Solves

Proxed (WEB):

- opening up the link leads to this output:
  -
```
untrusted IP: 10.152.0.21
```
- so we look at the source files and there's only one file with any substance "main.go".
  - "main.go" has 2 points of interest:

```
if ip != "31.33.33.7" {
    message := fmt.Sprintf("untrusted IP: %s", ip)
    http.Error(w, message, http.StatusForbidden)
    return
} else {
    w.Write([]byte(os.Getenv("FLAG")))
}
```

    - so you have to spoof your IP somehow to 31.33.33.7 in the GET request but in what way?
    - Well after googling some more of the random terms that show up in the code this one returns an interesting result:

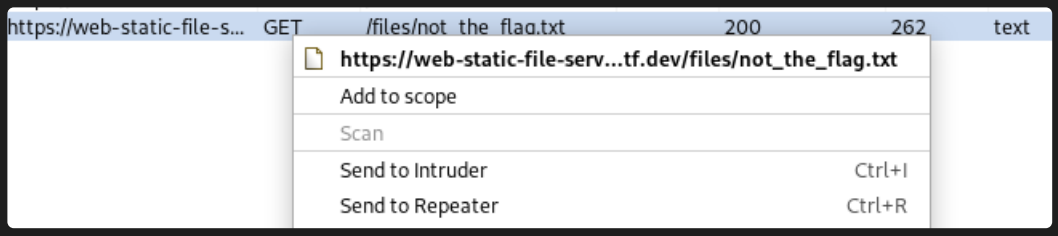
```
http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
    xff := r.Header.Values("X-Forwarded-For")
```

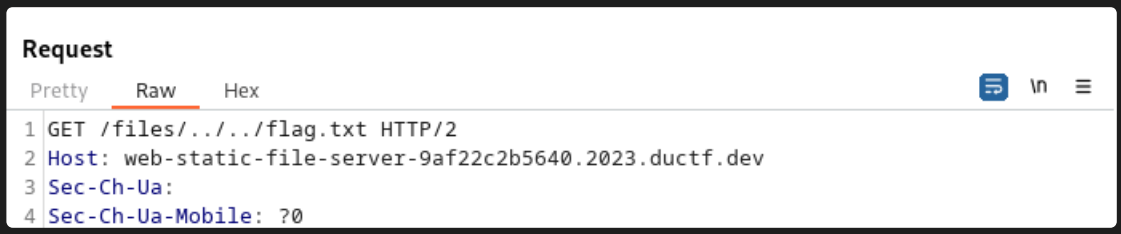
    - this is a header used to determine the IP address of someone connecting and is what you need to fool
- Spoofing IP address's is actually kind of a pain in the butt and I've never done it before so I double-checked if Burpsuite can do it for singular requests and lo-and-behold:
  - https://portswigger.net/burp/documentation/desktop/tutorials/using-match-and-replace
- I followed the guide (step 3 onwards) and you're rewarded with the flag:
- DUCTF{17_533m5_w3_f0rg07_70_pr0x}

Static File Server(Web):

- we can see due to the Dockerfile that the flag file is not in any subdirectory and is actually in the root directory, and the `not_the_flag.txt` file wasn't lying:
  -

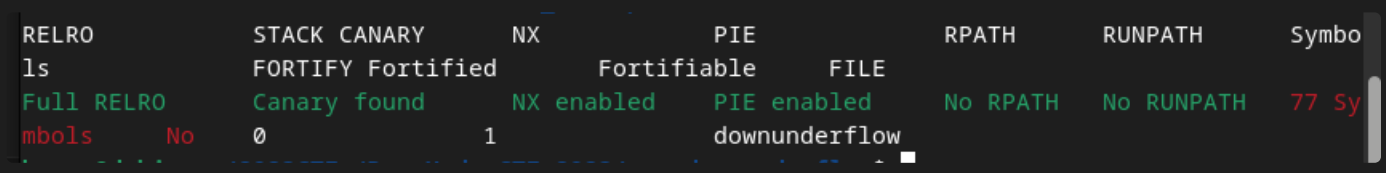
```
Dockerfile > ...
1  FROM python:3.10
2
3  WORKDIR /app
4  COPY app.py .
5  COPY flag.txt /flag.txt
6  COPY files/ files/
7
8  RUN pip3 install aiohttp
9
10 RUN /usr/sbin/useradd --no-create-home -u 1000 ctf
11 USER ctf
12
13 CMD ["python3", "app.py"]
14
```

