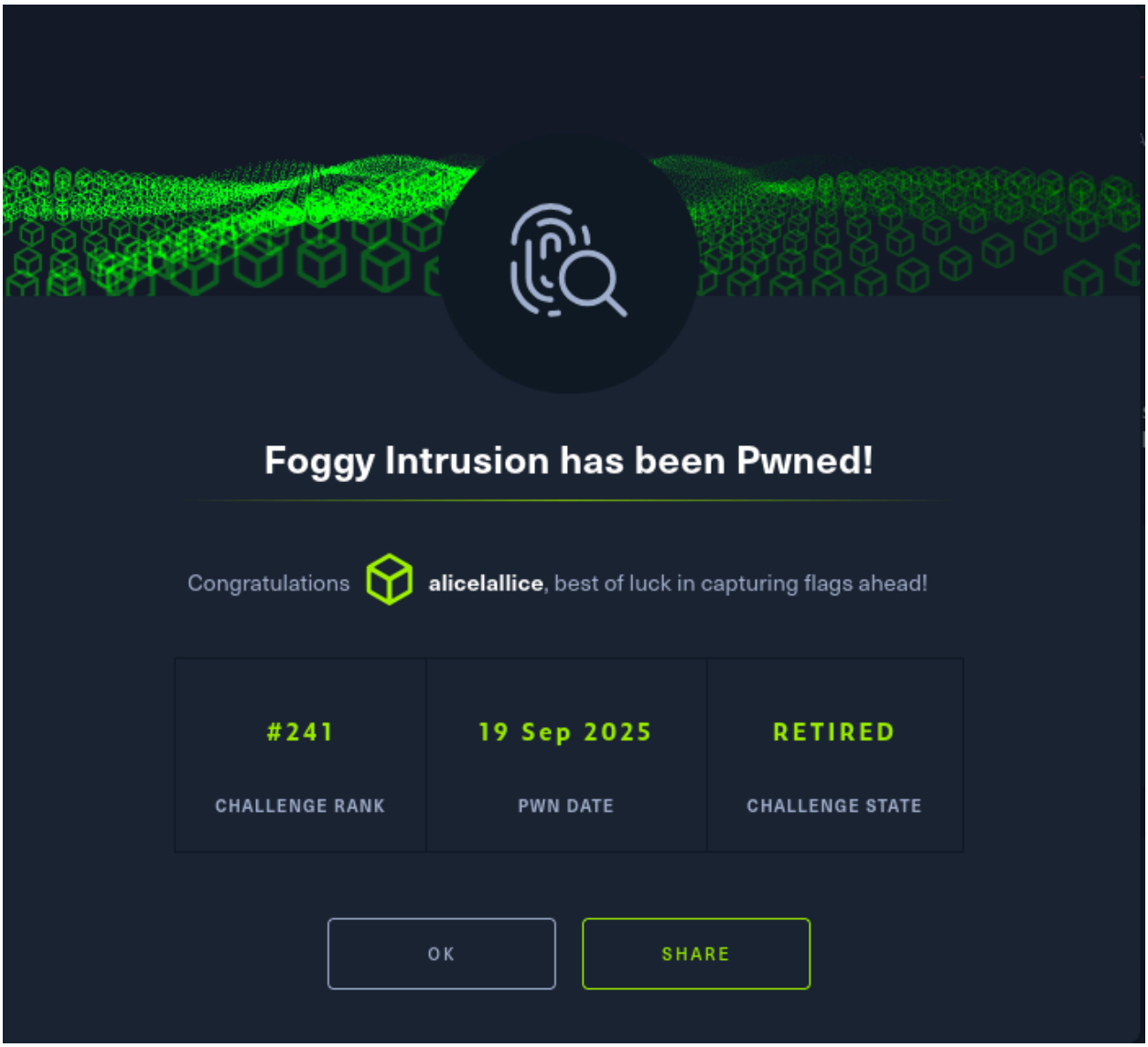
Foggy Intrusion

Types	forensic
CTF	HTB



TL;DR / Findings

- Attacker IP: **10.10.10.14**

- Victim IP: **10.10.10.13** (webserver `halloweencorp.htb`)
- Vector: HTTP POST that uploaded/ran PHP which executed **base64 → PowerShell** commands on the Windows host; server returned compressed (Deflate) + base64 outputs.
- Artifact with flag: `config.php` extracted from webserver; the DB password / config contained the flag:

```
HTB{f06_d154pp34r3d_4nd_fl46_w4s_f0und!}
```

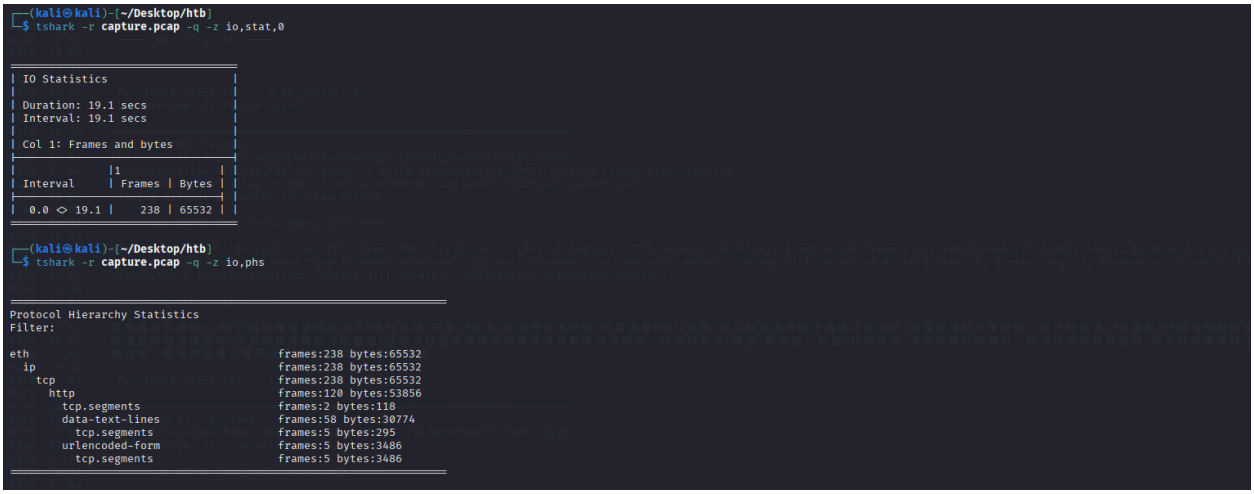
Full step-by-step (reproducible commands + explanation)

> Note: run these in your Kali shell in the directory that contains capture.pcap.

0) Quick pcap summary

Command (what it does):

```
tshark -r capture.pcap -q -z io,stat,0
tshark -r capture.pcap -q -z io,phs
```

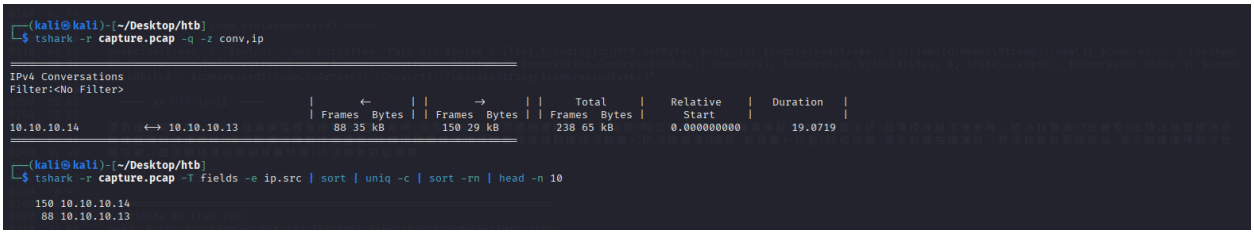


quick view of total packets, capture duration and the Protocol Hierarchy (shows HTTP heavy traffic)

238 packets, mostly TCP/HTTP between two hosts.

