Challenge 1 – File System and Hidden Flag

Teacher and Student Guide  
Author: Sarib Ul Haq

# Teacher Guide

Purpose:  
Teach students filesystem navigation, locating hidden files, directory traversal, and Base64 decoding. This version places part1 in level1 and part2 (hidden) in level3 named as config.jason.

## Setup Summary:

1. Create the following directory structure under the challenger user’s home folder:  
  
/home/challenger/ctf\_file\_challenge/  
├── level1/  
│ ├── .hint.txt  
│ ├── part1.txt  
│ └── level2/  
│ └── level3/  
│ └── .config.jason  
  
2. Add a short message in .hint.txt:  
 The flag is split into two Base64 encoded parts. Find and decode both to reveal it.  
  
3. Prepare the flag:  
 - Full flag: flag{filesystem\_hidden\_flag}  
 - Split and encode each part:  
 echo "flag{filesystem\_" | base64 > part1.txt  
 echo "hidden\_flag}" | base64 > .config.jason  
  
4. Permissions and ownership:  
 sudo chown -R challenger:challenger /home/challenger/ctf\_file\_challenge  
 sudo chmod -R 700 /home/challenger/ctf\_file\_challenge  
  
 The .hint.txt and .config.jason remain hidden by default (dotfiles).

## Verification (for teacher):

1. Log in as the challenger user.  
2. Navigate to the challenge directory:  
 cd ~/ctf\_file\_challenge  
 ls  
 ls -a  
 Verify that .hint.txt appears only with -a.  
  
3. Read the hint and navigate deeper:  
 cat level1/.hint.txt  
 cd level1/level2/level3  
 ls -a (verify .config.jason is hidden)  
  
4. Decode the files:  
 base64 -d ~/ctf\_file\_challenge/level1/part1.txt  
 base64 -d ~/ctf\_file\_challenge/level1/level2/level3/.config.jason  
 Combine both parts to form:  
 flag{filesystem\_hidden\_flag}  
  
Expected Outcome:  
Students will use ls -a to discover hidden files, navigate multiple directory levels, decode Base64, and combine the outputs to reconstruct the full flag.

# Student Guide

Objective:  
Find and reconstruct a split Base64-encoded flag hidden across nested folders, where one file is hidden.

## Scenario:

Inside your home directory, there’s a folder named ctf\_file\_challenge. It contains multiple subfolders (level1, level2, level3). Some files are hidden. The flag is split into two Base64-encoded parts: the first is in level1, and the second is hidden in level3 as config.jason.

## Steps to Solve:

1. Move into the challenge folder:  
 cd ~/ctf\_file\_challenge  
  
2. Check for visible and hidden files:  
 ls  
 ls -a  
 You will see a hidden file named .hint.txt.  
  
3. Read the hint:  
 cat level1/.hint.txt  
  
4. Go into the nested directories:  
 cd level1/level2/level3  
 ls  
 ls -a (to reveal the hidden .config.jason file)  
  
5. Decode both parts:  
 base64 -d ~/ctf\_file\_challenge/level1/part1.txt  
 base64 -d ~/ctf\_file\_challenge/level1/level2/level3/.config.jason  
  
6. Combine the outputs from both decoded files to reveal the complete flag:  
 flag{filesystem\_hidden\_flag}

## Learning Outcome:

- Use ls -a to view hidden files.  
- Explore multi-level directory structures.  
- Decode Base64 content and combine data.  
- Apply investigative thinking to locate hidden information.

Challenge 2 – Web (QR Code)

Teacher and Student Guide  
Author: Sarib Ul Haq  
Date: 16 October 2025

# Teacher Guide

Purpose:  
Teach students basic web enumeration using dirb, identifying hidden folders, and decoding a QR code that contains a Base64-encoded flag.

## Setup Summary:

1. Create the following directory structure under the challenger user’s home folder:  
  
/home/challenger/ctf\_web\_challenge/  
├── index.html  
└── .secret\_folder/  
 └── qr.png  
  
2. index.html (basic homepage):  
<html>  
<body>  
<h2>Welcome to CTF Web Challenge</h2>  
<p>There is nothing interesting here ... or is there?</p>  
</body>  
</html>  
  
3. QR Code (qr.png):  
 - Full flag: flag{web\_hidden\_qr\_flag}  
 - Encode flag in Base64:  
 echo "flag{web\_hidden\_qr\_flag}" | base64  
 - Copy the encoded string and generate the QR image:  
 qrencode -o qr.png "<Base64EncodedFlag>"  
 - Place qr.png inside .secret\_folder.  
  
4. Permissions:  
 chmod 755 /home/challenger/ctf\_web\_challenge/.secret\_folder  
  
 The leading dot keeps it hidden by default.  
  
5. Starting the challenge:  
 Students run:  
 cd ~/ctf\_web\_challenge  
 python3 -m http.server 8000  
  
 Then open the browser at http://127.0.0.1:8000

## Verification (for teacher):

1. Run the server yourself and open http://127.0.0.1:8000.  
 The homepage (index.html) should display normally.  
2. Run a directory brute-force scan:  
 dirb http://127.0.0.1:8000 /usr/share/dirb/wordlists/common.txt  
  
 The scan should reveal the path /.secret\_folder/.  
3. Visit http://127.0.0.1:8000/.secret\_folder/qr.png and confirm the QR image loads.  
4. Scan the QR image (e.g., zbarimg qr.png) to get a Base64 string.  
5. Decode it:  
 echo "<Base64StringFromQR>" | base64 -d  
  
 The result should display the flag:  
 flag{web\_hidden\_qr\_flag}  
  
Expected Outcome:  
Students successfully enumerate hidden web directories, locate the QR image, and decode the Base64 string from the QR code to reveal the flag.

