

## JASMINE C. OMANDAM

### [picoCTF - picoGym Challenges](#)

56 513 solves

93%

1 114 solves

80%

1 541 solves

### Binary Gauntlet 0

Medium Binary Exploitation picoCTF 2021

AUTHOR: MADSTACKS

#### Description

This series of problems has to do with binary protections and how they affect exploiting a very simple program. How far can you make it in the gauntlet? The flag for this challenge does not include the standard picoCTF{} wrapper. Additional details will be available after launching your challenge instance.

This challenge launches an instance on demand.  
Its current status is: **NOT\_RUNNING**

[Launch Instance](#)

#### Hints

(None)

1,095 users solved

17% Liked

[Submit Flag](#)

```
picoCTF Webshell Help [Icons] X
^C
urjas_mine-picoctf@webshell:~$ nc -v wily-courier.picoctf.net 52320
Connection to wily-courier.picoctf.net (18.189.99.27) 52320 port [tcp/*] succeeded!
a
a

urjas_mine-picoctf@webshell:~$
urjas_mine-picoctf@webshell:~$ nc -v wily-courier.picoctf.net 52320
Connection to wily-courier.picoctf.net (18.189.99.27) 52320 port [tcp/*] succeeded!
a
a
aa
aaa
urjas_mine-picoctf@webshell:~$ nc -v wily-courier.picoctf.net 52320
Connection to wily-courier.picoctf.net (18.189.99.27) 52320 port [tcp/*] succeeded!
a
a
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
dc42ef4afcf1c2d58c67c90679ac8041

urjas_mine-picoctf@webshell:~$ ^C
urjas_mine-picoctf@webshell:~$
```

## Write-Up: BINARY GAUNTLET 0

When I first opened the challenge, I saw that it was a binary exploitation problem with a simple program and a connection string to a remote server. My instinct was to peek inside the binary with Ghidra, and sure enough, the `main` function revealed two interesting things: it read the flag into memory, and it set up a custom segmentation fault handler that would print the flag if the program crashed. That was the key. I didn't need to build a complicated exploit chain, I just needed to make the program crash.

Looking closer, I noticed two user inputs. The first one was printed with `printf(local_10)`, which hinted at a format string vulnerability, but the second input was even more promising. It was copied into a buffer of size 108 using `strcpy`, which doesn't check bounds. That meant if I sent more than 108 characters, the buffer would overflow, the program would crash, and the signal handler would dutifully print the flag.

With that plan in mind, I connected to the challenge server using netcat:

```
nc -v wily-courier.picoctf.net 52320
```

The connection succeeded, and the program waited for input. For the first prompt, I just typed a single `a` and hit Enter. Then came the second prompt — this was the moment to trigger the crash. I pasted in a long string of `as`, well over 108 characters, and pressed Enter. The program immediately crashed, and just as expected, the signal handler printed out a hexadecimal string. That string was the flag.

In my run, the flag was:

**dc42ef4afc1d2d58c67c99679ac8041**

I copied it and submitted it, and the challenge was solved.

## Reflection

What made this challenge fun was how straightforward the vulnerability was once you spotted the signal handler. Instead of trying to hijack execution or build a ROP chain, the exploit was simply to crash the program. It's a reminder that in CTFs, sometimes the simplest path is the intended one.