1) Find top conversation(s) (who talked to whom)

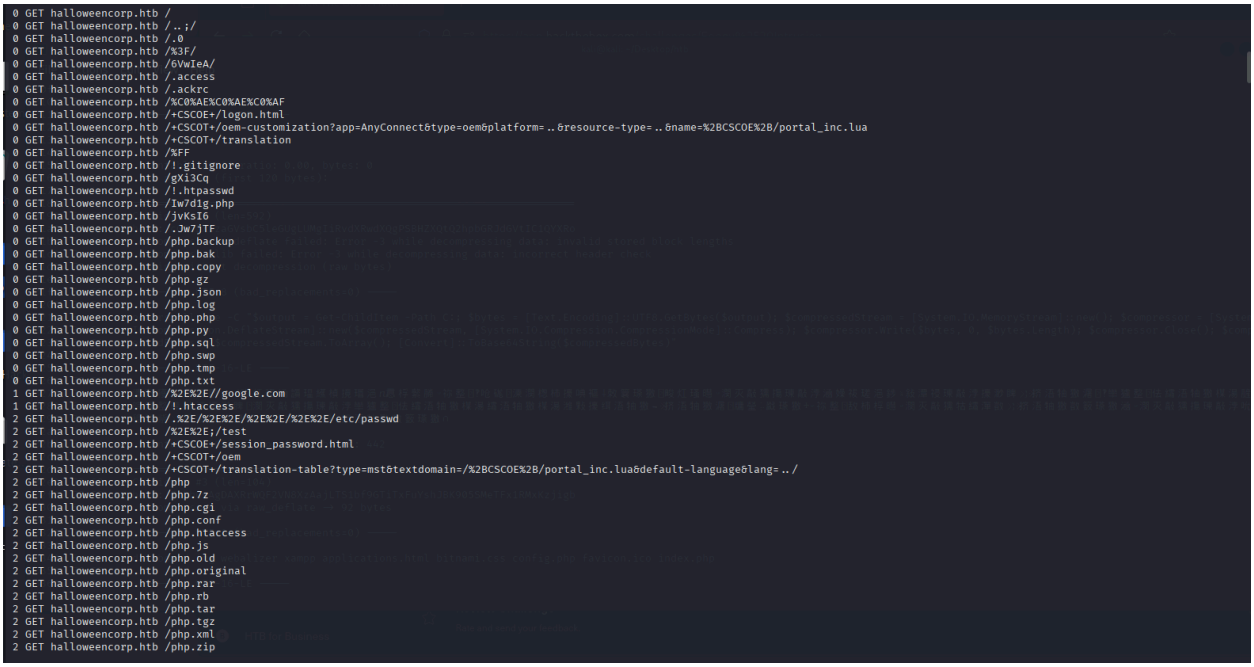
```
tshark -r capture.pcap -q -z conv,ip
tshark -r capture.pcap -T fields -e ip.src | sort | uniq -c | sort -rn | head -n 10
```



identifies attacker and victim IPs (top talkers). Result showed `10.10.10.14 ↔ 10.10.10.13`.

Enumerate HTTP requests (find GET/POSTs and stream indices)

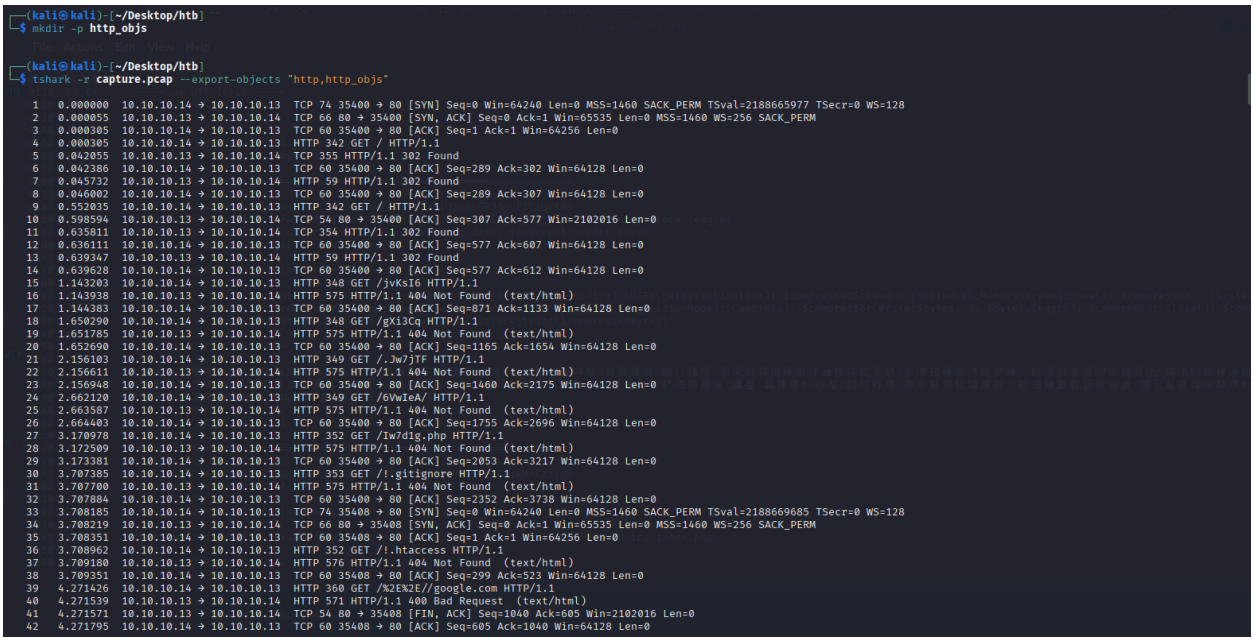
```
tshark -r capture.pcap -Y http.request -T fields -e tcp.stream -e http.request.method -e http.host -e http.request.uri | sed 's/\t/ /g' | sort -u | sed -n '1,200p'
```



