Name: Stephen Paradis
Date: May 27 2016
Current Module: Intel Assembly using C
Project Name: Bomb

Bomb Program

Stage 1:

For this stage I began with calling strings on the “bombs” executable. From this I found after the plain text stage[#]: I found the plaintext “swordfish” so I ran the program to test against that string and found it to be the correct password. When I looked at it through objdump I found the <stage_1> label and began looking through it. I saw the strcmp call so I realized that strcmp was comparing what I entered against the variable “swordfish”.

Stage 2:

For stage 2 I checked strings again and saw the word USER so I tried that first, when it did not work I went into objdump and found the <stage_2> label. Inside the label I saw a <getenv> call at 0x400eb8. Underneath that I also saw the comment <username> so I tested my username which worked.

Stage 3:

For stage 3 I checked strings which gave me no real value so I went into objdump and found the <stage_3> label. As I ran through it I found a strlen and strncat call. Based on a check my Wilding I tried ‘8675309’ which worked. Through later testing I found that the lower limit of the number entered could change based on the length of the username which was 8 characters for me which led to the lowest number i can enter being 1872.

Later with extensively talking with the rest of the class SSG Torres found out that the user input is concatenated with the username. Strlen then is is called on the concatenated string, the user input number is then divided by that number. The result of that is divided by 4 and the result of that is multiplied by 0xaaaaaaab. The returning value of that must overflow the data register, then that divided by 2 must be greater than the length of the concatenated string.

Stage 4:

For this stage I moved onto the <stage_4> label and went straight into gdb. Through extensive testing Follenbee and I found that the strcmp is comparing against “%u %u %u %u %u” however it kept failing before getting to the math portion. Through further testing of gdb I found these specific lines that sprung out at me:

- 0x400fb0 we grabbed the username
- 0x400fba it grabbed out the first character of my username and moved it into eax
- 0x40100c compares to make sure that five things were matched to sscanf

After sscanf is matched we start looping through first checking if the first number is greater than the ascii decimal value of the first letter of your username. If it is the fail check does not get hit and we continue by adding based on these lines:

- 0x401033 which moves the next input value into eax
- 0x401037 which adds the decimal value of the first ascii character to the current input value you are on in the loop so 1-5

So for me I entered 116 310 620 1240 5047, based on the code it would add 115(ascii value for s) to 116 and stored that into the variable at [rbp-0x2c] and then eax moves onto the next argument. If all checks don't lead to failure then we move onto the final part which either leads to success in the end or the bomb exploding. On the line:

- 0x40104f the final value of all input is stored into the counter register
- 0x401052-0x40106f arithmetic operations which if the correct numbers were entered will set eax to 0

The arithmetic operations only really check if the decimal value of the ascii character plus the sum of the 5 input values are divisible by 7 and if it is it returns successful.

Stage 5:

For this stage I first went to strings and saw the string "%c %w %c", from there I went to the <stage_5> label in objdump and began reading through. It was slightly helpful but it did not tell or show me what was happening so I moved into gdb. For the first attempt I put in "s 2080 w" and began watching as gdb stepped through. My assumption that the format had to be in "%c %w %c" format was correct because sscanf returned 3 which means it matched 3 things. I then watched it move my username into rax, and then rdi before calling strlen to get the length of my username. It the jumps to this line:

- 0x40121b which moves the counter variable into rax
- 0x40121f this compares this value to the length of my username(found from strlen)

This showed me that this was just looping through and for each letter in the username the counter variable would be incremented. It the jumps again to this line:

- 0x401103 moves the username into the data register
- 0x40110a moves the current number count into rax
- 0x40110e the it adds the username + the count variable to get the next character in the username

The above lines are what moves from the current character in the username to the next character in the username. It then moves the value of the current character the loop is on into the accumulator and makes a check to see if it is lower case ascii. From there is will jump to whichever case is found based on this line:

- 0x401125 It moves into rax the memory address of the switch statement to go to

From here it goes to whichever switch statement applies and based on my username are the cases that apply below:

- Case a:
  - Add username[position] and first %c argument then subtract that from the %d given
- Case b-d:

  - Subtract username[position] from %d given twice
- Case i:
  - Add username[position] and first %c argument then subtract that from the %d given
- Case o-t:
  - Add username[position] and second %c argument then subtract from the %d given

Finally once the loop has gone through each character in the username it tests the accumulator against itself to see if the %d given is 0, if it is the stage will be defused.

Stage 6: