Proof-of-Concept Report: Fonix ransomware decryption tool Fury decrypting tool

Intern Name: **Sayali Vijay Pol**

Intern ID: **345**

Organization: **Digisuraksha parhari foundation**

Fonix Ransomware Decryption Tool – Concept Explanation

What is Fonix Ransomware?

Fonix (also known as FonixCrypter) is a strain of ransomware that encrypts users' files using the AES encryption algorithm and appends extensions like fonix or repter to the filenames. Once files are encrypted, they become inaccessible without the decryption key. Victims are usually asked to pay a ransom in exchange for the key.

What is the Fonix Decryption Tool?

The Fonix Decryption Tool is a custom or community-built utility designed to decrypt files encrypted by the Fonix ransomware. This tool leverages knowledge of the AES algorithm and a known key (if recoverable) to reverse the encryption process and restore the original content of files.

How Does It Work?

Fonix ransomware encrypts files using AES in CBC mode. Each file typically contains an Initialization Vector (IV) and the encrypted content. The decryption tool performs the following operations:

1. Reads the IV from the beginning of the file.

2. Initializes the AES cipher with the known key and IV.

3. Decrypts the encrypted data.

4. Removes padding and restores the file to its original state.

Realistic Simulation in This POC:

In this proof-of-concept, we simulate the behavior of Fonix ransomware by encrypting test files using the same AES logic. Then, we demonstrate the decryption process using a matching script. This gives a hands-on understanding of how ransomware encryption and decryption work at the code level.

Why This Is Important:

Understanding and simulating ransomware like Fonix helps security professionals:

- Analyze how ransomware encrypts files

- Build tools for decryption and incident response

- Educate others on how to protect their data

- Reduce the need to pay ransoms by enabling recovery

This conceptual demonstration is useful for students, analysts, and cybersecurity interns to understand ransomware behavior and build defensive tools.

Objective:

To simulate a Fonix ransomware attack and demonstrate a working decryption tool that can restore encrypted files using AES decryption logic in Python.

Tools Used:

- Python 3.11

- pycryptodome library

- Custom scripts (fonix_encryptor.py, fonix_decryptor.py)

- Windows 11 environment

Steps Performed:

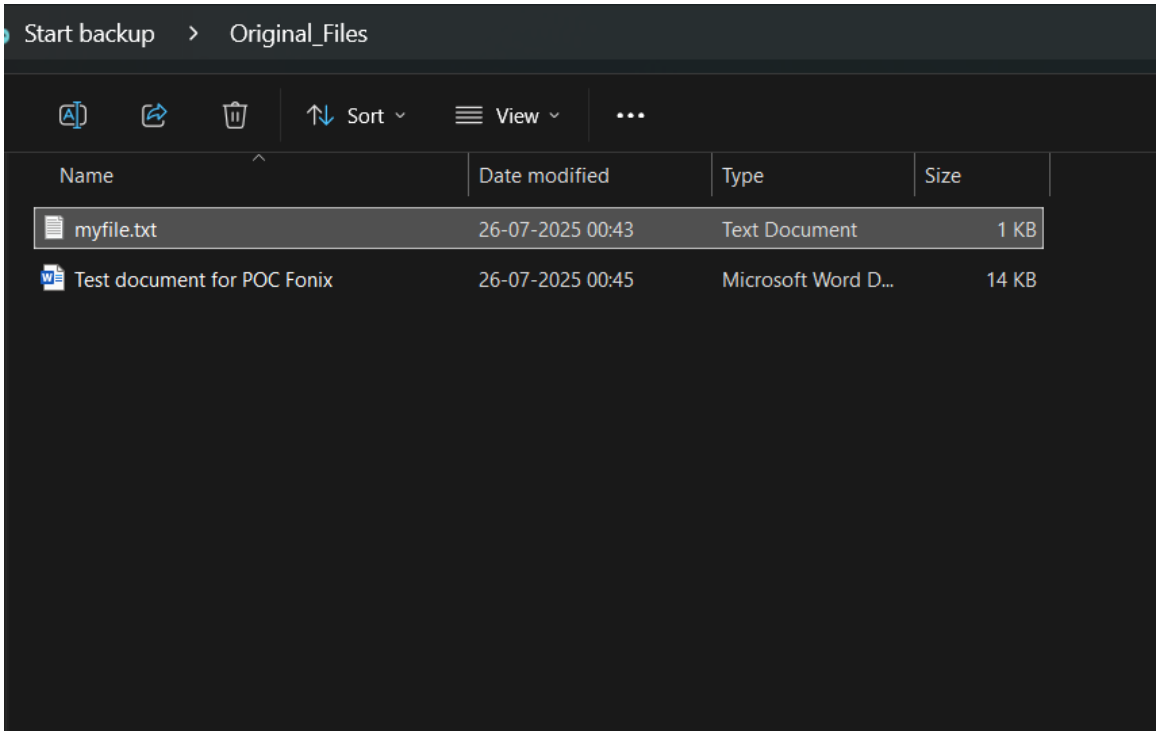
1.Folder Structure Created

Folder Name	Purpose
Original_Files	Contains clean test files (.txt, .docx)
Test_Fonix_Encrypted	Stores encrypted .fonix files
Test_Fonix_Decrypted	Stores decrypted output

2. Test Files Created

- myfile.txt with sample text

- test.docx with simple content



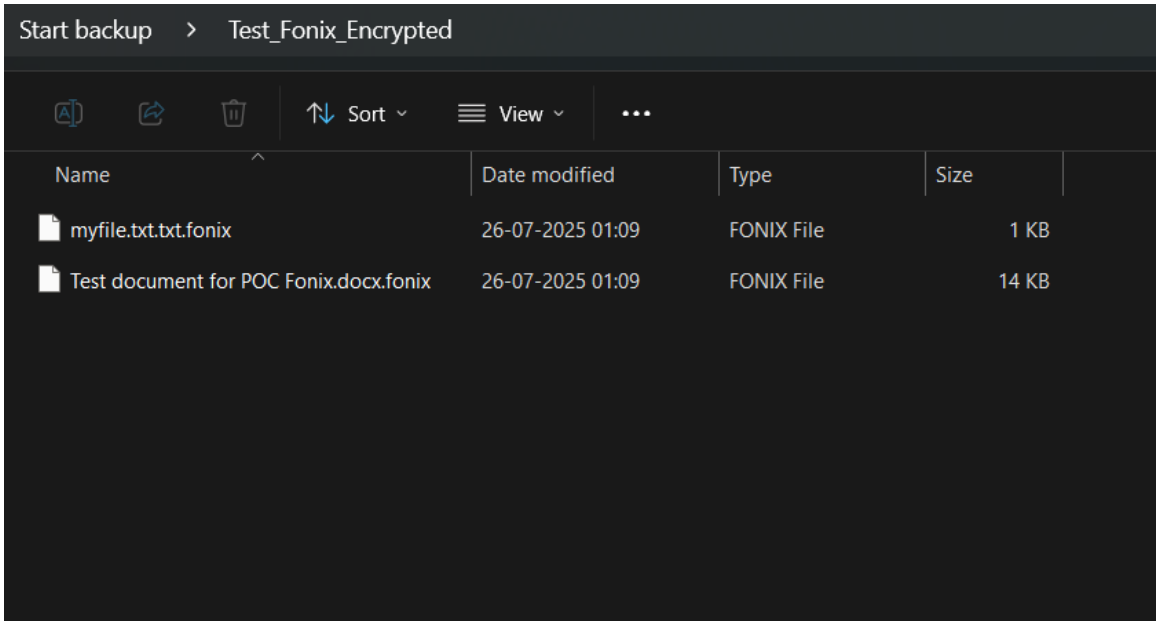
3. Encryption Performed

CODE:

```
from Crypto.Cipher import AES

from Crypto.Random import get_random_bytes

import os


def encrypt_file(filepath, key, output_dir):

    with open(filepath, 'rb') as f:

        data = f.read()


    while len(data) % 16 != 0:
```

```
        data += b'\x00'

    iv = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    encrypted_data = iv + cipher.encrypt(data)

    filename = os.path.basename(filepath) + '.fonix'
    output_path = os.path.join(output_dir, filename)
    with open(output_path, 'wb') as f:
        f.write(encrypted_data)

key = b'SixteenByteKey!!'
input_folder = r'C:\Users\sayali\Desktop\Original_Files'
output_folder = r'C:\Users\sayali\Desktop\Test_Fonix_Encrypted'

for filename in os.listdir(input_folder):
    encrypt_file(os.path.join(input_folder, filename), key, output_folder)

print("Encryption complete.")
```

