Important Information:

ssh -p 2231 krypton6@krypton.labs.overthewire.org

Passwords:

- Level0 --> 1: KRYPTONISGREAT
- Level1 --> 2: ROTTEN
- Level2 --> 3: CAESARISEASY
- Level3 --> 4: BRUTE
- Level4 --> 5: CLEARTEXT
- Level5 --> 6: RANDOM
- Level6 --> 7: LFSRISNOTRANDOM

Level 1 --> 2:

- cd /krypton/
- encrypted using simple rotation called ROT13
- echo "YRIRY GJB CNFFJBEQ EBGGRA" | tr 'A-Za-z' 'N-ZA-Mn-za-m'
- LEVEL TWO PASSWORD ROTTEN

Level 2 --> 3:

- Monoalphebetic cipher (one alphabet)
- Caesar Cipher (shifts alphabet by set number)
- Password for level 3 is in file krypton3
- encrypt is setuid executable (setuid allows user to perform tasks that normally require higher privileges)

```
krypton2@bandit:/tmp/tmp.XgqGNK4gX2$ ln -s /krypton/krypton2/keyfile.dat
krypton2@bandit:/tmp/tmp.XgqGNK4gX2$ ls
keyfile.dat
```

```
krypton2@bandit:/tmp/tmp.XgqGNK4gX2$ cat /etc/issue
Ubuntu 24.04.1 LTS \n \l
```

```
krypton2@bandit:/tmp/tmp.XgqGNK4gX2$ cat mommy
ABCDEFGHIJKLMNOPQRSTUVWXYZ
krypton2@bandit:/tmp/tmp.XgqGNK4gX2$ cat ciphertext
MNOPQRSTUVWXYZABCDEFGHIJKLkrypton2@bandit:/tmp/tmp.XgqGNK4gX2$
```

```
krypton2@bandit:/tmp/tmp.XgqGNK4gX2$ cat /krypton/krypton2/krypton3 | tr "[MNOPQRSTUVWXYZABCDEFGHIJKL]" "[A-Z]"
CAESARISEASY
krypton2@bandit:/tmp/tmp.XgqGNK4gX2$
```

- It is my assumption that we have to use encrypt to run krypton3 privileges to be able to open keyfile.dat. Since me (krypton2) has full control over the temporary directory, and by modifying the tmp directory's permission, I can let encrypt (running as krypton3) read keyfile.dat via the symbolic link, enabling me to decrypt the ciphertext indirectly.

The Condition `if [[ ${frequency[$char]+_} ]]; then`:

1. **When the key exists in `frequency`:**
   - `${frequency[$char]+_}` evaluates to `_`.
   - The condition inside the `if` statement is true because `_` is a non-empty string.
   - The `then` block runs.
2. **When the key does not exist in `frequency`:**
   - `${frequency[$char]+_}` evaluates to an empty string.
   - The condition inside the `if` statement is false.
   - The `then` block is skipped.

    - mktemp -d (/tmp/tmp.Wf2OnCpCDQ)
    - cd /tmp/tmp.Wf2OnCpCDQ
    - ln -s (this creates a link) to the keyfile.dat in my temporary directory --> allowing encrypt to access the key-file in the current working directory.
    - chmod 777 .
    - /krypton/krypton2/encrypt /etc/issue --> we will use the encrypt program to decode the /etc/issue (as a test run) so that it can pump out a ciphertext file of the decoded message according to the keyfile.dat (using the encrypt program).
    - What we have now is that we can run through anything in the encrypt file to decode things (this is a chosen plain text) --> so let us run the whole alphabet through to see.

- touch mommy, nano mommy, cat mommy = ABCDEFGHIJKLMNOPQRSTUVWXYZ
- ciphertext returns = MNOPQRSTUVWXYZABCDEFGHIJKL
  - tr translates characters. Replacing M = A, N = B, O = C etc.
  - ["A-Z"] stands for ABCDEF...

Level 3 --> 4:

- We will try to complete this level without an encryption key (like we had in the previous level)
- The password to the next level is found in the file 'krypton4'. You have also found 3 other files. (found1, found2, found3).
- Some letters are more prevalent in English than others.
- "Frequency Analysis" is your friend.
- cat krypton4 = KSVVW BGSJD SVSIS VXBMN YQUUK BNWCU ANMJS
  - Count the frequency of letters across all the ciphertexts
  - Compare the frequencies to English letter frequencies.
  - Deduce the substitution key and decode the files.
  - "while IFS= read -n1 char" reads the file one character at a time (-n1 means 1 by 1).
  - To be able to work with files in /krypton/krypton3 while using temporary directory, you use symbolic link.
  - The '.' means current working directory (you can check that with pwd)

Steps:

```
krypton3@bandit:/tmp/tmp.jf7KnT4dTo$ ls -la
total 17012
drwx------ 2 krypton3 krypton3     4096 Jan 11 04:01 .
drwxrwx-wt 1 root     root     17412096 Jan 11 04:02 ..
lrwxrwxrwx 1 krypton3 krypton3       24 Jan 11 04:01 found1 -> /krypton/krypton3/found1
lrwxrwxrwx 1 krypton3 krypton3       24 Jan 11 04:01 found2 -> /krypton/krypton3/found2
lrwxrwxrwx 1 krypton3 krypton3       24 Jan 11 04:01 found3 -> /krypton/krypton3/found3
```

