INFORMATION AND SYSTEMS SECURITY SOCIETY

UTCTF 2020

Forensics : One True Problem

Value : 50 Pts

Description : Two of my friends were arguing about which CTF category is the best, but they encrypted it because they didn't want anyone to see. Lucky for us, they reused the same key; can you recover it?

Attachment :

Here are the ciphertexts :

213c234c2322282057730b32492e720b35732b2124553d354c22352224237f1826283d7b0651

3b3b463829225b3632630b542623767f39674431343b353435412223243b7f162028397a103e

Solutions :

Looking at the challenge name and i was thinking of «OTP» One Time Pad. Doing a bit of research about «OTP reuse key attack» on google and i find a writeup of a similiar challenge.

Source : http://dann.com.br/alexctf2k17-crypto100-many_time_secrets/

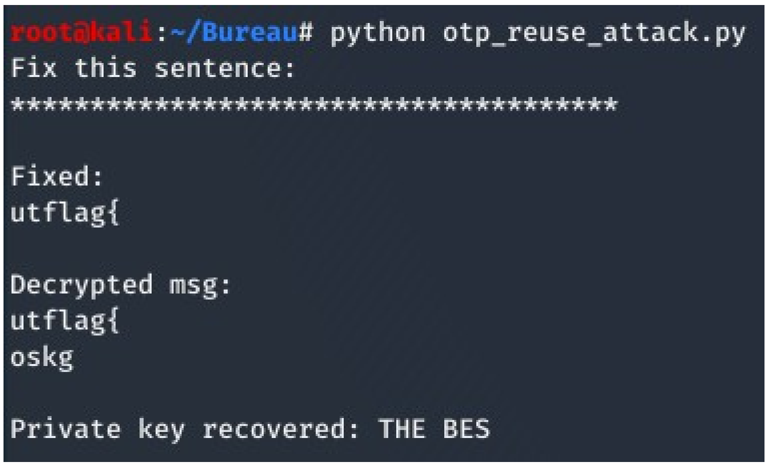
Taking the script and adapting it for our challenge and it will be like it.

```
#!/usr/bin/python
## OTP - Recovering the private key from a set of messages that were encrypted w/ the same private key
(Many time pad attack) - crypto100-many_time_secret @ alexctf 2017
# @author intrd - http://dann.com.br/
# Original code by jwomers: https://github.com/Jwomers/many-time-pad-attack/blob/master/attack.py)

import string
import collections
import sets, sys

# 11 unknown ciphertexts (in hex format), all encrpyted with the same key
c1 = "213c234c2322282057730b32492e720b35732b2124553d354c22352224237f1826283d7b0651"
c2 = "3b3b463829225b3632630b542623767f39674431343b353435412223243b7f162028397a103e"

ciphers = [c1, c2]
```

```
# The target ciphertext we want to crack
target_cipher =
"213c234c2322282057730b32492e720b35732b2124553d354c22352224237f1826283d7b0651"

# XORs two string
def strxor(a, b):     # xor two strings (trims the longer input)
    return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b)])

# To store the final key
final_key = [None]*150
# To store the positions we know are broken
known_key_positions = set()

# For each ciphertext
for current_index, ciphertext in enumerate(ciphers):
        counter = collections.Counter()
        # for each other ciphertext
        for index, ciphertext2 in enumerate(ciphers):
                if current_index != index: # don't xor a ciphertext with itself
                        for indexOfChar, char in enumerate(strxor(ciphertext.decode('hex'),
ciphertext2.decode('hex'))): # Xor the two ciphertexts
                                # If a character in the xored result is a alphanumeric character, it means there was
probably a space character in one of the plaintexts (we don't know which one)
                                if char in string.printable and char.isalpha(): counter[indexOfChar] += 1 # Increment the
counter at this index
        knownSpaceIndexes = []

        # Loop through all positions where a space character was possible in the current_index cipher
        for ind, val in counter.items():
                # If a space was found at least 7 times at this index out of the 9 possible XORS, then the space
character was likely from the current_index cipher!
                if val >= 7: knownSpaceIndexes.append(ind)
        #print knownSpaceIndexes # Shows all the positions where we now know the key!

        # Now Xor the current_index with spaces, and at the knownSpaceIndexes positions we get the key
back!
        xor_with_spaces = strxor(ciphertext.decode('hex'),' '*150)
        for index in knownSpaceIndexes:
                # Store the key's value at the correct position
                final_key[index] = xor_with_spaces[index].encode('hex')
                # Record that we known the key at this position
                known_key_positions.add(index)

# Construct a hex key from the currently known key, adding in '00' hex chars where we do not know (to
make a complete hex string)
final_key_hex = ''.join([val if val is not None else '00' for val in final_key])
# Xor the currently known key with the target cipher
output = strxor(target_cipher.decode('hex'),final_key_hex.decode('hex'))

print "Fix this sentence:"
print ''.join([char if index in known_key_positions else '*' for index, char in enumerate(output)])+"\n"

# WAIT.. MANUAL STEP HERE
# This output are printing a * if that character is not known yet
# fix the missing characters like this: "Let*M**k*ow if *o{*a" = "cure, Let Me know if you a"
# if is too hard, change the target_cipher to another one and try again
# and we have our key to fix the entire text!
```

```
#sys.exit(0) #comment and continue if u got a good key

target_plaintext = "utflag{"
print "Fixed:"
print target_plaintext+"\n"

key = strxor(target_cipher.decode('hex'),target_plaintext)

print "Decrypted msg:"
for cipher in ciphers:
	print strxor(cipher.decode('hex'),key)

print "\nPrivate key recovered: "+key+"\n"
```

