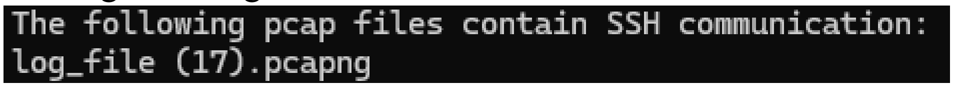
Solution writup for: IR_CTF by Yahel Koler:

1. Log file analysis:
   We need to scan the log files given to up for any that contain SSH communication(As stated by the document). For that I've written the following python script:

```
import os
from scapy.all import rdpcap, TCP

def is_ssh_packet(packet):
    """
    Check if a packet is an SSH packet.
    SSH typically runs on port 22.
    """
    if packet.haslayer(TCP):
        tcp_layer = packet.getlayer(TCP)
        if tcp_layer.dport == 22 or tcp_layer.sport == 22:
            return True
    return False

def scan_pcap_for_ssh(pcap_file):
    """
    Scan a pcap file for SSH communication.
    """
    packets = rdpcap(pcap_file)
    for packet in packets:
        if is_ssh_packet(packet):
            return True
    return False

def scan_folder_for_ssh(folder_path):
    """
    Scan all capture files in a folder for SSH communication.
    """
    ssh_pcap_files = []
    for filename in os.listdir(folder_path):
        if filename.endswith('.pcap') or filename.endswith('pcapng') or
filename.endswith('.cap'):
            pcap_file = os.path.join(folder_path, filename)
            if scan_pcap_for_ssh(pcap_file):
                ssh_pcap_files.append(filename)
    return ssh_pcap_files

folder_path = '.'
ssh_pcap_files = scan_folder_for_ssh(folder_path)

if ssh_pcap_files:
    print("The following pcap files contain SSH communication:")
    for pcap_file in ssh_pcap_files:
        print(pcap_file)
else:
    print("No pcap files with SSH communication found.")
```

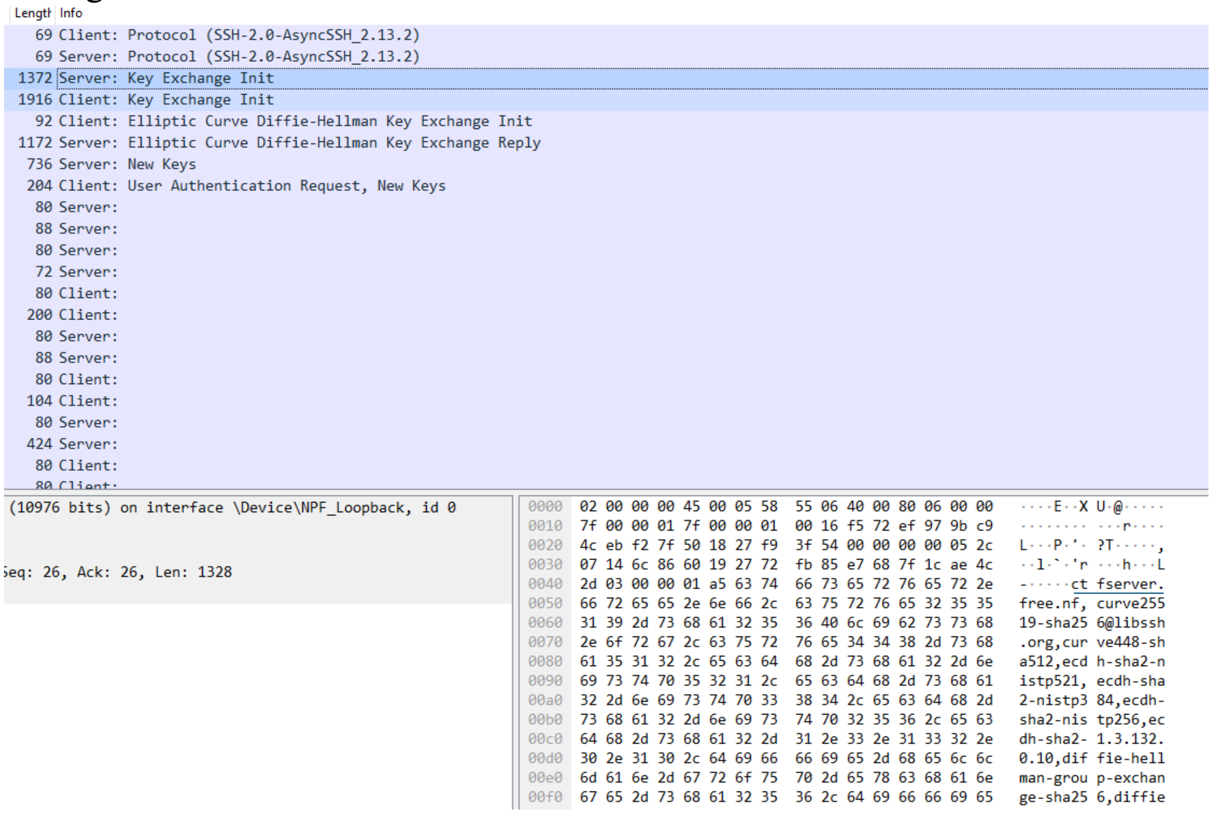
   This script runs for every capture file in the current directory and looks to see if there is SSH communication in it.

