back-to-the-past

March 1, 2025

```
int64_t time = time(nullptr)
printf("time : %ld\n", 0)
srand(time.d)
void* filehandle = fopen(arg2[1], "rb+")

if (filehandle != 0)
    while (true)
        int32_t input = _IO_getc(filehandle)

        if (input == 0xffffffff)
            break

        fseek(filehandle, -1, 1)
        int32_t rand = rand()
        int32_t rand2 = rand s/ 0x7f
        fputc((rand.b - ((rand2 << 7).b - rand2.b)) ^ input.b, filehandle)
```

These are the functions to create the random values

```
uint64_t srand(int32_t arg1)
    uint64_t result = zx.q(arg1 - 1)
    seed = result
    return result

uint64_t rand()
    uint64_t rax_1 = 0x5851f42d4c957f2d * seed + 1
    seed = rax_1
    return rax_1 u>> 0x21
```

For the time, they give this hint:

Using the provided binary and the encrypted file, find a way to retrieve the flag contained in "flag.enc". Note that the binary would have been run in May 2024. Note: The flag is in the format PWNME{…}

[13]:
```
import datetime
stime = "01/05/2024"
etime = "31/05/2024"
```

```
start = datetime.datetime.strptime(stime, "%d/%m/%Y").replace(tzinfo=datetime.
 ↪timezone.utc).timestamp()
end = datetime.datetime.strptime(etime, "%d/%m/%Y").replace(tzinfo=datetime.
 ↪timezone.utc).timestamp()
int(end - start + 1)
```

[13]: 2592001

1 Solving

We brute force all unix timestamps, which only amount to 2592001 possibilites.

Next, there is an char by char encryption in OTP style. We we know the start of the plaintext `PWNME{`, we can check if the ciphertext applied to the keys generated by a timestamp results into our target plaintext.

```
 gcc solve.c
 ./a.out
found 1715198477
PWNME{4baf3723f62a15f22e86d57130bc40c3}

done
```