# Sneaky

Lets start by firing up wireshark. First thing we notice is that there are around 500 packets here to deal with

Upon checking the `protocol hierarchy` we can see that we are dealing with

the following protocols:

- DNS
- HTTP
- HTTPS
- ICMP
- SMB

Since everything is soo "random" and its getting hard to analyse anything, we sort everything by `Protocol` to get a clear view of what sort of traffic is in the pcap.

## Trolls

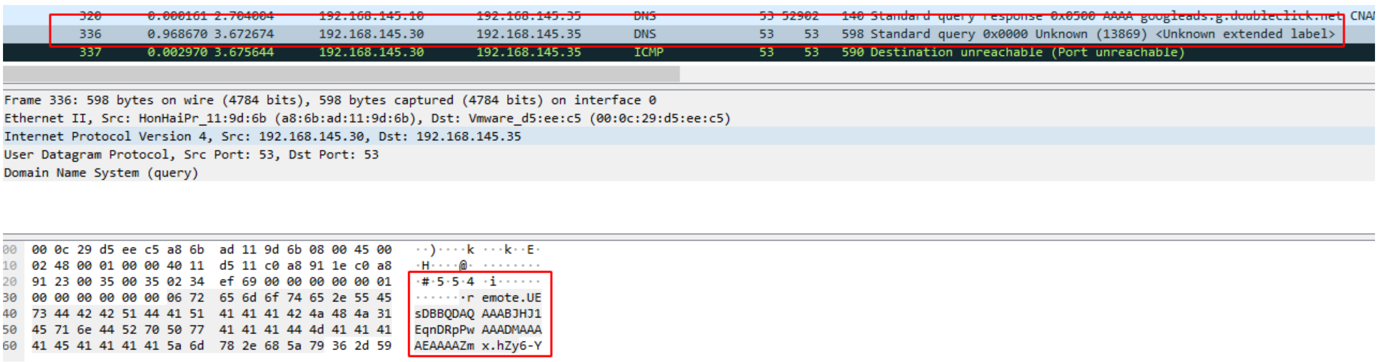
Casually sifting through the traffic, we some files like `flag.txt` and `tryharder.png`, which seem to be just trolls at this stage as the text inside flag.txt just says `TryHarder` and nothing hidden inside the image. Just to make it a little more interesting.

## Sneaky hidden stuff

### Hidden DNS Data

Since most of the traffic is encrypted so for the time being it would only be a complete waste of time to look through it. We start by looking through DNS traffic to look for any abnormal looking queries.

We notice something weird in one of the dns queries:



```
39 77 53 39   ID-XQ4_t HcFc9wS9
53 4f 39 63   NpJNG7BD jF14SO9c
57 61 77 51   ar2-L.nZ -0nggawQ
78 70 6a 75   mgGKcmh_ tXaWxpju
59 31 39 72   VDVCmgKY VDGki19r
44 41 51 41   9.VBLAQI _AxQDAQA
70 50 77 41   AABJHJ1E qnDRpPwA
41 2e 41 41   AADMAAAA EACQA.AA
41 41 41 41   AAAAAAII CkgQAAAA
41 41 41 41   BmbGFnCg AgAAAAAA
75 53 45 31   ABABgAgN N.4BuSE1
59 44 54 65   gEAl0II5 ITWAYDTe
57 41 41 41   AbkhNYBU EsFBgAAA
41 47 45 41   AABAA.EA VgAAAGEA
79 6c 65 67   AAAAAA.r eallyleg
2e 2e 2e 2e   it.com·· ····....
2e 2e 2e 2e   ........ ........
```

That looks fishy, a query with a domain starting with `remote...` and ending with `...reallylegit.com` with some weird encoded data. Now we simply copy this payload from here and put it into a file for further analysis.