   running that we get:



```
The following pcap files contain SSH communication:
log_file (17).pcapng
```

opening up logfile(17) we see a simple ssh handshake, but scanning through it reveals two anomalies:

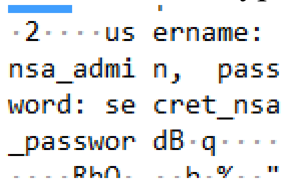


```
Length Info
  69 Client: Protocol (SSH-2.0-AsyncSSH_2.13.2)
  69 Server: Protocol (SSH-2.0-AsyncSSH_2.13.2)
1372 Server: Key Exchange Init
1916 Client: Key Exchange Init
  92 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
1172 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply
 736 Server: New Keys
 204 Client: User Authentication Request, New Keys
  80 Server:
  88 Server:
  80 Server:
  72 Server:
  80 Client:
 200 Client:
  80 Server:
  88 Server:
  80 Client:
 104 Client:
  80 Server:
 424 Server:
  80 Client:
  80 Client:
```

```
(10976 bits) on interface \Device\NPF_Loopback, id 0

Seq: 26, Ack: 26, Len: 1328
```

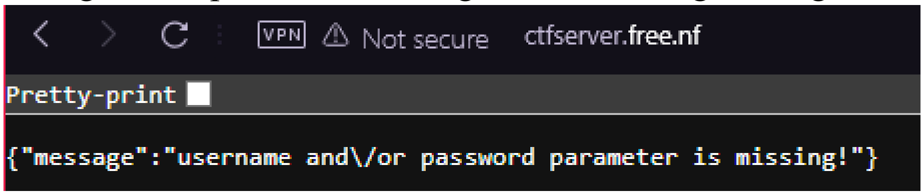
```
0000  02 00 00 00 45 00 05 58  55 06 40 00 80 06 00 00   ····E··X U·@·····
0010  7f 00 00 01 7f 00 00 01  00 16 f5 72 ef 97 9b c9   ········ ···r····
0020  4c eb f2 7f 50 18 27 f9  3f 54 00 00 00 00 05 2c   L···P·'· ?T·····,
0030  07 14 6c 86 60 19 27 72  fb 85 e7 68 7f 1c ae 4c   ··l·`·'r ···h···L
0040  2d 03 00 00 01 a5 63 74  66 73 65 72 76 65 72 2e   -·····ct fserver.
0050  66 72 65 65 2e 6e 66 2c  63 75 72 76 65 32 35 35   free.nf, curve255
0060  31 39 2d 73 68 61 32 35  36 40 6c 69 62 73 73 68   19-sha25 6@libssh
0070  2e 6f 72 67 2c 63 75 72  76 65 34 34 38 2d 73 68   .org,cur ve448-sh
0080  61 35 31 32 2c 65 63 64  68 2d 73 68 61 32 2d 6e   a512,ecd h-sha2-n
0090  69 73 74 70 35 32 31 2c  65 63 64 68 2d 73 68 61   istp521, ecdh-sha
00a0  32 2d 6e 69 73 74 70 33  38 34 2c 65 63 64 68 2d   2-nistp3 84,ecdh-
00b0  73 68 61 32 2d 6e 69 73  74 70 32 35 36 2c 65 63   sha2-nis tp256,ec
00c0  64 68 2d 73 68 61 32 2d  31 2e 33 2e 31 33 32 2e   dh-sha2- 1.3.132.
00d0  30 2e 31 30 2c 64 69 66  66 69 65 2d 68 65 6c 6c   0.10,dif fie-hell
00e0  6d 61 6e 2d 67 72 6f 75  70 2d 65 78 63 68 61 6e   man-grou p-exchan
00f0  67 65 2d 73 68 61 32 35  36 2c 64 69 66 66 69 65   ge-sha25 6,diffie
```