gives `tcp.stream` numbers per HTTP request — critical for “Follow TCP stream” later. You saw many GETs probing for backups (php.7z, php.zip, php.bak) and a POST with suspicious URI: `/?...allow_url_include=1...auto_prepend_file=php://input`.

3) Export HTTP objects / files

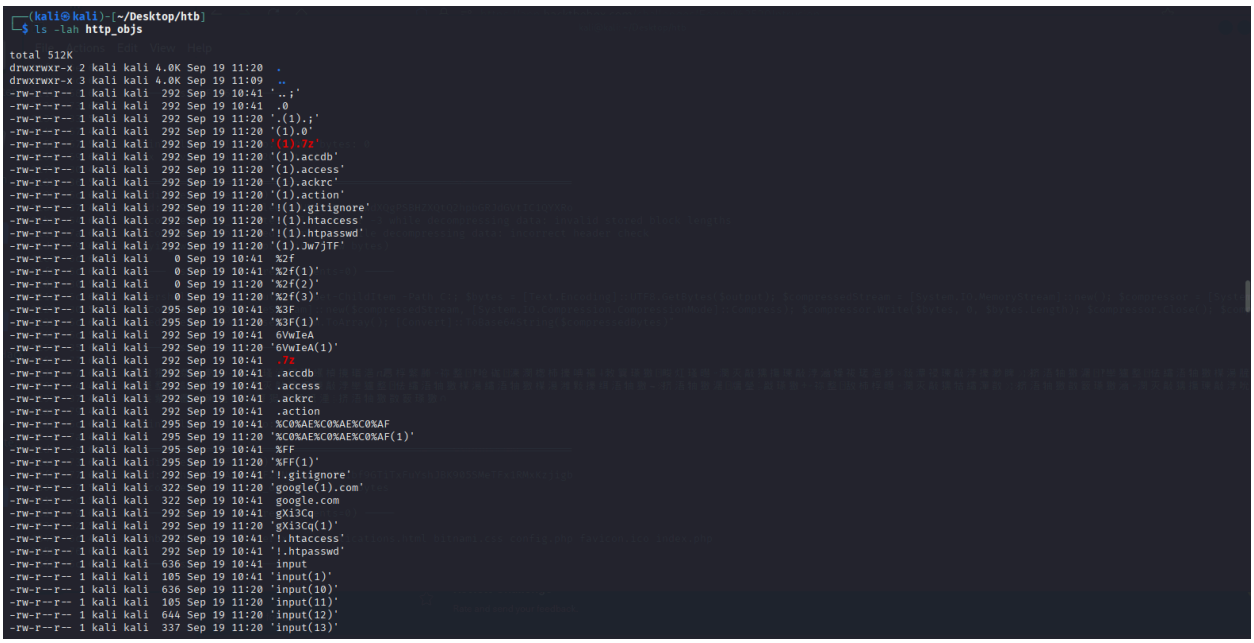
```
mkdir -p http_objs
tshark -r capture.pcap --export-objects "http,http_objs"
ls -lah http_objs
```



```
┌──(kali㉿kali)-[~/Desktop/htb]
└─$ mkdir -p http_objs

┌──(kali㉿kali)-[~/Desktop/htb]
└─$ tshark -r capture.pcap --export-objects "http,http_objs"

   1   0.000000  10.10.10.14 → 10.10.10.13  TCP 74 35400 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2188665977 TSecr=0 WS=128
   2   0.000055  10.10.10.13 → 10.10.10.14  TCP 66 80 → 35400 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
   3   0.000305  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
   4   0.000305  10.10.10.14 → 10.10.10.13  HTTP 342 GET / HTTP/1.1
   5   0.042055  10.10.10.13 → 10.10.10.14  TCP 355 HTTP/1.1 302 Found
   6   0.042386  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=289 Ack=302 Win=64128 Len=0
   7   0.045732  10.10.10.13 → 10.10.10.14  HTTP 59 HTTP/1.1 302 Found
   8   0.046002  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=289 Ack=307 Win=64128 Len=0
   9   0.552035  10.10.10.14 → 10.10.10.13  HTTP 342 GET / HTTP/1.1
  10   0.598594  10.10.10.13 → 10.10.10.14  TCP 54 80 → 35400 [ACK] Seq=307 Ack=577 Win=2102016 Len=0
  11   0.635811  10.10.10.13 → 10.10.10.14  TCP 354 HTTP/1.1 302 Found
  12   0.636111  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=577 Ack=607 Win=64128 Len=0
  13   0.639347  10.10.10.13 → 10.10.10.14  HTTP 59 HTTP/1.1 302 Found
  14   0.639628  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=577 Ack=612 Win=64128 Len=0
  15   1.143203  10.10.10.14 → 10.10.10.13  HTTP 348 GET /jvKsI6 HTTP/1.1
  16   1.143938  10.10.10.13 → 10.10.10.14  HTTP 575 HTTP/1.1 404 Not Found  (text/html)
  17   1.144383  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=871 Ack=1133 Win=64128 Len=0
  18   1.650290  10.10.10.14 → 10.10.10.13  HTTP 348 GET /gXi3Cq HTTP/1.1
  19   1.651785  10.10.10.13 → 10.10.10.14  HTTP 575 HTTP/1.1 404 Not Found  (text/html)
  20   1.652690  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=1165 Ack=1654 Win=64128 Len=0
  21   2.156103  10.10.10.14 → 10.10.10.13  HTTP 349 GET /.Jw7jTF HTTP/1.1
  22   2.156611  10.10.10.13 → 10.10.10.14  HTTP 575 HTTP/1.1 404 Not Found  (text/html)
  23   2.156948  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=1460 Ack=2175 Win=64128 Len=0
  24   2.662120  10.10.10.14 → 10.10.10.13  HTTP 349 GET /6VwIeA/ HTTP/1.1
  25   2.663587  10.10.10.13 → 10.10.10.14  HTTP 575 HTTP/1.1 404 Not Found  (text/html)
  26   2.664403  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=1755 Ack=2696 Win=64128 Len=0
  27   3.170978  10.10.10.14 → 10.10.10.13  HTTP 352 GET /Iw7dig.php HTTP/1.1
  28   3.172509  10.10.10.13 → 10.10.10.14  HTTP 575 HTTP/1.1 404 Not Found  (text/html)
  29   3.173381  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=2053 Ack=3217 Win=64128 Len=0
  30   3.707385  10.10.10.14 → 10.10.10.13  HTTP 353 GET /!.gitignore HTTP/1.1
  31   3.707700  10.10.10.13 → 10.10.10.14  HTTP 575 HTTP/1.1 404 Not Found  (text/html)
  32   3.707884  10.10.10.14 → 10.10.10.13  TCP 60 35400 → 80 [ACK] Seq=2352 Ack=3738 Win=64128 Len=0
  33   3.708185  10.10.10.14 → 10.10.10.13  TCP 74 35408 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2188669685 TSecr=0 WS=128