- mktemp -d
- cd /tmp/tmp.jf7KnT4dTo
- ln -s /krypton/krypton3/found1 .
- ln -s /krypton/krypton3/found2 .
- ln -s /krypton/krypton3/found3 .
- nano analysis.sh (put in the commands)
- chmod +x analysis.sh (to make it executable)
- ./analysis.sh

```
Top 6 Single Letters:
S: 456 (12.93%)
Q: 340 (9.64%)
J: 301 (8.53%)
U: 257 (7.28%)
B: 246 (6.97%)
N: 240 (6.80%)

Top 6 Bigrams:
JD: 75 (2.65%)
DS: 66 (2.34%)
SN: 52 (1.84%)
CG: 49 (1.73%)
SU: 49 (1.73%)
QN: 44 (1.56%)

Top 6 Trigrams:
JDS: 37 (1.75%)
DSN: 15 (0.70%)
DCU: 14 (0.66%)
QGW: 14 (0.66%)
SQN: 14 (0.66%)
SNS: 13 (0.61%)
```

Matching:

Password: KSVVW BGSJD SVSIS VXBMN YQUUK BNWCU ANMJS

Password: WELLD ONETH ELEVE LFOUR PASSW ORDIS .....

--> WELL DONE THE LEVEL FOUR PASSWORD IS

ANMJS = -RUTE (BRUTE)

Level 4 --> 5 Info:

- So far we have worked with simple substitution ciphers. They have also been 'monoalphabetic', meaning using a fixed key, and giving a one to one mapping of plaintext (P) to ciphertext (C).
- We will now do 'polyalphabetic cipher' --> an example is called Vigenère Cipher
- Key length in this exercise is 6
- Frequency analysis still works, but you need to analyse it by "keylength" (analyse cipher text at 1, 6, 12 etc). Treat this as 6 different mono-alphabetic ciphers. The length of the key determines how

often the substitution rules repeat. Instead of one number that is used to encrypt the whole text, the Vigenère Cipher uses a secret key.

- password = HCIKV RJOX (must decrypt ciphertext)
- Using online software on the found1 and found2, I have deduced that the key = frekey
- Using online software for a decoder, I have deduced --> CLEARTEXT

Level 5 --> 6:

- password = BELOS Z
- The ciphertext likely has patterns and repetitions that can help deduce the key length
- Search for repeating substrings of three or more characters (common prime divisors --> 2, 3, 5, 7)
- Record the distances between their occurrences.

Kasiski Examination:

Find Repeated Sequences:

- "XJGLA": (41, 42, 43, 44, 45), (131, 133, 133, 134, 135), (176, 177, 178, 179, 180) --> 41, 131, 176
- "CTZDL": (61, 62, 63, 64, 65), (106, 107, 108, 109, 110) --> 61, 106
    - Delta Values = difference between each index
- Delta Values --> 90, 45
- Delta Values --> 45
- Look for factors that match into 45 and 90 (either 5 or 9) --> if you had to guess, guess the most secure option which is the longer key.
    - Using software on found1 for a keyword solver --> key = "keylength"
    - Using software on found2 for a keyword solver --> key = "keylength"
        - Using software to solve for the password = BELOS Z
        - Password --> RANDOM

Level 6 --> 7 Information:

- Frequency analysis can destroy repeating/fixed key substitution crypto.
- Modern ciphers are similar to older plain substitution ciphers, but improve the ‘random’ nature of the key.
- Modern ciphers come in two types: symmetric and asymmetric.
    - Symmetric ciphers come into two flavours: block and stream.
        - Block ciphers use a fixed key to perform substitution and transposition ciphers on each block discretely.
        - Its time to employ a stream cipher. A stream cipher attempts to create an on-the-fly ‘random’ key-stream to encrypt the incoming plaintext one byte at a time.
        - From this example forward, we will be working with bytes, not ASCII text, so a hex editor/ dumper like hexdump is a necessity. Now is the right time to start to learn to use tools like cryptool.
    - HINT1 = The 'random' generator has a limited number of bits, and is periodic. Entropy analysis and good look at the bytes in a hex editor helps (there is a pattern)
    - HINT2 = 8 bit LFSR (The random generator is an 8-bit LFSR, meaning it generates a repeating pseudo-random sequence.)
- A One-Time Pad is an encryption method where:

- The key is as long as the plaintext
- The key is used only once
- The ciphertext is truly random and unbreakable if the key is random and never reused
- Since you can't directly read the keyfile, you'll use plaintext attacks to exploit the weak random generator.
- The program encrypt6 lets you encrypt your own plaintext.
- By comparing the plaintext and resulting ciphertext, you can reverse-engineer the encryption process and figure out the key or randomness.

Level 6 --> 7:

- password = PNUKLYLWRQKGKBE
- encrypt6 = setuid (must create a link then to decrypt keyfile.dat)
- mktemp -d
- cd /tmp/tmp.x91MCADjlT
- ln -s /krypton/krypton6/* /tmp/tmp.x91MCADjlT/
- chmod 777 .
- Now we do a plaintext attack by going: echo "(A * 40)" > plaintext.txt
- ./encrypt6 plaintext.txt ciphertext.txt
- hexdump -C ciphertext.txt (40 A's = EICTDGYIYZKTHNSIRFXYCPFUEOCKRN EICTDGYIYZ)
  - Every A (in plaintext) gets mapped to a different letter in ciphertext (similar to Vigenère cipher)
  - key = EICTDGYIYZKTHNSIRFXYCPFUEOCKRN