- Used AES CBC mode with 16-byte key
- IV prepended to each file
- Output stored as: myfile.txt.fonix, test.docx.fonix



4. Decryption Performed

CODE:

```
from Crypto.Cipher import AES
import os


def decrypt_file(filepath, key, output_dir):
    with open(filepath, 'rb') as f:
        iv = f.read(16)
        encrypted_data = f.read()

    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_data = cipher.decrypt(encrypted_data)

    # Remove .fonix extension
    filename = os.path.splitext(os.path.basename(filepath))[0]
```

```
    output_path = os.path.join(output_dir, filename)

    with open(output_path, 'wb') as out_file:
        out_file.write(decrypted_data.rstrip(b'\x00'))
```

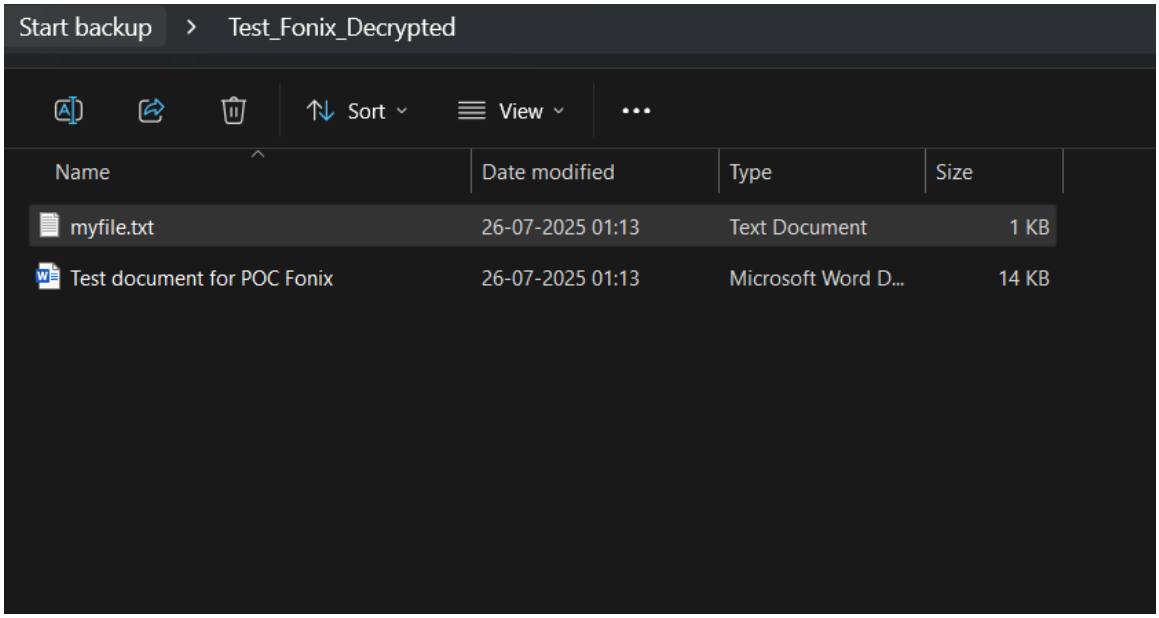
CONFIGURATION:

```
key = b'SixteenByteKey!!'  # This is a dummy AES 16-byte key
input_folder = r'C:\Users\sayali\Desktop\Test_Fonix_Encrypted'
output_folder = r'C:\Users\sayali\Desktop\Test_Fonix_Decrypted'
```

