

Cotd Day 0 Write-Up:

-By spyd3r

In this challenge, participants were tasked with uncovering a hidden message encoded in four layers of Hexadecimal and Base64 formats. The following steps outline the process to decode the layered ciphertext and retrieve the flag.

Step 1: Initial Cipher Identification

The ciphertext was analyzed for encoding patterns, revealing that it was encoded through alternating layers of Hexadecimal and Base64. The decoding would require reversing these layers in the correct order to extract the final message.

Hints Provided:

- $4^2(16 \rightarrow \text{hexadecimal})$ and $4^3(64 \rightarrow \text{base64})$ were given as clues, suggesting the number of encoding layers and possible relationships to the structure of the message.

Step 2: Decoding the Cipher

The ciphertext followed this pattern: Hex \rightarrow Base64 \rightarrow Hex \rightarrow Base64. Below are the steps to decode the message:

In Terminal:

1. First Layer (Base64):

```
echo "base64_string" | base64 --decode
```

2. Second Layer (Hex):

```
echo "hex_string" | xxd -r -p
```

Repeat the same steps for the final two layers of Hex and Base64 decoding until the plaintext is revealed.

Step 3: Decoding via CyberChef

Alternatively, CyberChef can be used to decode the layers visually:

1. Paste the encoded string into the input section.
2. Apply From Base64 to get the Hex encoded data.
3. Apply Hex Decode to get the next Base64 string.
4. Repeat the operations until the plaintext is revealed.

Step 4: Conclusion

Both methods, using command-line tools and CyberChef, successfully decoded all four layers of encoding. The final message revealed the flag: COTD{welc0m3_2_c0td_h3cker5}.

Terminal Solution:

```
cyphe@club ~  
➤ echo NGU0NDRkMzA1YTZhNTUzMDRlNDQ1MTMzNTk2YTYzMzM0ZTZhNTUzMju5N2E1OTdhN  
GQ3YTQxMzI1YTQ0NGQ3YTRlNTc1OTdhNGQ2YTU2NmQ0ZTZhNGQ3YTRkNDQ2MzMwNGU2YTUxMzE1  
YTZhNTkzNDRkN2E0ZDMYNGQ3YTVhNjk0ZTZhNTUzMzRkNmE0ZDMxNGUzMjUxM2Q= | base64 -  
d | xxd -r -p | base64 -d | xxd -r -p  
COTD{welc0m3_2_c0td_h3cker5}
```

Cyberchef Solution:

The image displays four screenshots of the CyberChef web application, illustrating various recipes and their outputs.

Top Left Screenshot: Shows the 'Operations' sidebar on the left. The 'Recipe' panel is active, showing a recipe named 'From Base64' with the alphabet 'A-Za-z0-9+/=' and the option 'Remove non-alphabet chars' checked. The 'Input' panel contains a long Base64-encoded string. The 'Output' panel shows the decoded hexadecimal string: `4a5444d380a6a55384e46513596a63334e6a5532597a597a6d7a61325a44d7a4e57597a46a566d4e6a4d7a4d4463384e6a5531515a6a59344d7a4d324d7a5a694e6a553346a64514e325134`.

Top Right Screenshot: Shows the 'Recipe' panel with a recipe named 'From Base64' and the option 'Remove non-alphabet chars' checked. The 'Input' panel contains a long Base64-encoded string. The 'Output' panel shows the decoded hexadecimal string: `4a5444d380a6a55384e46513596a63334e6a5532597a597a6d7a61325a44d7a4e57597a46a566d4e6a4d7a4d4463384e6a5531515a6a59344d7a4d324d7a5a694e6a553346a64514e325134`.

Bottom Left Screenshot: Shows the 'Recipe' panel with a recipe named 'From Base64' and the option 'Remove non-alphabet chars' checked. The 'Input' panel contains a long Base64-encoded string. The 'Output' panel shows the decoded hexadecimal string: `434f54447b77656c63386d335f325f633874645f6833636865723257d`.

Bottom Right Screenshot: Shows the 'Recipe' panel with a recipe named 'From Base64' and the option 'Remove non-alphabet chars' checked. The 'Input' panel contains a long Base64-encoded string. The 'Output' panel shows the decoded hexadecimal string: `c0td[we1c0a3_2_c0td_h3ckers]`.

Cotd Day 1 Write-Up:

-by armoredvortex

In this challenge, participants are tasked with uncovering a hidden message that has been encoded through two different methods. The first layer uses the **Vigenère Cipher**, and the second transforms the encoded text into a **Morse Code audio file**. The goal is to reverse these steps to retrieve the hidden flag.

Step 1: Analyzing the audio file

Upon playing the audio.wav file, one can immediately recognize the distinctive **short beeps (dots)** and **long beeps (dashes)**, identifying it as Morse Code.