  34   3.708219  10.10.10.13 → 10.10.10.14  TCP 66 80 → 35408 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
  35   3.708351  10.10.10.14 → 10.10.10.13  TCP 60 35408 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
  36   3.708962  10.10.10.14 → 10.10.10.13  HTTP 352 GET /!.htaccess HTTP/1.1
  37   3.709180  10.10.10.13 → 10.10.10.14  HTTP 576 HTTP/1.1 404 Not Found  (text/html)
  38   3.709351  10.10.10.14 → 10.10.10.13  TCP 60 35408 → 80 [ACK] Seq=299 Ack=523 Win=64128 Len=0
  39   4.271426  10.10.10.14 → 10.10.10.13  HTTP 360 GET /%2E%2E//google.com HTTP/1.1
  40   4.271539  10.10.10.13 → 10.10.10.14  HTTP 571 HTTP/1.1 400 Bad Request  (text/html)
  41   4.271571  10.10.10.13 → 10.10.10.14  TCP 54 80 → 35408 [FIN, ACK] Seq=1040 Ack=605 Win=2102016 Len=0
  42   4.271795  10.10.10.14 → 10.10.10.13  TCP 60 35408 → 80 [ACK] Seq=605 Ack=1040 Win=64128 Len=0
```



```
┌──(kali㉿kali)-[~/Desktop/htb]
└─$ ls -lah http_objs

total 512K
drwxrwxr-x 2 kali kali 4.0K Sep 19 11:20  .
drwxrwxr-x 3 kali kali 4.0K Sep 19 11:09  ..
-rw-r--r-- 1 kali kali  292 Sep 19 10:41 '..;'
-rw-r--r-- 1 kali kali  292 Sep 19 10:41  .0
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '.(1).;'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '(1).0'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '(1).7z'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '(1).accdb'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '(1).access'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '(1).ackrc'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '(1).action'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '!(1).gitignore'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '!(1).htaccess'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '!(1).htpasswd'
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '(1).Jw7jTF'
-rw-r--r-- 1 kali kali    0 Sep 19 10:41  %2f
-rw-r--r-- 1 kali kali    0 Sep 19 10:41 '%2f(1)'
-rw-r--r-- 1 kali kali    0 Sep 19 11:20 '%2f(2)'
-rw-r--r-- 1 kali kali    0 Sep 19 11:20 '%2f(3)'
-rw-r--r-- 1 kali kali  295 Sep 19 10:41  %3F
-rw-r--r-- 1 kali kali  295 Sep 19 11:20 '%3F(1)'
-rw-r--r-- 1 kali kali  292 Sep 19 10:41  6VwIeA
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 '6VwIeA(1)'
-rw-r--r-- 1 kali kali  292 Sep 19 10:41  .7z
-rw-r--r-- 1 kali kali  292 Sep 19 10:41  .accdb
-rw-r--r-- 1 kali kali  292 Sep 19 10:41  .access
-rw-r--r-- 1 kali kali  292 Sep 19 10:41  .ackrc
-rw-r--r-- 1 kali kali  292 Sep 19 10:41  .action
-rw-r--r-- 1 kali kali  295 Sep 19 10:41  %C0%AE%C0%AE%C0%AF
-rw-r--r-- 1 kali kali  295 Sep 19 11:20 '%C0%AE%C0%AE%C0%AF(1)'
-rw-r--r-- 1 kali kali  295 Sep 19 10:41  %FF
-rw-r--r-- 1 kali kali  295 Sep 19 11:20 '%FF(1)'
-rw-r--r-- 1 kali kali  292 Sep 19 10:41 '!.gitignore'
-rw-r--r-- 1 kali kali  322 Sep 19 11:20 'google(1).com'
-rw-r--r-- 1 kali kali  322 Sep 19 10:41  google.com
-rw-r--r-- 1 kali kali  292 Sep 19 10:41  gXi3Cq
-rw-r--r-- 1 kali kali  292 Sep 19 11:20 'gXi3Cq(1)'
-rw-r--r-- 1 kali kali  292 Sep 19 10:41 '!.htaccess'
-rw-r--r-- 1 kali kali  292 Sep 19 10:41 '!.htpasswd'
-rw-r--r-- 1 kali kali  636 Sep 19 10:41  input
-rw-r--r-- 1 kali kali  105 Sep 19 10:41 'input(1)'
-rw-r--r-- 1 kali kali  636 Sep 19 11:20 'input(10)'
-rw-r--r-- 1 kali kali  105 Sep 19 11:20 'input(11)'
-rw-r--r-- 1 kali kali  644 Sep 19 11:20 'input(12)'
-rw-r--r-- 1 kali kali  337 Sep 19 11:20 'input(13)'
```