there is a simple suspicious link in the beginning of the key exchange and there is an unencrypted User Authentication Request holding credentials:



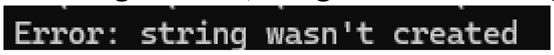
```
·2····us ername:
nsa_admi n,  pass
word: se cret_nsa
_passwor dB·q····
····RhQ· ··h·%··"
```

2. Server:
   Going to the specified link we get the following message:



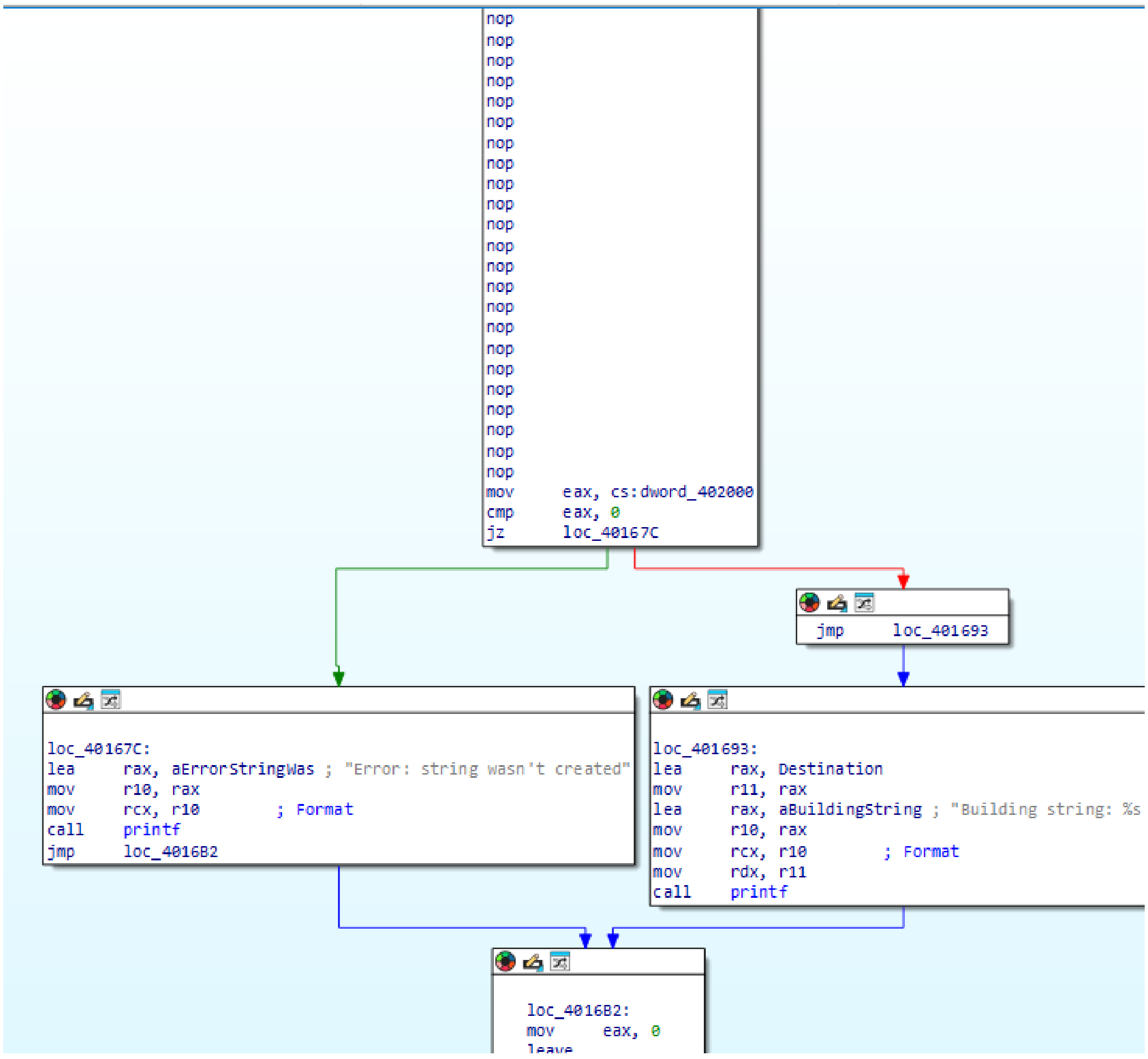
```
ctfserver.free.nf
Pretty-print
{"message":"username and\/or password parameter is missing!"}
```