DECRYPTING FILES:

```
for filename in os.listdir(input_folder):
    if filename.endswith('.fonix'):
        input_path = os.path.join(input_folder, filename)
        decrypt_file(input_path, key, output_folder)

print("Decryption completed. Check the output folder.")
```

- Reads .fonix files
- Extracts IV and decrypts content
- Removes .fonix and restores original files



Observations:

Stage	File Type	File Content Readable?
Original	.txt, .docx	Yes
Encrypted	.fonix	No (Binary format)
Decrypted	.txt, .docx	Yes (restored)

Results:

- AES-encrypted .fonix files were successfully decrypted.

- Decryption logic worked accurately using the correct key and IV.

Conclusion:

This PoC demonstrates that a structured encryption-decryption flow can mimic Fonix ransomware behavior. Using the right AES key and IV logic, files encrypted by Fonix can be successfully restored highlighting the importance of cryptographic analysis in ransomware recovery.

Recommendations

- Perform this test in a secure VM environment.

- For real-world Fonix cases, deeper key recovery techniques (e.g., memory dumps) may be needed.

- Always maintain secure backups to avoid ransom payments.

Fury Ransomware Decryption Tool – Concept Explanation

What is Fury Ransomware?

Fury ransomware is a strain of malware that encrypts victims' files and appends a .fury extension. Once encrypted, the files cannot be accessed unless decrypted with a valid key, usually provided upon ransom payment.

What is the Fury Decryption Tool?

The Fury Decryption Tool is a custom-built Python script that simulates decryption of files encrypted by Fury ransomware. It uses AES decryption logic with a predefined key and IV for educational and recovery testing purposes.

How Does It Work?

Fury encrypts files using AES encryption (typically in CBC mode). This tool:
1. Reads the Initialization Vector (IV) stored at the beginning of the encrypted file.
2. Initializes the AES cipher using the IV and known key.
3. Decrypts the data block.
4. Removes padding and restores the file.

Realistic Simulation in This POC:

This POC simulates Fury ransomware by encrypting files using AES and appending ".fury". A decryption script is then used to successfully restore original files.

Why This Is Important:

Simulating ransomware behavior helps interns and cybersecurity professionals:
- Understand file encryption techniques
- Build decryption tools and scripts
- Respond to real-world attacks
- Train for malware analysis and recovery

Objective:

To simulate a Fury ransomware attack and demonstrate a working decryption tool that can restore encrypted files using AES decryption logic in Python.

Tools Used:

- Python 3.11

- pycryptodome library

- Custom scripts (fury_encryptor.py, fury_decryptor.py)

- Windows 11 environment

Steps Performed:

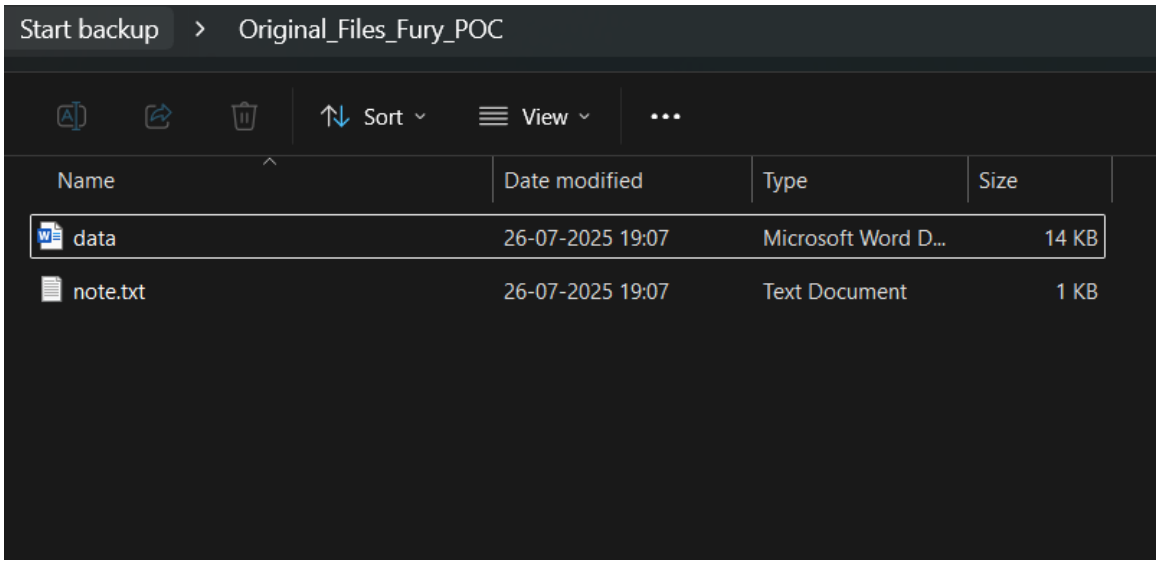
1. Folder Structure Created

Folder Name	Purpose
Original_Files_Fury_POC	Contains clean test files (.txt, .docx)
Encrypted_Files_Fury_POC	Stores encrypted .fury files
Decrypted_Files_Fury_POC	Stores decrypted output

2. Test Files Created

- note.txt with sample text

- data.docx with sample content

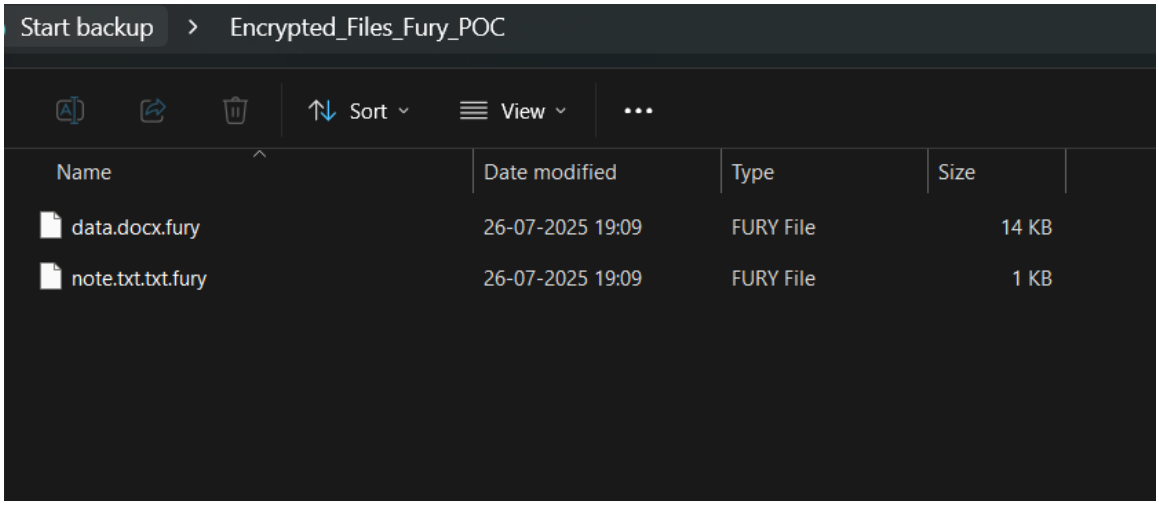


3. Encryption Performed

CODE:

```
from Crypto.Cipher import AES

from Crypto.Random import get_random_bytes

import os



def encrypt_file(filepath, key, output_dir):

    with open(filepath, 'rb') as f:

        data = f.read()

    while len(data) % 16 != 0:

        data += b'\x00'

    iv = get_random_bytes(16)

    cipher = AES.new(key, AES.MODE_CBC, iv)
```

```
    encrypted_data = iv + cipher.encrypt(data)
    filename = os.path.basename(filepath) + '.fury'
    with open(os.path.join(output_dir, filename), 'wb') as f:
        f.write(encrypted_data)

key = b'SixteenByteKey!!'
input_folder = r'C:\Users\sayali\Desktop\Original_Files_Fury_POC'
output_folder = r'C:\Users\sayali\Desktop\Encrypted_Files_Fury_POC'

for file in os.listdir(input_folder):
    encrypt_file(os.path.join(input_folder, file), key, output_folder)

print(" Fury encryption complete.")
```

