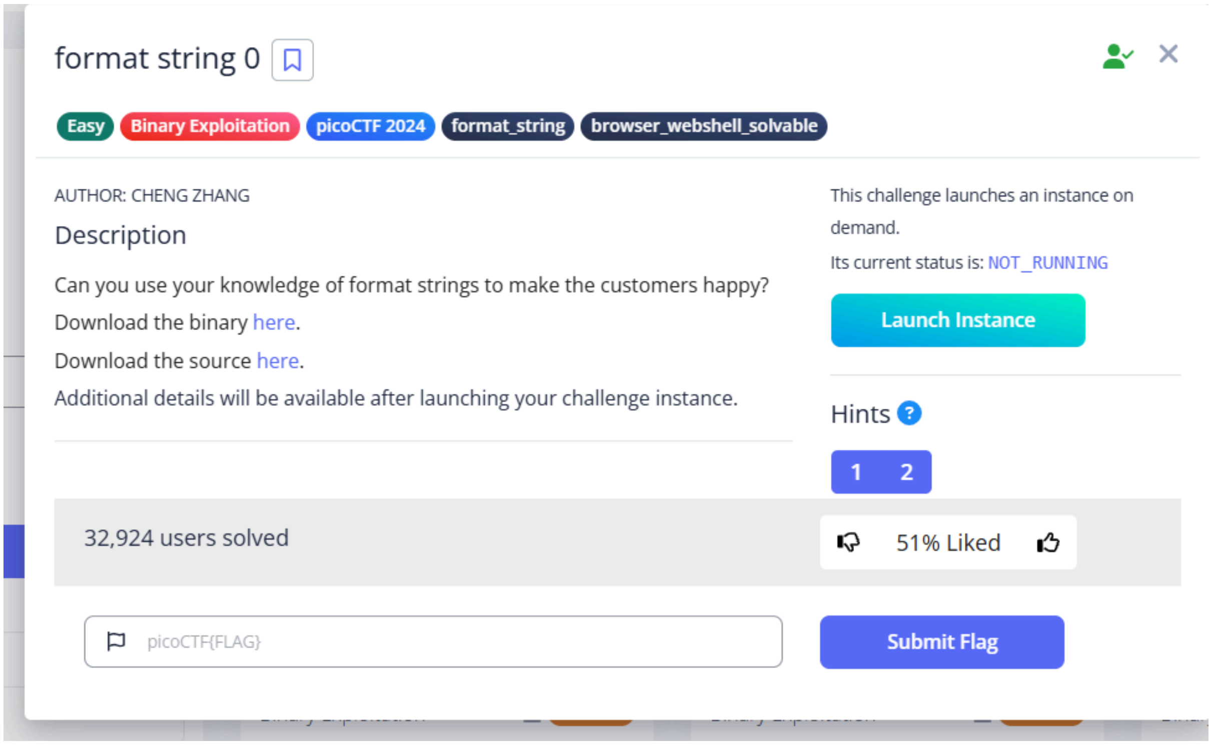
JASMINE C. OMANDAM
[picoCTF - picoGym Challenges]