Step 2: Transcribing the Morse Code

If you're familiar with Morse Code, you may begin transcribing manually as you listen by hand.

Alternatively, you can use online tools to help transcribe the Morse Code directly from the audio. One such tool is available at morsecode.world

After uploading the audio.wav file and clicking “Play”, the hidden message is revealed.

Step 3: Identifying the unknown cipher

After transcribing the Morse Code, you obtain the hidden message:

ESTJI BK EYVWHEI YZXA NMIVRKW GKGLXJ

To determine which cipher was used, you can employ online cipher identification tools, such as [dcode cipher identifier](#) or [boxentriq cipher identifier](#).

Upon analysis, you'll discover that this is a Vigenere Cipher

Step 4: Decrypting the Vigenere Cipher

Now you know this message is encrypted using the Vigenère Cipher, and your task now is to decrypt it using the correct key.

Identifying the key:

Referring back to the original prompt, “**To crack it, you'll need a key—one that's secret**”. This is likely a wordplay indicating that the key we're looking for is the word ‘**secret**’ itself.

Decrypt using the key:

This can be done using various methods:

1) Using a Vigenère Table:

- A Vigenère Table (also called a tabula recta) is a grid that helps you manually decrypt the cipher by matching the ciphertext letters with the key. However, this method can be quite tedious and time-consuming, especially for longer messages.
- Because of the manual nature of this process, it can be beyond the scope of this write-up. For those interested in understanding how to use the Vigenère Table in detail, you can refer to the [Vigenère Cipher article on Wikipedia](#).

2) Using Online Tools:

- There are many Vigenère Cipher decryption tools available online, for example cryptii.com

Both methods will give you the plaintext message hidden in the ciphertext.

Step 4: Conclusion

Both methods—manual (by-hand) or command-line and online resources—successfully decoded the Morse Code and decrypted the Vigenère Cipher. The final message revealed the flag: **COTD{morse_is_awesome_with_vigenre_cipher}**.

morsecode.world:

International Morse Decoders

Audio DecoderAudio Decoder (Expert)Gaze Decoder

Use the microphone:

Or analyse an audio file containing Morse code:

Listen

Stop

Upload

Play

Stop

Filename: "audio.wav"

ESTJI BK EYVWHEI YZXA NMIVRKW GKGLXJ

Clear Message

WPM

Farnsworth WPM

Frequency (Hz)

Minimum volume

Maximum volume

Volume threshold

20

20

750

-60

-30

200

☐ Manual

☐ Manual

Identifying the unknown Cipher:

Search for a tool

★ SEARCH A TOOL ON dCode BY KEYWORDS:
e.g. type 'caesar'

★ BROWSE THE [FULL dCode TOOLS' LIST](#)

Results

dCode's analyzer suggests to investigate:

Warning The text has a **short length**, this can affect the quantity and reliability of the results (see FAQ)

Warning Few or no significative results (see FAQ)

↑↓	↑↓
Vigenere Cipher	■
Autoclave Cipher	■
Beaufort Cipher	■
Rozier Cipher	■
Vernam Cipher (One Time Pad)	■
Variant Beaufort Cipher	■
Gronsfeld Cipher	■
Enigma Machine	■

CIPHER IDENTIFIER

Cryptography > Cipher Identifier

ENCRYPTED MESSAGE IDENTIFIER

★ CIPHERTEXT TO RECOGNIZE (?)
ESTJI BK EYVWHEI YZXA NMIVRKW GKGLXJ

★ CLUES/KEYWORDS (IF ANY)

See also: [Frequency Analysis](#) — [Index of Coincidence](#)

SYMBOLS IDENTIFIER

▶ Go to: [Symbols Cipher List](#)

Answers to Questions (FAQ)

What is a cipher identifier? (Definition)

A encryption detector is a computer tool designed to recognize encryption/encoding from a text message. The detector performs cryptanalysis, examines various features of the text, such as letter distribution, character repetition, word length, etc. to determine the type of encryption and guide users to the right tools based on the type of code or encryption identified.

cryptii.com:

VIEW

Plaintext ▾

MORSE IS AWESOME WITH VIGENRE CIPHER

+

ENCODE DECODE

Vigenère cipher ▾

VARIANT
Standard Vigenère cipher ▾

KEY
secret

KEY MODE
Repeat ▾

VIEW

Ciphertext ▾

ESTJI BK EYVWHEI YZXA NMIVRKW GKGLXJ

COTD Day 2 Write - Up



(By Miracle Invoker)

Problem Statement:

Find out the flag using the script below

```
def encrypt(message, key):
    return ''.join([chr(ord(char) ^ key) for char in message])

message = "----FLAG----"

# Just Like yesterday you would need a key today, but this time you will gu
# The Key is
# An Integer Less than 100
# Multiplication of two primes
# Favourite number of Guys

key = "----INT----"

encrypted_message = encrypt(message, key).encode().hex()

print("Encrypted Message:", encrypted_message)

## OUTPUT OF THE SCRIPT ##
## Encrypted Message: 3d75371a74361a37203376373674272976 ##
```