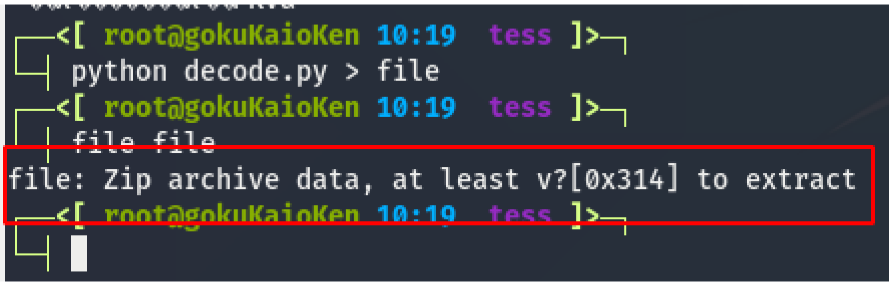

```
remote.UEsDBBQDAQAAABJHJ1EqnDRpPwAAADMAAAAEAAAAZmx.hZy6-YID-XQ4_t
HcFc9wS9NpJNG7BDjF14SO9car2-L.nZ-0nggawQmgGKcmh_tXaWxpjuVDVCmgKYV
DGki19r9.VBLAQI_AxQDAQAAABJHJ1EqnDRpPwAAADMAAAAEACQA.AAAAAAAAIICk
gQAAAABmbGFnCgAgAAAAAAABABgAgNN.4BuSE1gEAl0II5ITWAYDTeAbkhNYBUEsF
BgAAAAABAA.EAVgAAAGEAAAAAAA.reallylegit.com
```


After fiddling around it for a while we notice that this is `Base64-URL` without any padding, so I made a script to decode it the base64

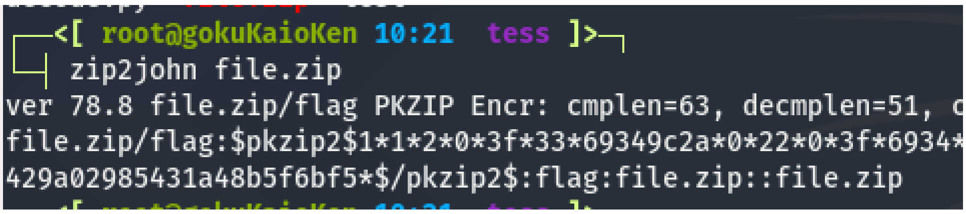
```python
import base64
data="UEsDBBQDAQAAABJHJ1EqnDRpPwAAADMAAAAEAAAAZmx.hZy6-YID-XQ4_t
HcFc9wS9NpJNG7BDjF14SO9car2-L.nZ-0nggawQmgGKcmh_tXaWxpjuVDVCmgKYV
DGki19r9.VBLAQI_AxQDAQAAABJHJ1EqnDRpPwAAADMAAAAEACQA.AAAAAAAAIICk
gQAAAABmbGFnCgAgAAAAAAABABgAgNN.4BuSE1gEAl0II5ITWAYDTeAbkhNYBUEsF
BgAAAAABAA.EAVgAAAGEAAAAAAA"

def decode():
# https://gist.github.com/catwell/3046205
    _data = data.replace('.', '').replace('_', '/').replace('-', '+')
    padding = len(_data) % 4
    if padding == 2:
        _data += '=='
    elif padding == 3:
        _data += '='
```

```
    return base64.b64decode(_data)


if __name__ == '__main__':
    decoded_data = decode()
    print(decoded_data)
```

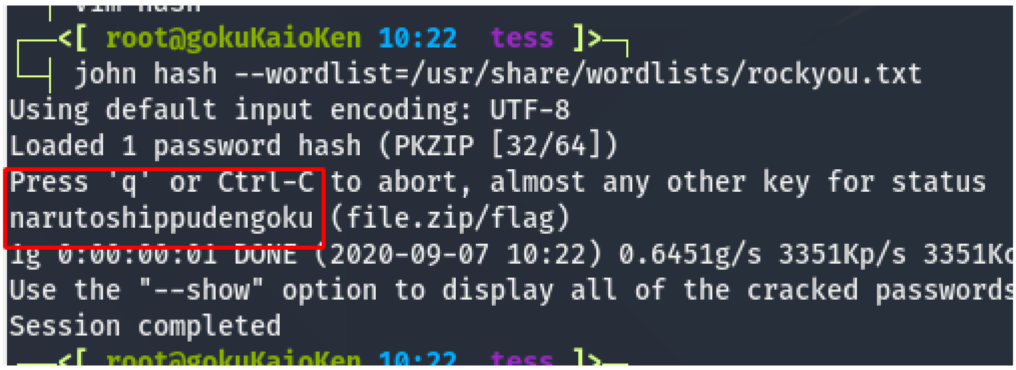
Which outputs some rubbish, but after redirecting the "rubbish" to a file and running a file command on it tells me that its a `zip` file.