Decoding our cipher using the «**target_plaintext**» «**utflag{**» and we got a part of the key.



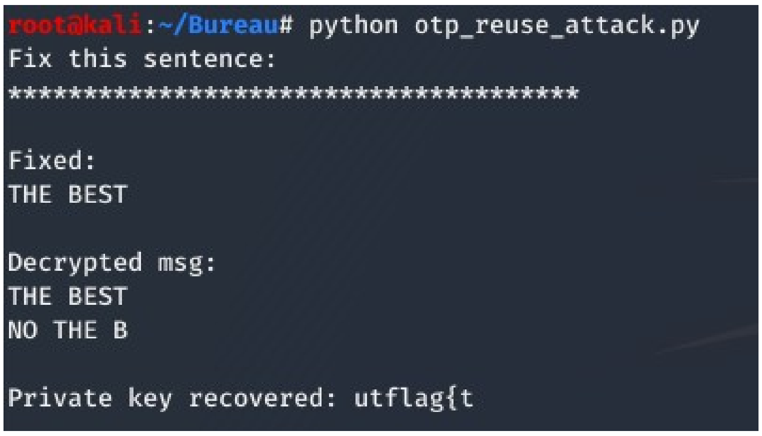
```
root@kali:~/Bureau# python otp_reuse_attack.py
Fix this sentence:
****************************************

Fixed:
utflag{

Decrypted msg:
utflag{
oskg

Private key recovered: THE BES
```

Private key recovered : THE BES

A bit of guessing and replace the target_plaintext variable of the script with «THE BEST».

```
target_plaintext = "THE BEST"
```

Runing the tool again and we decoded a bit of our flag.



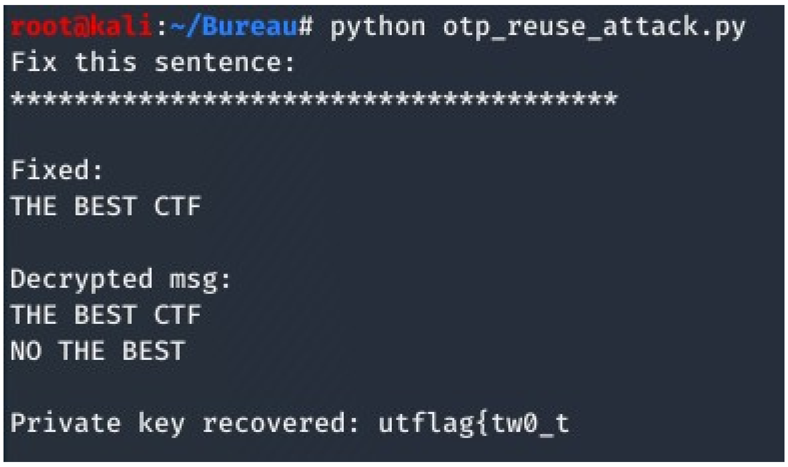
```
root@kali:~/Bureau# python otp_reuse_attack.py
Fix this sentence:
****************************************

Fixed:
THE BEST

Decrypted msg:
THE BEST
NO THE B

Private key recovered: utflag{t
```

Private key recovered : utflag{t

Guessing a bit more and i modified the target plaintext to «THE BEST CTF».

```
target_plaintext = "THE BEST CTF"
```

Running the tool and we decoded a bit more for our flag !



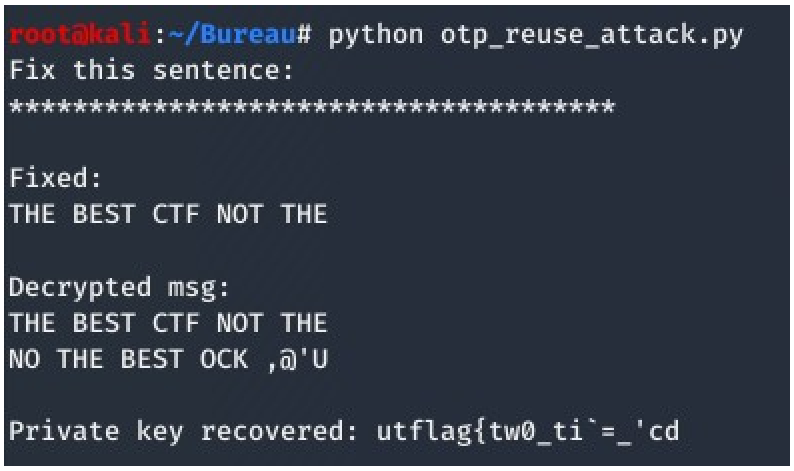
```
root@kali:~/Bureau# python otp_reuse_attack.py
Fix this sentence:
**************************************

Fixed:
THE BEST CTF

Decrypted msg:
THE BEST CTF
NO THE BEST

Private key recovered: utflag{tw0_t
```

Private key recovered : utflag{tw0_t

As we can see too, the decrypted msg give «**THE BEST CTF NO THE BEST**», so i guessed once again the target plaintext and modified it with «**THE BEST CTF NOT THE**».

```
target_plaintext = "THE BEST CTF NOT THE"
```

Runing the tool and we decoded a bit more, but corrupted.



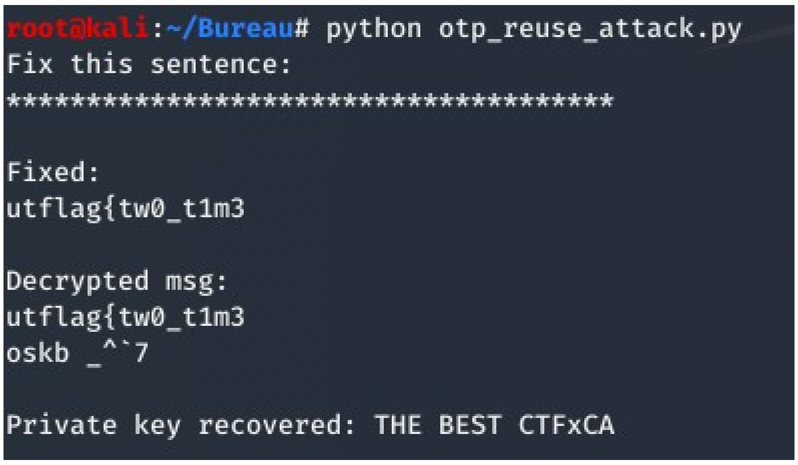
```
root@kali:~/Bureau# python otp_reuse_attack.py
Fix this sentence:
**************************************

Fixed:
THE BEST CTF NOT THE

Decrypted msg:
THE BEST CTF NOT THE
NO THE BEST OCK ,@'U

Private key recovered: utflag{tw0_ti`=_'cd
```

Private key recovered : utflag{tw0_ti

At this step i guessed so hard, and tryied to change the target plaintext to «**utflag{tw0_tim3**»

```
target_plaintext = "utflag{tw0_tim3"
```

Running the tool and it worked ! We decoded a good part of the key.



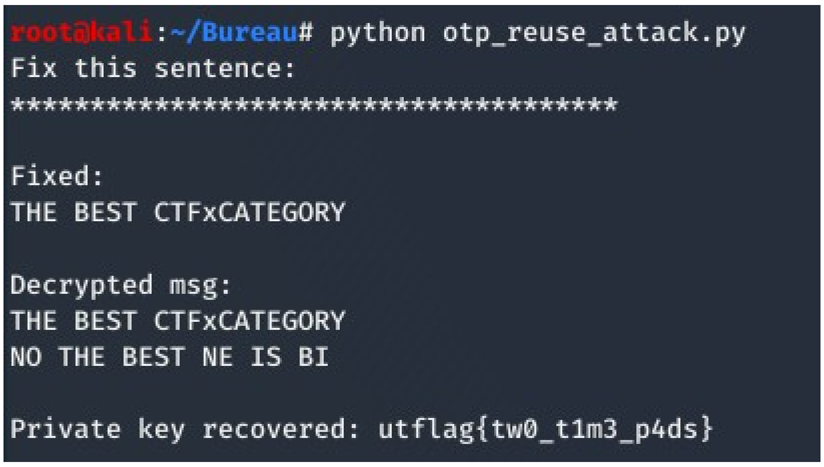
```
root@kali:~/Bureau# python otp_reuse_attack.py
Fix this sentence:
**************************************

Fixed:
utflag{tw0_t1m3

Decrypted msg:
utflag{tw0_t1m3
oskb _^`7

Private key recovered: THE BEST CTFxCA
```

Private key recovered : THE BEST CTFxCA

As this step, i looked to the descriptions, they said, they don’t know which CTF Category is the best. So i guessed one last time, modified the target plaintext to «THE BEST CTFxCATEGORY».

```
target_plaintext = "THE BEST CTFxCATEGORY"
```

Running the tool and we finnally got the flag ! I’m finally a l33t Gu3ss3r !



```
root@kali:~/Bureau# python otp_reuse_attack.py
Fix this sentence:
**************************************

Fixed:
THE BEST CTFxCATEGORY

Decrypted msg:
THE BEST CTFxCATEGORY
NO THE BEST NE IS BI

Private key recovered: utflag{tw0_t1m3_p4ds}
```

Flag : utflag{tw0_t1m3_p4ds}