   Well, using the specified credentials we found in the ssh to enter into this site as follows:
   ctfserver.free.nf/?username=nsa_admin&password=secret_nsa_password
   the server downloads an exe file for us:
3. Exe patching:
   Running the exe, we get the following error:



```
Error: string wasn't created
```

   We open the exe in IDA to find:



```
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
mov     eax, cs:dword_402000
cmp     eax, 0
jz      loc_40167C
```

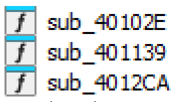
```
jmp     loc_401693
```

```
loc_40167C:
lea     rax, aErrorStringWas ; "Error: string wasn't created"
mov     r10, rax
mov     rcx, r10        ; Format
call    printf
jmp     loc_4016B2
```

```
loc_401693:
lea     rax, Destination
mov     r11, rax
lea     rax, aBuildingString ; "Building string: %s
mov     r10, rax
mov     rcx, r10        ; Format
mov     rdx, r11
call    printf
```

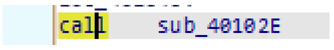
```
loc_4016B2:
mov     eax, 0
leave
```

We understand that we need to put something in the nops that will put a value into dword_402000, we also see that apart from main, there are only three other functions in the program:



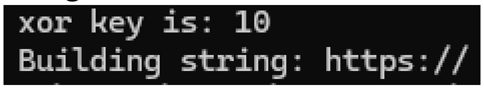
```
sub_40102E
sub_401139
sub_4012CA
```

And when we look at where there is a reference to dword_402000, we see that each of them runs str_cat with it, meaning they each contribute to the total string.
The first function is the only one that doesn't take an argument, so, adding a call to it in the main function:



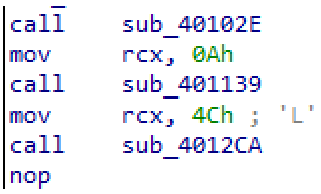
```
call    sub_40102E
```

We get:



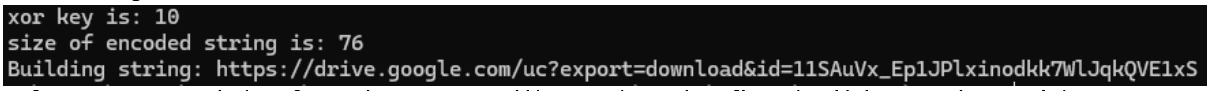
```
xor key is: 10
Building string: https://
```

We will run the second function with the argument 10(using fastcall) and the third function with the print we get from the second(76):



```
call    sub_40102E
mov     rcx, 0Ah
call    sub_401139
mov     rcx, 4Ch ; 'L'
call    sub_4012CA
nop
```

And we get:

