mov     rsi, rcx        ; source (username)
mov     rdi, rax        ; destination (input)
call    strncat
```

The destination will now look similar to this: "100jabraham". The bomb then calls strlen on the original username variable.

```
mov     rax, [rbp+0x28]
mov     rdi, rax        ; username ("jabraham")
call    strlen
```

The length of the username is saved on the stack at [rbp-0x10]. Then the bomb gets ready for a call to sscanf. This function scans a string using a format string and stores each match in following parameters. Here sscanf is passed the concatenated string, the format specifier "%u", and the address of a stack variable. The "%u" tells us that sscanf is looking for a number, meaning the password for this stage is a single number.

```
mov     [rbp-0x10], rax
mov     [rbp-0x14], 0
lea     rdx, [rbp-0x14]
mov     rax, [rbp-0x28]
mov     esi, 0x401907  ; "%u"
mov     rdi, rax
mov     eax, 0
call    sscanf
```

Next the binary performs series of calculations with the total length and input number. Note: for readability, I have changed the offsets to reflect each stack variable's purpose.

```
mov     eax, [rbp+input_as_a_number]
mov     eax, eax
mov     edx, 0
div     [rbp+input_len]
mov     [rbp+input_as_a_number], eax
mov     eax, [rbp+input_as_a_number]
shr     eax, 2
mov     [rbp+input_as_a_number], eax
mov     eax, [rbp+input_as_a_number]
mov     edx, 0AAAAAAABh
mul     edx
mov     eax, edx
shr     eax, 1
cmp     rax, [rbp+input_len]
setnbe  al
```

Let us decode what is happening in this block of code. The input number is moved to eax. Then it is divided by the total length of the concatenated string. The result is stored in the input number variable, and a logical shift right 2 is performed on it. This value is then multiplied by the value 0xAAAAAAAB and shifted right 1. The calculated value is compared to the original input length and al is set to one if it is greater. To summarize the previous in a more readable format, in order to pass this stage your input must satisfy the following:

$$\frac{input\_number \div input\_len}{12} > input\_len$$

If so, the function returns non zero and the bomb continues to stage 6.

Solve Script:

In order to solve the stage I wrote a script in Python 3 to quickly perform the math needed. The script is called "stage3.py" and is located in this repository for your convenience.

```
from os import getenv
username = getenv("USER")       # Calculate the length of the username
usernamelen = len(username)
ans = 0                         # Start at zero

# Loop until the first number is greater than
# the length of the number plus the username length

while True:
    ans += 1
    length = len(str(ans)) + usernamelen
    if (ans // length // 12) > length:
        break

print(ans)
```

On the UMBC server, this code gives me the integer value 1872.

5 Works Cited

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[2] Albert Einstein. *Zur Elektrodynamik bewegter Körper*. (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891921, 1905.

[3] Knuth: Computers and Typesetting,
`http://www-cs-faculty.stanford.edu/~uno/abcde.html`