    - we can also see there's WORKDIRectory called "/app" which means files like `not_the_flag.txt` are actually inside this path:
        - `root/app/files/not_the_flag.txt`
    - `app.py` was moved to the work directory due to the `COPY` command keeping it from being moved to `.` (working directory) or any other subdirectory
- from looking at the documentation for aiohttp here we can see how this python module manages a static file server:
    - https://docs.aiohttp.org/en/stable/web_advanced.html#static-file-handling
    - we can't be in the root or `app/` directory, each `GET` request can only start with `/files` for this local-file-inclusion attack
- if you try `https://web-static-file-server-9af22c2b5640.2023.ductf.dev/files/../flag.txt` the periods will get filtered out by your browser so I'll use Burpsuite again.
- go into burpsuite, open a browser, paste in the challenge URL and click on the "not the flag" hyperlink
    - right-click on the target, and send to repeater:
        - 

    - go into the "Repeater" tab and edit the `GET` request directly or in the "Inspector" side-panel into this
        - 

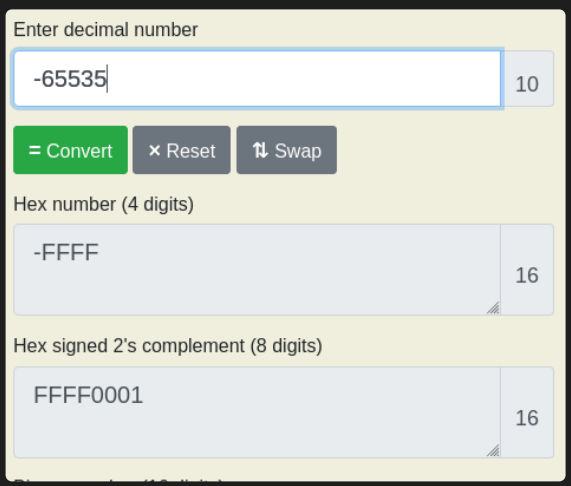
    - one `../` to get into the `app/` directory, another `../` to get back into the root directory
- Which gives us this:
    - DUCTF{../../../p4th/tr4v3rsal/as/a/s3rv1c3}

DownUnderFlow(PWN)

- as always with PWN challenges check what security the file has with `checksec` and this one actually has all of them:
    - 

- with all of these enabled it'll be hard to do a buffer overflow or inject Shellcode not to mention addresses will shuffle around a bit
- but this is a beginner challenge so let's look at it some more, here's the code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define USERNAME_LEN 6
#define NUM_USERS 8
char logins[NUM_USERS][USERNAME_LEN] = { "user0", "user1", "user2", "user3", "user4", "user5", "user6", "admin" };

void init() {
    setvbuf(stdout, 0, 2, 0);
    setvbuf(stdin, 0, 2, 0);
}

int read_int_lower_than(int bound) {
    int x;
    scanf("%d", &x);
    if(x >= bound) {
        puts("Invalid input!");
        exit(1);
    }
    return x;
}

int main() {
    init();

    printf("Select user to log in as: ");
    unsigned short idx = read_int_lower_than(NUM_USERS - 1);
    printf("Logging in as %s\n", logins[idx]);
    if(strncmp(logins[idx], "admin", 5) == 0) {
        puts("Welcome admin.");
        system("/bin/sh");
    } else {
        system("/bin/date");
    }
}
```

- one thing to keep an eye out for in PWN challenges is conversions between variable types
- what's weird here is how `read_int_lower_than()` returns an integer, but that gets converted to an unsigned short back in `main()`
  - large data type --> smaller data type is a bad idea so that's worth a google
  - in summary the int gets truncated once converted, and the leftover is used as the unsigned short
  - But we don't know how it's truncated for example an int is: `0×1234|ABCD` we don't know if the 1234 or the ABCD gets cut in this case
  - And it also needs to be a signed/negative input to pass the `if()` check so it's iffy how that conversion will work as well.
- So let's just try it with the knowledge 65535 is the max an unsigned short can be:
  - -65535 --> user1
    - if you check rapidtables.com:
    - 

    - so it looks like the left half gets gets cut

    - -65536 --> user0
    - -65530 --> user6
    - -65529 --> admin shell
- DUCTF{-65529==_7(mod_65536)}