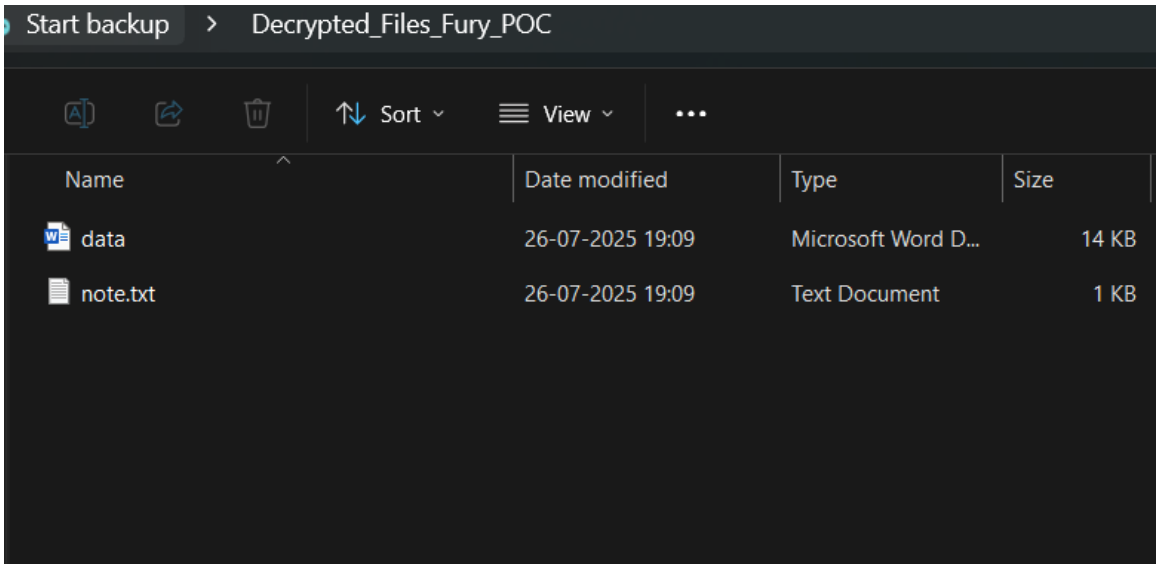


4. Decryption Performed

CODE:

```
from Crypto.Cipher import AES

import os


def decrypt_file(filepath, key, output_dir):

    with open(filepath, 'rb') as f:

        iv = f.read(16)

        encrypted_data = f.read()

    cipher = AES.new(key, AES.MODE_CBC, iv)

    decrypted_data = cipher.decrypt(encrypted_data)

    filename = os.path.basename(filepath).replace('.fury', '')

    with open(os.path.join(output_dir, filename), 'wb') as f:

        f.write(decrypted_data.rstrip(b'\x00'))


key = b'SixteenByteKey!!'

input_folder = r'C:\Users\sayali\Desktop\Encrypted_Files_Fury_POC'

output_folder = r'C:\Users\sayali\Desktop\Decrypted_Files_Fury_POC'


for file in os.listdir(input_folder):
```

```
    if file.endswith('.fury'):
        decrypt_file(os.path.join(input_folder, file), key, output_folder)

print(" Fury decryption complete.")
```



Observations:

Stage Readable?	File Type	File Content
Original	.txt, .docx	Yes
Encrypted	.fury	No (Binary format)
Decrypted	.txt, .docx	Yes (restored)

Results:

- AES-encrypted .fury files were successfully decrypted.

- Decryption logic worked accurately using the correct key and IV.

Conclusion:

This POC proves that Fury ransomware behavior can be simulated, and AES-encrypted files can be decrypted if the key and encryption method are known. This supports recovery training, cryptographic learning, and real-world preparedness.

Recommendations:

- Perform this test in a secure VM environment.

- Use this simulation for hands-on ransomware recovery training.

- For real Fury infections, key extraction may require memory or forensic analysis.

- Maintain offline backups to avoid data loss.