

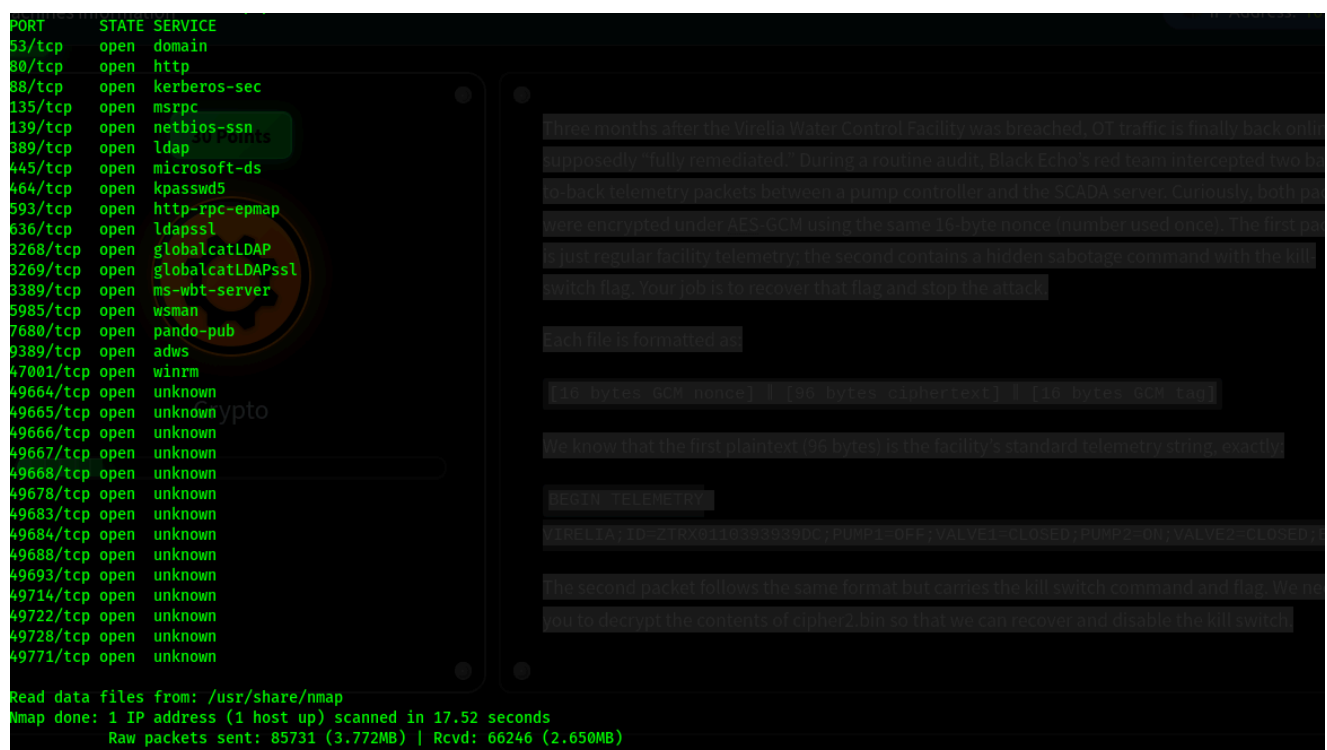
TryHackMe - Cipher

Task Overview:

The task was to analyze telemetry traffic between a SCADA server and a pump controller. The traffic was encrypted using **AES-GCM** with a shared **16-byte nonce**. The objective was to decrypt the second packet and extract the hidden sabotage command containing a flag.

Step 1: Nmap Scan

I began with a quick Nmap scan to check for open ports and services. Upon scanning, I found a variety of services including HTTP, LDAP, SMB, and others.



```
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
88/tcp    open  kerberos-sec
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
593/tcp   open  http-rpc-epmap
636/tcp   open  ldapssl
3268/tcp  open  globalcatLDAP
3269/tcp  open  globalcatLDAPssl
3389/tcp  open  ms-wbt-server
5985/tcp  open  wsman
7680/tcp  open  pando-pub
9389/tcp  open  adws
47001/tcp open  winrm
49664/tcp open  unknown
49665/tcp open  unknown
49666/tcp open  unknown
49667/tcp open  unknown
49668/tcp open  unknown
49678/tcp open  unknown
49683/tcp open  unknown
49684/tcp open  unknown
49688/tcp open  unknown
49693/tcp open  unknown
49714/tcp open  unknown
49722/tcp open  unknown
49728/tcp open  unknown
49771/tcp open  unknown

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 17.52 seconds
Raw packets sent: 85731 (3.772MB) | Rcvd: 66246 (2.650MB)
```

Three months after the Virelia Water Control Facility was breached, OT traffic is finally back online, supposedly "fully remediated." During a routine audit, Black Echo's red team intercepted two binary telemetry packets between a pump controller and the SCADA server. Curiously, both packets were encrypted under AES-GCM using the same 16-byte nonce (number used once). The first packet is just regular facility telemetry; the second contains a hidden sabotage command with the kill switch flag. Your job is to recover that flag and stop the attack.