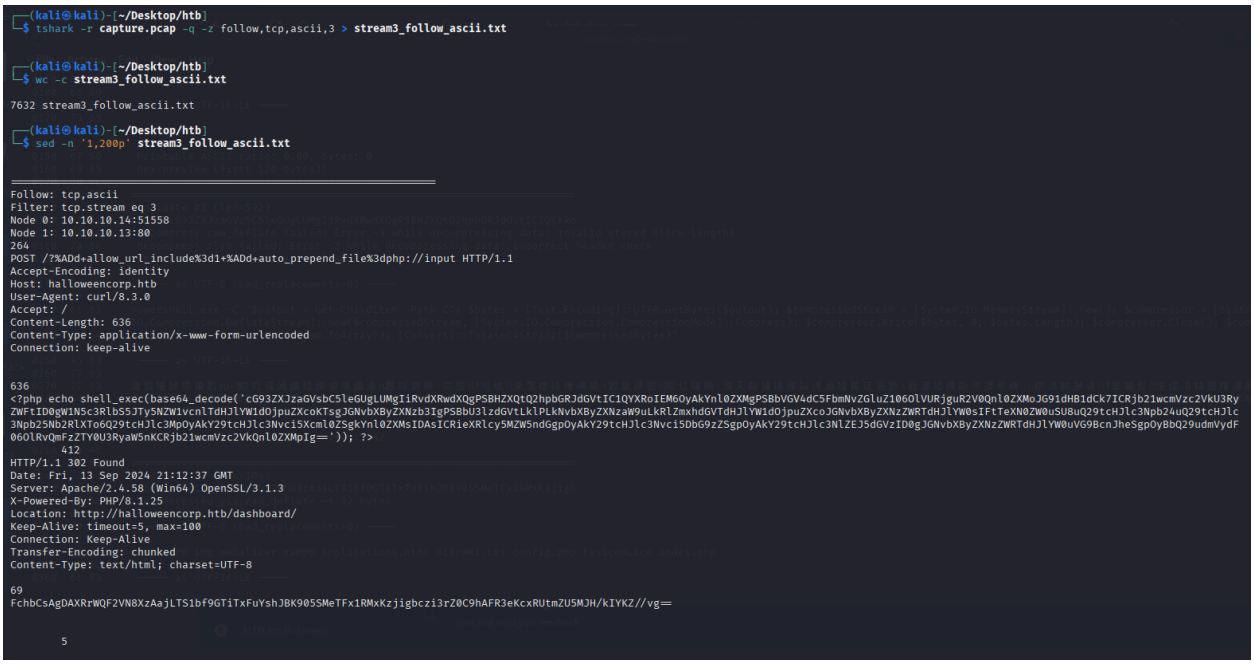
pulls files (objects) transferred via HTTP into `http_objs/` so you can inspect saved artifacts (you saw many files like `php.zip`, `php.7z`, `php.php`, `config.php`, `passwd`, `input` etc).