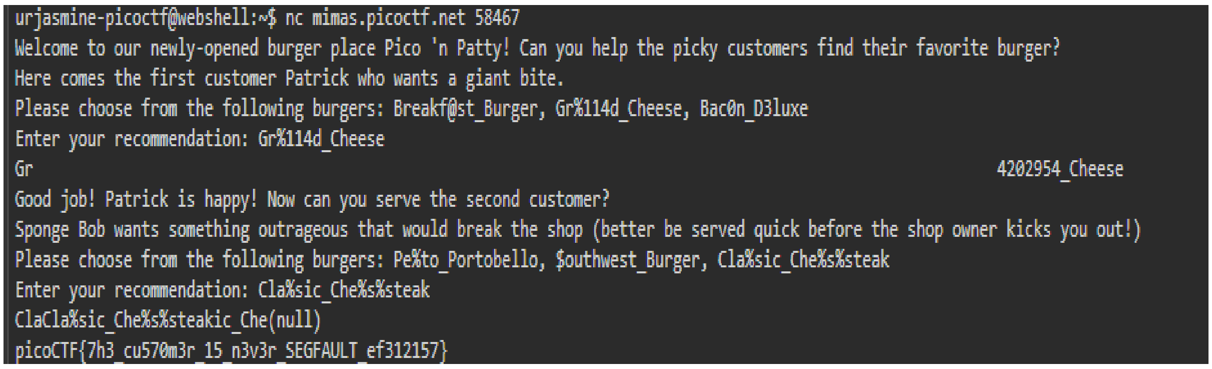
AUTHOR: CHENG ZHANG

Description

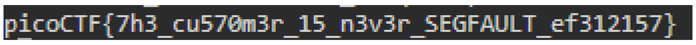
Can you use your knowledge of format strings to make the customers happy?
Download the binary here.
Download the source here.
Additional details will be available after launching your challenge instance.

This challenge launches an instance on demand.
Its current status is: NOT_RUNNING

32,924 users solved



```
urjasmine-picoctf@webshell:~$ nc mimas.picoctf.net 58467
Welcome to our newly-opened burger place Pico 'n Patty! Can you help the picky customers find their favorite burger?
Here comes the first customer Patrick who wants a giant bite.
Please choose from the following burgers: Breakf@st_Burger, Gr%114d_Cheese, Bac0n_D3luxe
Enter your recommendation: Gr%114d_Cheese
Gr                                                                                                           4202954_Cheese
Good job! Patrick is happy! Now can you serve the second customer?
Sponge Bob wants something outrageous that would break the shop (better be served quick before the shop owner kicks you out!)
Please choose from the following burgers: Pe%to_Portobello, $outhwest_Burger, Cla%sic_Che%s%steak
Enter your recommendation: Cla%sic_Che%s%steak
ClaCla%sic_Che%s%steakic_Che(null)
picoCTF{7h3_cu570m3r_15_n3v3r_SEGFAULT_ef312157}
```



```
picoCTF{7h3_cu570m3r_15_n3v3r_SEGFAULT_ef312157}
```

Write-Up: Format String 0

When I first connected to the server using nc mimas.picoctf.net 58467, I honestly thought it would just be a simple input challenge. The burger shop theme looked playful, but I quickly realized something unusual was happening with the inputs.

For the first customer, Patrick, I selected **Gr%114d_Cheese** exactly as written. Instead of printing the burger name normally, the output became distorted:

```
Gr 4202954_Cheese
```

That was my first clue. The %114d was clearly being interpreted as a **format specifier**, not plain text. It was behaving like a printf vulnerability. That's when I understood this was a **format string exploitation challenge**.

The second customer's hint said SpongeBob wanted something outrageous that would "break the shop." That phrase immediately made me think of causing a crash — specifically a **segmentation fault**. I initially tried manipulating inputs like Pe%to_Portobello%s, but the program rejected them as invalid burgers.

Then I looked more carefully at the provided options. One of them was:

```
Cla%sic_Che%s%steak
```

This burger name itself contained multiple %s format specifiers. That was suspicious. Instead of modifying it, I entered it exactly as written.

The output became corrupted:

```
ClaCla%sic_Che%s%steakic_Che(null)
```

And then suddenly the flag appeared:

picoCTF{7h3_cu570m3r_15_n3v3r_SEGFAULT_ef312157}

At that moment, I understood the vulnerability clearly. The program was likely using something like:

printf(user_input);

instead of:

printf("%s", user_input);

Because of that, the %s specifiers inside the burger name were interpreted by printf, causing it to read unintended memory addresses. This memory mismanagement triggered abnormal behavior, effectively "breaking the shop" and revealing the flag.

Challenges I Faced

- At first, I didn’t immediately recognize it as a format string vulnerability.
- I tried manually injecting %s into other burger names, but the program validated inputs strictly.
- It took careful observation of the first distorted output to understand that the provided menu itself contained the exploit.