Understand the Script, and enclose the message in

COTD{}

In this challenge participants are tasked with decrypting the following encrypted message:

3d75371a74361a37203376373674272976

The string is encoded in hexadecimal as seen by the line `encrypt(message, key).encode().hex()`

Our first step would be to decode the string back to its' original form which is in ASCII .

```
encryptedMESSAGE = "3d75371a74361a37203376373674272976"  
encryptedMESSAGE = bytes.fromhex(encryptedMESSAGE).decode()
```

Note: It is recommended to do the above directly in the code as using online Hex to ASCII Converters can cause loss of data due to Non Printable ASCII Characters.

Now our `encryptedMESSAGE` string contains the output of

```
def encrypt(message, key):  
    return ''.join([chr(ord(char) ^ key) for char in message])
```

Now we need to make a algorithm to reverse the changes done by the `encrypt` function.

Let us analyze the function,

It uses two functions [chr](#) and [ord](#) .

You can read the documentation about these two functions which are inverse of each other.

After we get the ASCII value of the character from `ord(char)` , we use the [BITWISE XOR](#) operation on it with the `key` .

The rest of the function iterates through the characters and joins them in the end.

The interesting property of the XOR (\oplus) is that

If

$$a \oplus b = c$$

then

$$c \oplus b = a$$

or in our terms.

$$\text{message} \oplus \text{key} = \text{encryptedMESSAGE}$$

$$\text{encryptedMESSAGE} \oplus \text{key} = \text{message}$$

Hence, the decryption process is exactly the same as the encryption.

This is known as [Simple XOR Cipher](#).

Now we can find the key using the given hints.

key is an integer less than 100 and is a multiple of two primes which means it is a [Semiprime](#).

We can write a [simple algorithm](#) for checking Semiprimes then loop through 1 to 99 while applying the encrypt function each iteration.

```
encryptedMESSAGE = "3d75371a74361a37203376373674272976"
encryptedMESSAGE = bytes.fromhex(encryptedMESSAGE).decode()

def encrypt(message, key):
    return ''.join([chr(ord(char) ^ key) for char in message])

def isSEMIPRIME(num):
    count = 0
    i = 2
    while (i * i <= num and count < 2):
        while (num % i == 0):
            num //= i
            count += 1
        i += 1
    if num > 1:
        count += 1
    return count == 2


for i in range(100):
    if isSEMIPRIME(i):
        print(encrypt(encryptedMESSAGE, i))
```

Manually checking the outputs reveals the flag as x0r_1s_rev3rs1b13

Summary

Each character on a computer is assigned a unique code and the preferred standard is ASCII (American Standard Code for Information Interchange). For example, uppercase A = 65, asterisk (*) = 42, and lowercase k = 107.

A modern encryption method is to take a text string, convert each character to ASCII, then XOR each integer with a given value, taken from a secret key. The advantage with the XOR function is that using the same encryption key on the cipher text, restores the plain text; for example, $65 \text{ XOR } 42 = 107$, then $107 \text{ XOR } 42 = 65$.



Day 3 Writeup

Upon inspecting the webpage at the provided [link](#), you will notice that it instructs you to go to /hidden.txt, with a hint that **even blank spaces have meaning**. When visiting this [endpoint](#), you'll see a blank page with no visible text. However, following the hint, if you try to select text, you will discover that something is being selected.

Next, you can use the cipher identifier tool at [dcode.fr](#). It will indicate that the Whitespace language is being used. By decoding the content of the webpage with the tool provided at [dcode.fr](#) for Whitespace language, you will be able to uncover the flag. [Whitespace-language](#).

Following is a python script to automate the process,

```
import requests
url="https://web.infosec.org.in/hidden.txt"
response=requests.get(url)
responseText = response.text.replace(" ", "0").replace("\t", "1").replace("
# " " -> 0          Replace spaces with zeroes.
# "\t" -> 1          Replace the tabs with ones.
# "\n1" -> ""        Replace the extra \n1's with "".
# Then create a list of the binary data (they are separated by '\n').
for i in responseText:
    if len(i) == 0: continue
    # Skip the data if it is 0 in length to prevent error while converting
    print(chr(int(i, 2)), end="") # convert the ascii values back to char
```

flag{didn't_think_blank_spaces_can_hold_messages}

Day 7 writeup

```
import random
score = input()
score=score.split()
count = 0
while True:
    random.seed(count)
    l = [random.randint(1,6) for _ in range(10)]

    l = [str(int(int(score[i])== l[i])) for i in range(len(score))]

    if(''.join(l) == '1111111111'):
        print(count)
        ll = [random.randint(1,6) for _ in range(10)]
        print(ll)
        break

    count += 1
```