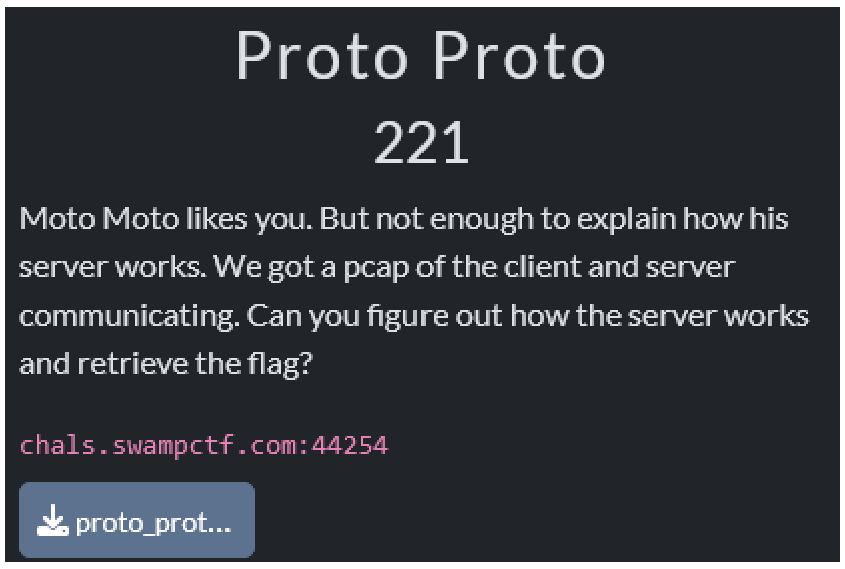
Forensics - Proto Proto



Proto Proto

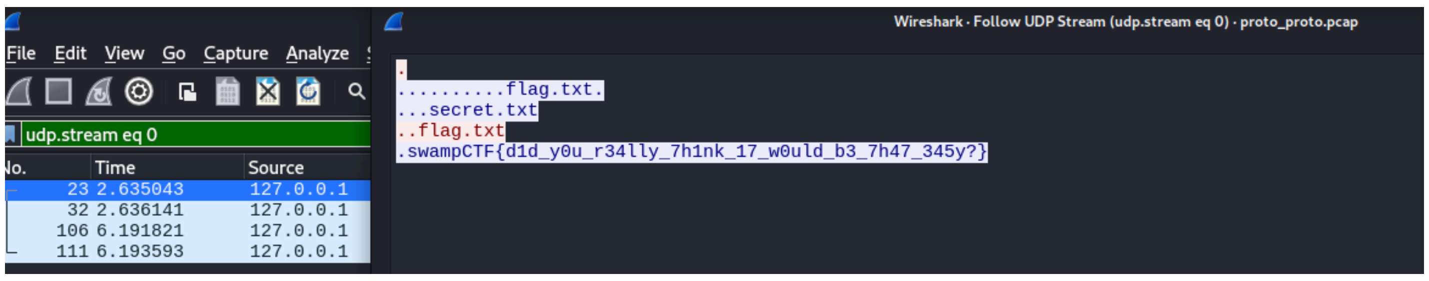
221

Moto Moto likes you. But not enough to explain how his server works. We got a pcap of the client and server communicating. Can you figure out how the server works and retrieve the flag?

chals.swampctf.com:44254

Connection via netcat is unsuccessful, but there is a ping to the server, so we will find another way to connect.

Having opened and analyzed the pcap file, we can notice interesting UDP traffic:



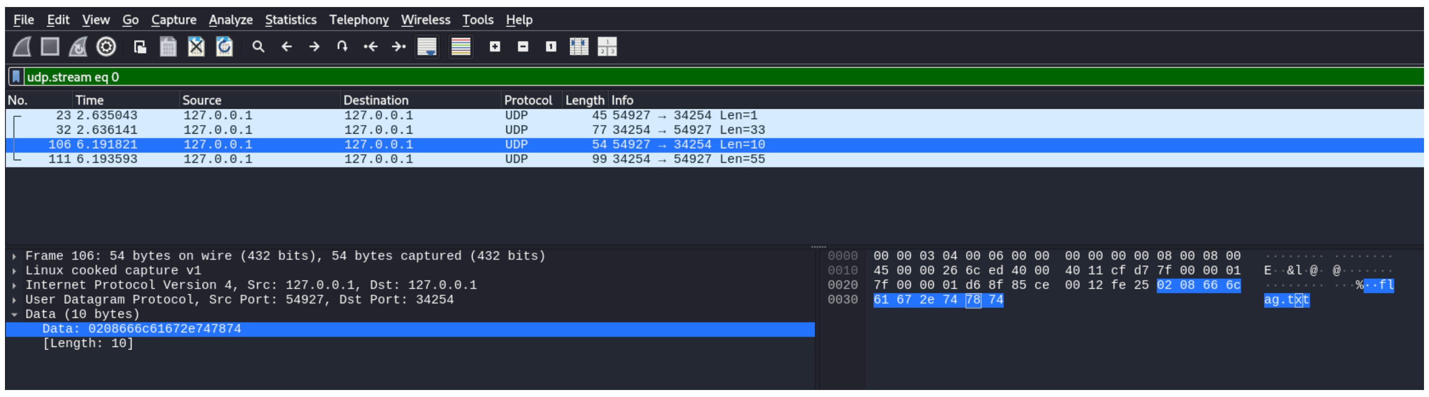
File Edit View Go Capture Analyze
Wireshark · Follow UDP Stream (udp.stream eq 0) · proto_proto.pcap

```
udp.stream eq 0
No.  Time      Source
 23 2.635043   127.0.0.1
 32 2.636141   127.0.0.1
106 6.191821   127.0.0.1
111 6.193593   127.0.0.1
```

```
.
..........flag.txt.
...secret.txt
..flag.txt
.swampCTF{d1d_y0u_r34lly_7h1nk_17_w0uld_b3_7h47_345y?}
```

We see that the client sent the flag.txt command to the server and received the flag.