Unfortunately this zip file is protected by a password. We try to bruteforce this using john and rockyou to get the password. To do that we first use `zip2john` to create a hash which we can crack using john.




We can see straight away that the password is `narutoshippudengoku`.

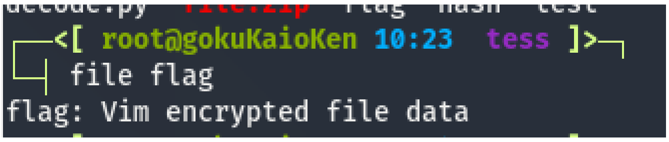



unzipping the file we have `flag` in there BUT ....

### The Final Boss

Now comes the brainy part of the challenge which will probably break alot of them !

after decrypting the zip file, we find the `flag` but unfortunately we cannot read it as its `encrypted`(WTF!).





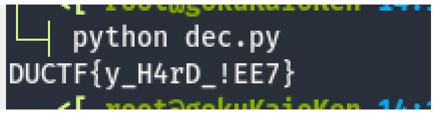
If anyone has used `ViM` before(which you should), then it should be clear that its encrypted using one of the algorithms supported by vim. Upon further digging we can find that VimCrypt has 3 encryption modes which is 1 = pkzip, 2 = blowfish and 3 = blowfish2. From the header of the file, we can see `VimCrypt~02!` which suggest that this is `blowfish` encrypted which is another `XOR cipher` encryption. Upon further googling we can find this link(below) which tells us why its vulnerable and how you can get the plaintext without any key if you have know some part of the plaintext(flag format).
[Vulnerable ViM algorithm]

It says that the first 28 bytes are header, made up of the encryption descriptor (12), salt (8) and IV (8), so if you XOR the first block with the plaintext and keep Xoring the remaining with the previously xored block then you can retrieve some part of the plaintext. I made this quick script to retrieve the flag based on this vulnerability.

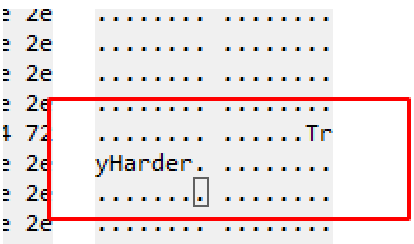
```
#!/bin/env python

import os
import sys
import itertools

plaintext = bytes("DUCTF{")
print plaintext
blocks = []
keystream = ""

def xor(block, key):
    return ''.join(chr(ord(x) ^ ord(y)) for (x,y) in itertools.izip(block, key))

with open("flag","rb") as infile:
    header = infile.read(12)
    salt = infile.read(8)
    iv = infile.read(8)

    blocks.append(infile.read(8))
```

```
    blocks.append(infile.read(8))
    blocks.append(infile.read(8))
    blocks.append(infile.read(8))

    key = xor(blocks[0],plaintext)

    plaintext+=xor(blocks[1],key)
    plaintext+=xor(blocks[2],key)
    plaintext+=xor(blocks[3],key)

    print plaintext
```

Now here we already know that the known part is actually the flag format `DUCTF{` so we run this script with that as the known part.




From here we can see that this `H4rD` looks like the word `Hard` and the word before that ends with `y` so we can take an easy guess that it must be the word `Try` so it becomes `Try_Harder`. So now the known part becomes `DUCTF{Tr` and one we run the script again we finally get the flag !





gokuKaioKen