Tap into Hash - picoCTF 2024

Cryptography Challenge

Challenge Information

Challenge Name	Tap into Hash
Category	Cryptography
Difficulty	Medium
Event	picoCTF 2024
Author	NGIRIMANA Schadrack

Description

Can you make sense of this source code file and write a function that will decode the given encrypted file content? Find the encrypted file here. It might be good to analyze source file to get the flag.

Files Provided:

- `block_chain.py` - Source code file
- `enc_flag` - Encrypted flag file

Hints:

- Do you know what blockchains are? If so, you know that hashing is used in blockchains
- Download the encrypted flag file and the source file and reverse engineer the source file

Solution

TL;DR

1. Analyze blockchain encryption source code
2. Extract encryption key from enc_flag
3. Decrypt using XOR with SHA256(key)
4. Extract flag from middle of decrypted blockchain string

Initial Analysis

The challenge provides two files:

- `block_chain.py` - Contains the encryption implementation

- `enc_flag` - Contains the encryption key and encrypted blockchain

Understanding the Encryption

Let's analyze the encryption process in `block_chain.py`:

1. Blockchain Creation

The program creates a simple blockchain:

- Genesis block (index 0)
- 4 additional blocks with proof-of-work (hash must start with '00')
- Each block contains: index, previous_hash, timestamp, transactions, nonce

2. Blockchain to String

The blockchain is converted to a string of hashes separated by dashes:

```
hash1-hash2-hash3-hash4-hash5
```

3. Flag Insertion

The critical `encrypt()` function does the following:

```
def encrypt(plaintext, inner_txt, key):
    midpoint = len(plaintext) // 2
    first_part = plaintext[:midpoint]
    second_part = plaintext[midpoint:]
    modified_plaintext = first_part + inner_txt + second_part
```

The `inner_txt` (which is the flag) is inserted in the middle of the blockchain string!

4. XOR Encryption

After inserting the flag, the data is encrypted using XOR:

- Data is padded to 16-byte blocks
- Key is hashed with SHA256
- Each block is XORed with the key hash

Encryption Key Analysis

Looking at the `enc_flag` file, we find:

```
Key: b'\xffo\xb6\x94g\xe1%\xfd`\x1f-
\x1b\x8a\xab\xc5\xb5\xech\x82Q\xe4\xda1}+\x1c\xf1\x91\x83z\x12;'
```

Excellent! We have the encryption key! This means we can decrypt the data.

Decryption Strategy

Since XOR encryption is symmetric (A ⊕ B = C means C ⊕ B = A), we can decrypt by:

5. Computing SHA256(key) to get the same key hash

6. XORing each 16-byte block of ciphertext with the key hash
7. Removing padding from the result
8. Extracting the flag from the middle of the decrypted string

Decryption Script

Here's the decryption script:

```
import hashlib

# Data from enc_flag
key = b'\xffo\xb6\x94g\xe1%\xfd`\x1f-
\x1b\x8a\xab\xc5\xb5\xech\x82Q\xe4\xda1}+\x1c\xf1\x91\x83z\x12;'
encrypted_blockchain = b'\xb8\x948/\xad\x03d\xd8\xa0WOp\xf4\x92...' #
truncated

def xor_bytes(a, b):
    return bytes(x ^ y for x, y in zip(a, b))

def unpad(data):
    padding_length = data[-1]
    return data[:-padding_length]

# Decrypt
key_hash = hashlib.sha256(key).digest()
block_size = 16

plaintext = b''
for i in range(0, len(encrypted_blockchain), block_size):
    cipher_block = encrypted_blockchain[i:i + block_size]
    plain_block = xor_bytes(cipher_block, key_hash)
    plaintext += plain_block

# Unpad and decode
plaintext = unpad(plaintext)
print('Decrypted plaintext:')
print(plaintext.decode())
```

Output

Running the decryption script produces:

```
Decrypted plaintext:
8b9748b7a7e6030bcb1dfbae48b9ce1e6fce05cd0017a9023b85ca40abaf0de4-
00ef9ad789d4b3730b9cb42f1cbaa84fbd7082cb61ee5cddec83cd87551cab58-
003014760c8c4cba0d64bb17787a4fd4
picoCTF{block_3SRhViRbT1qcX_XUjM0r49cH_qCzmJZzBK_27c69efb}
```

`654b087cd7e2dc433e25b51e0ed7adad-005ecdeca28563e8d48a3d2f90f67ec10464d80a66eea6a345da73ecd1f45343-00f1bb000dffe780fafd60ee68d912cb762b91f8117c42e52c0600d40acf281c`

The structure is clear: `blockchain_hash1-hash2-hash3-FLAG-hash4-hash5-hash6`

The flag was successfully extracted from the middle of the blockchain string, exactly as designed in the encryption algorithm!

Flag

`picoCTF{block_3SRhViRbT1qcX_XUjM0r49cH_qCzmJZzBK_27c69efb}`

Tools Used

- **Python 3** - For decryption script
- **hashlib** - SHA256 hashing library

Key Takeaways

- **Source code analysis** - Understanding the encryption implementation is crucial for finding the attack vector
- **XOR properties** - XOR encryption is symmetric: if $A \oplus B = C$, then $C \oplus B = A$
- **Key exposure vulnerability** - Having access to the encryption key completely breaks the security
- **Data insertion technique** - The flag was cleverly hidden by inserting it in the middle of legitimate blockchain data
- **Blockchain structure** - Understanding basic blockchain concepts (blocks, hashes, proof-of-work) helps in analyzing the code

References

- Python hashlib documentation: https://docs.python.org/3/library/hashlib.html
- XOR cipher: https://en.wikipedia.org/wiki/XOR_cipher
- Blockchain basics: https://en.wikipedia.org/wiki/Blockchain

Author
Glenvio Regalito Rahardjo

Solved: December 2024