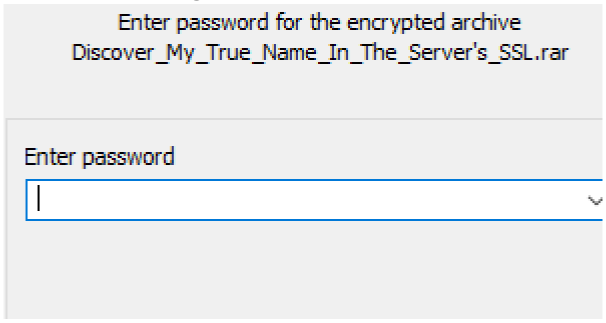


```
xor key is: 10
size of encoded string is: 76
Building string: https://drive.google.com/uc?export=download&id=11SAuVx_Ep1JPlxinodkk7WlJqkQVE1xS
```

(If we reversed the functions we will see that the first builds a string with indecies, the second xors an encrypted string with the received key, and the last does base64 encoding of a length if receives)

4. ZIP lock

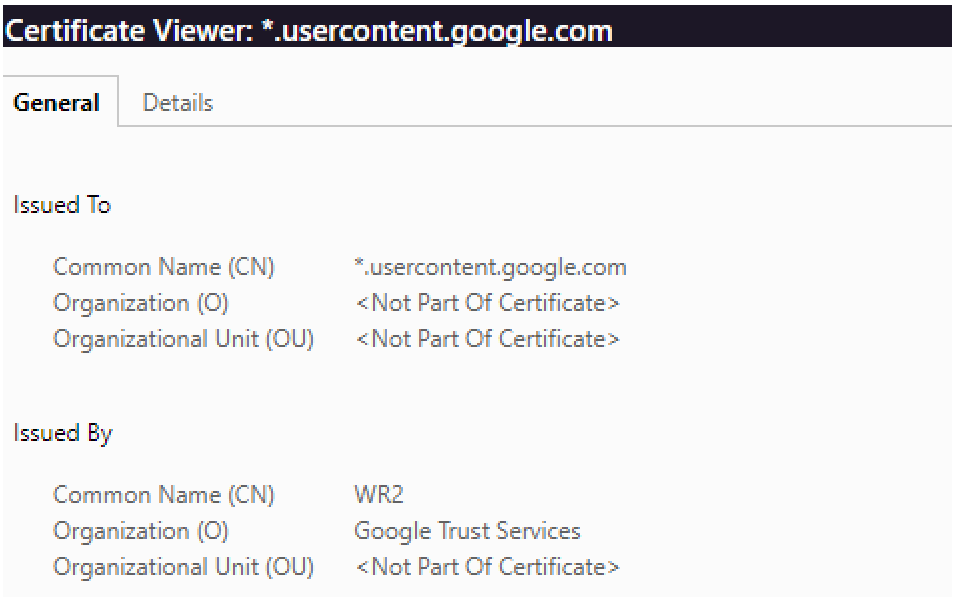
Downloading the file from the link we receive a locked rar file:



Enter password for the encrypted archive
Discover_My_True_Name_In_The_Server's_SSL.rar

Enter password

Going to the certificate of the server from which we downloaded the file(google drive):



Certificate Viewer: *.usercontent.google.com

General Details

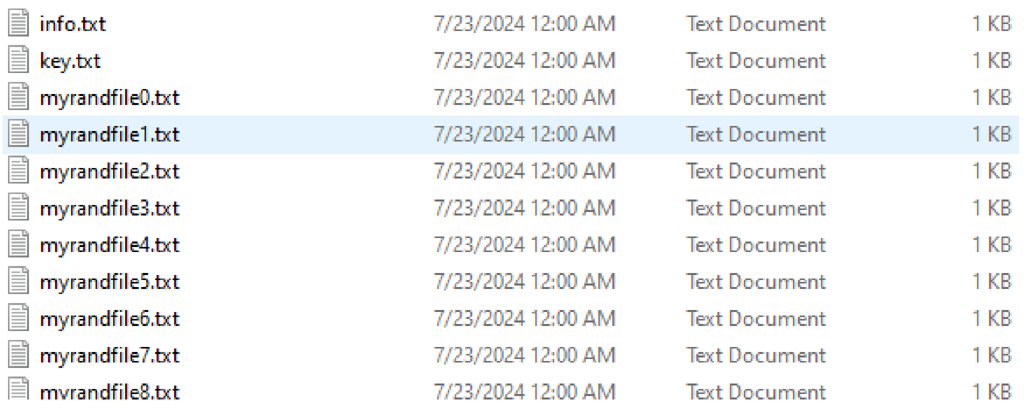
Issued To

Common Name (CN)	*.usercontent.google.com
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	WR2
Organization (O)	Google Trust Services
Organizational Unit (OU)	<Not Part Of Certificate>