Each file is formatted as:

[16 bytes: GCM nonce] | [96 bytes: ciphertext] | [16 bytes: GCM tag]

We know that the first plaintext (96 bytes) is the facility's standard telemetry string, exactly:

BEGIN TELEMETRY

VIRELIA: ID=2TRX9110383039DC;PUMP1=OFF;VALVE1=CLOSED;PUMP2=ON;VALVE2=CLOSED;

The second packet follows the same format but carries the kill switch command and flag. We need you to decrypt the contents of cipher2.bin so that we can recover and disable the kill switch.

Key Observations from Nmap scan:

Several services including LDAP and SMB were running on the target machine, indicating possible access points for further exploration.

Step 2: Analyzing Encrypted Telemetry Packets

I was provided with two encrypted binary files, **cipher1.bin** and **cipher2.bin**, which contained the telemetry data.

The telemetry data was split into a 16-byte nonce, a 96-byte ciphertext, and a 16-byte GCM tag.

The first packet (**cipher1.bin**) contained the standard telemetry data, while the second packet (**cipher2.bin**) carried the hidden kill switch command along with the flag.



EchoedSteams-1749197535699.zip

Completed — 2.2 KB



```
(fc0d3x_guest@kali)~[~/Downloads]
$ cd EchoedSteams-1749197535699
(fc0d3x_guest@kali)~[~/Downloads/EchoedSteams-1749197535699]
$ ls
cipher1(1).bin  'cipher2(1).bin'  __MACOSX
(fc0d3x_guest@kali)~[~/Downloads/EchoedSteams-1749197535699]
$
```

is just regular facility telemetry; the second contains a hidden sabotage command with the kill-switch flag. Your job is to recover that flag and stop the attack.

Each file is formatted as:

```
[16 bytes GCM nonce] || [96 bytes ciphertext] || [16 bytes GCM tag]
```

We know that the first plaintext (96 bytes) is the facility's standard telemetry string, exactly:

```
BEGIN TELEMETRY
VIRELIA;ID=ZTRX0110393939DC;PUMP1=OFF;VALVE1=CLOSED;PUMP2=ON;VALVE2=CLOSED;E
```

Step 3: Decrypting the Data

To decrypt the second packet, I wrote a Python script that:

- Reads both ciphertext files.

- Extracts the 16-byte nonce, 96-byte ciphertext, and the GCM tag.

- XORs the known plaintext (from the first packet) with the ciphertext to generate the keystream.

- Decrypts the second ciphertext using the keystream.

Python Code Used for Decryption:

```
from Crypto.Cipher import AES
import binascii

# Specify the correct paths to your files
cipher1_path = "/home/fc0d3x_guest/Downloads/EchoedSteams-1749197535699/cipher1(1).bin"
cipher2_path = "/home/fc0d3x_guest/Downloads/EchoedSteams-1749197535699/cipher2(1).bin"

# Read the cipher1 and cipher2 binary files
with open(cipher1_path, "rb") as f1, open(cipher2_path, "rb") as f2:
    cipher1 = f1.read()
    cipher2 = f2.read()

# Split the data into nonce, ciphertext, and tag (16-byte nonce, 96-byte ciphertext, 16-byte tag)
nonce1 = cipher1[:16]
nonce2 = cipher2[:16]

ciphertext1 = cipher1[16:112] # 96 bytes ciphertext
tag1 = cipher1[112:] # 16 bytes tag

ciphertext2 = cipher2[16:112] # 96 bytes ciphertext
tag2 = cipher2[112:] # 16 bytes tag

# Known plaintext from cipher1
plaintext1 = b"BEGIN TELEMETRY VIRELIA;ID=ZTRX0110393939DC;PUMP1=OFF;VALVE1=CLOSED;PUMP2=ON;VALVE2=CLOSED;END;"

# Step 1: Compute the keystream by XORing ciphertext1 and plaintext1
keystream = bytes([c1 ^ p1 for c1, p1 in zip(ciphertext1, plaintext1)])

# Step 2: Decrypt the second ciphertext by XORing ciphertext2 with the keystream
plaintext2 = bytes([c2 ^ k for c2, k in zip(ciphertext2, keystream)])

# Print the result, which should be the kill switch command and flag
print("Decrypted plaintext:", plaintext2.decode())
```

Step 4: Revealing the Flag

After running the script, I decrypted the second packet successfully, revealing the kill switch command and flag:

```
(fc0d3x_guest@kali)-[~]  
└─$ python3 decrypt.py  
Decrypted plaintext: BEGIN TELEMETRY VIRELIA;ID=TRX0110393939DC;PUMP=ON;VALVE=OPEN;TEMP=1.0;KILL=THM{Echo_Telemetry} and disable the kill switch.
```

THM Echo_Telemetry

Conclusion:

The task was a great exercise in working with AES-GCM encryption and learning to decrypt real-world industrial telemetry traffic. By using a known plaintext attack combined with Python, I was able to recover the hidden flag and disable the kill switch in the second telemetry packet.