# Student Guide

Objective:  
Find and decode a hidden QR code to reveal the Base64-encoded flag.

## Scenario:

You have a folder named ctf\_web\_challenge. When served with Python’s built-in web server, it displays a simple web page. Somewhere inside this web project is a hidden folder containing a QR code image. Your task is to find the folder, locate the QR image, and decode the Base64 message to get the flag.

## Steps to Solve:

1. Start the challenge server:  
 cd ~/ctf\_web\_challenge  
 python3 -m http.server 8000  
  
 Open your browser and go to: http://127.0.0.1:8000  
  
2. The homepage looks plain. Start enumerating directories:  
 dirb http://127.0.0.1:8000 /usr/share/dirb/wordlists/common.txt  
  
 This will reveal a hidden folder (e.g., .secret\_folder).  
  
3. Visit the discovered path in your browser:  
 http://127.0.0.1:8000/.secret\_folder/qr.png  
  
4. Download or open qr.png and scan it using:  
 zbarimg qr.png  
  
 The result will show a Base64-encoded string.  
  
5. Decode it to get the flag:  
 echo "<Base64String>" | base64 -d  
  
 Example output:  
 flag{web\_hidden\_qr\_flag}

## Learning Outcome:

- Learn to enumerate web directories using dirb.  
- Understand how hidden folders can contain important files.  
- Decode Base64 data embedded in a QR code.  
- Apply basic web reconnaissance and decoding skills.

Challenge 3 – PCAP Network Forensics

Teacher and Student Guide  
Author: Sarib Ul Haq  
Date: 16 October 2025

This document contains the Teacher and Student guides for Challenge 3: a PCAP-based network forensics challenge. Students must analyze a provided packet capture (challenge.pcap), locate a TCP stream carrying a Base64-encoded flag, reassemble the payload if split across packets, and decode the flag.

# Teacher Guide

Purpose:  
Teach students how to analyze network traffic, identify suspicious flows, reassemble TCP streams, and decode embedded data (e.g., Base64).

## Setup Summary:

1. Create a PCAP file (challenge.pcap) containing a TCP session that carries the flag text encoded in Base64. Use a non-standard port for the session (example: 4444) to encourage filtering.  
  
2. Example construction methods:  
 - Use netcat to send the payload between two local addresses and capture with tcpdump:  
 echo "<Base64Flag>" | nc -l 4444 &  
 nc 127.0.0.1 4444 < payload.txt  
 tcpdump -i lo -w challenge.pcap tcp port 4444  
  
 - Or craft the PCAP with scapy or merge legitimate traffic with the payload to create decoys.  
  
3. Payload formatting tips:  
 - Base64-encode the full flag (e.g., flag{network\_forensics\_hard}) and place it in one TCP stream.  
 - Optionally split the Base64 string across multiple packets to require reassembly.  
 - Add decoy traffic on other ports to increase difficulty.  
  
4. Place challenge.pcap where students can access it (e.g., /home/challenger/challenge.pcap) or host a download link.

## Verification (for teacher):

1. Open challenge.pcap in Wireshark.  
2. Apply a filter for the chosen port (example):  
 tcp.port == 4444  
3. Identify the TCP session of interest and use Follow → TCP Stream to view the ASCII payload.  
4. If the Base64 string is split across packets, ensure the Follow Stream output shows the reconstructed data in correct order.  
5. Decode the Base64 payload:  
 echo "<Base64String>" | base64 -d  
 The result should reveal the flag, e.g., flag{network\_forensics\_hard}.  
  
Expected Outcome:  
Students will demonstrate ability to filter PCAPs, reassemble streams, extract ASCII payloads, and decode Base64 to recover the flag.

# Student Guide

Objective:  
Extract and decode a Base64-encoded flag embedded in a TCP stream inside a provided PCAP file.

## Scenario:

You are provided with a packet capture file (challenge.pcap). The flag has been transmitted over the network inside a TCP session on a designated port (for example, 4444). The Base64-encoded data may be split across multiple packets and mixed with decoy traffic.

## Steps to Solve:

1. Open the PCAP in Wireshark:  
 File → Open → challenge.pcap  
  
2. Look for suspicious flows or unusual ports. Apply a filter to narrow candidates (example):  
 tcp.port == 4444  
  
3. Identify the likely TCP session and right-click a packet → Follow → TCP Stream.  
 - Choose the ASCII view to see readable payload.  
 - Save or copy the stream output if needed.  
  
4. If the Base64 string is split across packets, use the Follow Stream output or tshark to reconstruct the raw ASCII payload in order.  
 Example tshark command to show follow-tcp ascii for port 4444:  
 tshark -r challenge.pcap -q -z follow,tcp,ascii,4444  
  
5. Extract the Base64 blob, remove any non-data characters, and decode:  
 echo "<Base64String>" | base64 -d  
  
6. The decoded output is the flag (for example, flag{network\_forensics\_hard}).

## Tips and Tools:

- Wireshark: use display filters and Follow → TCP Stream.  
- tshark: for command-line stream extraction and automation.  
- tcpflow / tcpslice: alternative tools to extract flows.  
- strings/xxd: quick checks for readable ASCII inside PCAPs.  
- Keep an eye on packet order and retransmissions; use sequence numbers if reassembly seems incorrect.

## Learning Outcome:

- Identify suspicious network traffic using display filters.  
- Reassemble TCP streams that may be fragmented across packets.  
- Extract, clean, and decode embedded data to recover artifacts.  
- Practice documenting findings and the steps taken to extract evidence.