We see that the issuer of the crttificate is “Google Trust Services”, entering that gives us:



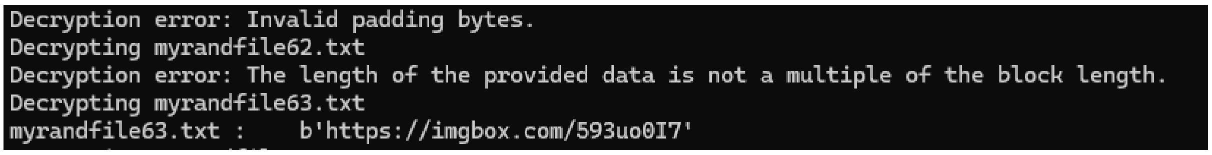
info.txt	7/23/2024 12:00 AM	Text Document	1 KB
key.txt	7/23/2024 12:00 AM	Text Document	1 KB
myrandfile0.txt	7/23/2024 12:00 AM	Text Document	1 KB
myrandfile1.txt	7/23/2024 12:00 AM	Text Document	1 KB
myrandfile2.txt	7/23/2024 12:00 AM	Text Document	1 KB
myrandfile3.txt	7/23/2024 12:00 AM	Text Document	1 KB
myrandfile4.txt	7/23/2024 12:00 AM	Text Document	1 KB
myrandfile5.txt	7/23/2024 12:00 AM	Text Document	1 KB
myrandfile6.txt	7/23/2024 12:00 AM	Text Document	1 KB
myrandfile7.txt	7/23/2024 12:00 AM	Text Document	1 KB
myrandfile8.txt	7/23/2024 12:00 AM	Text Document	1 KB

The info.txt file tells us this the files are encrypted in AES_CBC and that the IV is the first 16 bytes of each file, and that we are looking for one which will decrypt using the key to a readble text.
We will write the following python script:

```python
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.backends import default_backend
import os

# Function to load the key from a file
def load_key(filename='key.txt'):
    with open(filename, 'rb') as key_file:
        return key_file.read()

# Function to decrypt a file
def decrypt_file(file_name, key, output_file):
    with open(file_name, 'rb') as file:
        encrypted_data = file.read()

    iv = encrypted_data[:16]
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()

    try:
        decrypted_data = decryptor.update(encrypted_data[16:]) +
decryptor.finalize()
        unpadded_data = unpadder.update(decrypted_data) + unpadder.finalize()
    except ValueError as e:
        print(f"Decryption error: {e}")
        return

    print(file_name, ":    ", unpadded_data)

def main():
    # Decrypt all files in the current directory
    for filename in os.listdir('.'):
        if filename.endswith('.txt') and filename != 'success.txt':
            print(f"Decrypting {filename}")
            decrypt_file(filename, load_key(), f'dec_{filename}')

if __name__ == "__main__":
    main()
```

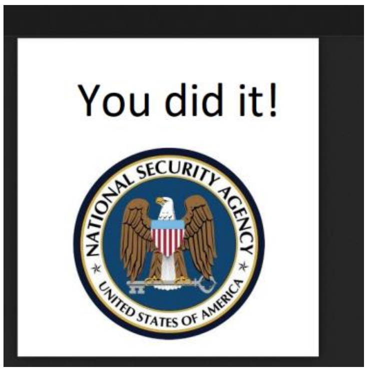
This script loads a key from a file, and runs on every file in the current directory, attempting to decrypt it and prints the result.

Running it gives:



```
Decryption error: Invalid padding bytes.
Decrypting myrandfile62.txt
Decryption error: The length of the provided data is not a multiple of the block length.
Decrypting myrandfile63.txt
myrandfile63.txt :    b'https://imgbox.com/593uo0I7'
```

And opening the link gives us:



Success!

For the scripts, patched files of the solution and every file I used to create this CTF, go here: