Syngard rearranged folders and added upload writeup — Latest commit 62fa4e9 on Dec 29, 2017

..		
README.md	rearranged folders and added upload writeup	a year ago
top.py	adding 34c3 folder and beginning of first writeup	a year ago
top_secret	adding 34c3 folder and beginning of first writeup	a year ago
top_solve.py	adding 34c3 folder and beginning of first writeup	a year ago

README.md

top challenge writeup

Category: Crypto

Description:

> Encrypted flag
>
> Encryption program

Solution write-up

So the program is encrypting a given string with a key that is randomly generated. Each letter is individually XORed with the letter of the key in the same position. However, we can notice two important things :

```
cur_time = str(time.time()).encode('ASCII')
random.seed(cur_time)
```

The random function is seeded with the time at which the program is launched. Since the `random.random()` function genrerates pseudo-random number from that seed, it means we can generate the same key if we get a hold of the seed.

```
c = [m ^ k for (m,k ) in zip(msg + cur_time, key + [0x88]*len(cur_time))]
```

Moreover, not only is the seed encrypted alongside the flag, but it also is with a fixed string of only "0x88" chars. That means it can be easily retrieved. That is because the "XOR" operation can trivially be reversed. In fact, if a^b = c then a^c = b (or b^c = a, because the operation is also commutative).

Let's then start by recovering the time value that was used as a seed. To do that, the first thing we need to know is the actual format of the value. So we generate a new one to see what it looks like.

```
$ python
>>> import time
>>> t = str(time.time()).encode('ASCII')
>>> t
'1514581353.81'
>>> len(t)
13
```

As we can see, `time.time()` returns a 13-char long string, whith 10 digits, a dot and then 2 other digits. Since it represents the time in seconds since January 1st, 1970, the first few digits of the seed should be the same (because the challenge and the flag were created at best a few months back). Let's take the last 13 characters of the encrypted flag, which should be the seed, and XOR them with "0x88" chars to get back the original timestamp.

```
with open("top_secret",'rb') as msg:
    encr_time = str(msg.read())[-14:]
```

```
    c = [chr(ord(m) ^ k) for (m,k) in zip(encr_time,  [0x88]*13)]
    print ''.join(c)
```

```
$ python top_solve.py
19133.8728752
```

Something is not right. We don't have enough numbers before the dot and too much after it. Maybe our time format wasn't actually the right one. Since I don't know if the program was used with python2 or python3, I can suppose that the time.time() function behaves differently depending on the version. How does it look in python3 ?

```
$ python3
>>> import time
>>> t = str(time.time()).encode('ASCII')
>>> t
b'1514582787.0529299'
>>> len(t)
18
```

Okay that looks more like what we got. So in this format the timestamp is actually 18 characters long. Let's tweak the values in our previous function.

```
$ python top.py
1513719133.8728752
```

And that seems to be it. So we got the timestamp that was used as seed for the generation of the random key. We can now move on to the next phase : decrypting the flag.
To do that we need to generate the key with the same seed, and use it to XOR the encrypted message to recover the flag.

```
cur_time = '1513719133.8728752'.encode('ASCII')
random.seed(cur_time)

with open("top_secret",'rb') as msg:
    flag_plus_timestamp = msg.read()
    #I remove the last 18 characters that corresponds to the encrypted timestamp
    flag = flag_plus_timestamp[:-19]

    key = [random.randrange(256) for _ in flag]

    c = [chr(m ^ k) for (m,k) in zip(flag, key)]

    print("".join(c))
```

```
$ python3 top.py
Here is your flag: 34C3_otp_top_pto_pot_tpo_opt_wh0_car3s
```

And we got the flag o/

During my solve, I first used python2 in this step. However, it seems that, just like time.time(), the random.randrange() function behaves differently in python2 and python3. So it wasn't giving me the proper key when I tried to generate it.