Let's find it in UDP traffic:



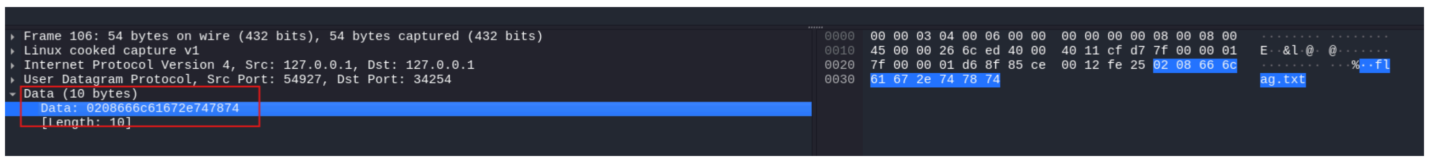
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
udp.stream eq 0
No.  Time      Source     Destination Protocol Length Info
 23 2.635043   127.0.0.1  127.0.0.1   UDP      45 54927 → 34254 Len=1
 32 2.636141   127.0.0.1  127.0.0.1   UDP      77 34254 → 54927 Len=33
106 6.191821   127.0.0.1  127.0.0.1   UDP      54 54927 → 34254 Len=10
111 6.193593   127.0.0.1  127.0.0.1   UDP      99 34254 → 54927 Len=55
```

```
› Frame 106: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
› Linux cooked capture v1
› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
› User Datagram Protocol, Src Port: 54927, Dst Port: 34254
⌄ Data (10 bytes)
    Data: 0208666c61672e747874
    [Length: 10]
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 08 00 08 00   ········ ········
0010  45 00 00 26 6c ed 40 00  40 11 cf d7 7f 00 00 01   E··&l·@· @·······
0020  7f 00 00 01 d6 8f 85 ce  00 12 fe 25 02 08 66 6c   ········ ···%··fl
0030  61 67 2e 74 78 74                                  ag.txt
```

Let's also try to send "flag.txt" to the server using UDP. We'll send the value of "flag.txt" to the server as bytes. We'll copy the bytes from the pcap file:



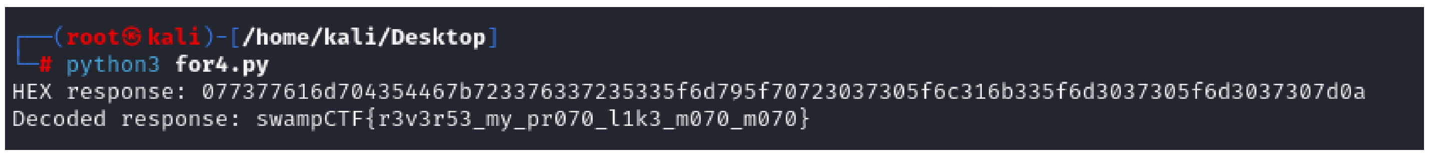
```
› Frame 106: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
› Linux cooked capture v1
› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
› User Datagram Protocol, Src Port: 54927, Dst Port: 34254
⌄ Data (10 bytes)
    Data: 0208666c61672e747874
    [Length: 10]
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 08 00 08 00   ········ ········
0010  45 00 00 26 6c ed 40 00  40 11 cf d7 7f 00 00 01   E··&l·@· @·······
0020  7f 00 00 01 d6 8f 85 ce  00 12 fe 25 02 08 66 6c   ········ ···%··fl
0030  61 67 2e 74 78 74                                  ag.txt
```

Our code will look like this (with comments for understanding):

```
import socket  # Import the socket library to handle network communication

# Define the server address and port to connect to
server = "chals.swampctf.com"  # The hostname of the CTF challenge server
port = 44254  # The port number specified in the challenge (UDP port)

# Define the payload to request "flag.txt" from the server
# - b"\x02" is the command code (possibly indicating a file request)
# - b"\x08" is the length of the filename (8 bytes for "flag.txt")
# - b"\x66\x6c\x61\x67\x2e\x74\x78\x74" is "flag.txt" in HEX (ASCII encoding)
payload = b"\x02\x08\x66\x6c\x61\x67\x2e\x74\x78\x74"

# Create a UDP socket object
# - socket.AF_INET specifies the IPv4 address family
# - socket.SOCK_DGRAM specifies UDP (datagram) protocol
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Send the payload to the server via UDP
# - sendto() sends the data to the specified (server, port) tuple
s.sendto(payload, (server, port))

# Receive a response from the server
# - recvfrom(4096) waits for up to 4096 bytes of data and returns the response and sender's address
response, addr = s.recvfrom(4096)

# Print the response in hexadecimal format for raw byte analysis
print("HEX response:", response.hex())

# Attempt to decode the response as ASCII text, ignoring any decoding errors
# - errors='ignore' ensures the script doesn't crash if non-ASCII bytes are present
print("Decoded response:", response.decode(errors='ignore'))

# Close the socket to free up resources
s.close()
```

Let's run this script:



```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 for4.py
HEX response: 077377616d704354467b7233763337235335f6d795f70723037305f6c316b335f6d3037305f6d3037307d0a
Decoded response: swampCTF{r3v3r53_my_pr070_l1k3_m070_m070}
```

We received the flag: **swampCTF{r3v3r53_my_pr070_l1k3_m070_m070}**