Extract POST bodies (raw hexdump)

We need the attacker-sent code (uploaded PHP) — easiest is to follow the TCP stream that contains the POST. You found the POST stream was **tcp.stream == 3**.

Command (extract full follow output for stream 3, ASCII view):

```
tshark -r capture.pcap -q -z follow,tcp,ascii,3 > stream3_follow_ascii.txt
wc -c stream3_follow_ascii.txt
sed -n '1,200p' stream3_follow_ascii.txt
```



`follow,tcp,ascii,N` reconstructs the full ASCII conversation for that TCP stream (both request and response). From this you can see the POST body (the PHP) and subsequent HTTP responses.

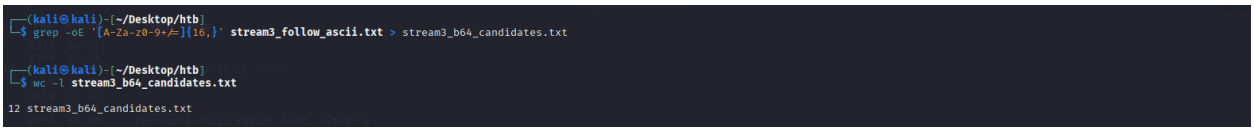
What you saw in that file: POST bodies containing `<?php echo shell_exec(base64_decode('...')); ?>` — the attacker passed PowerShell commands encoded as Base64 inside PHP.

5) Pull out the base64 blobs from the stream and decode them

The server responses were **base64** strings containing compressed (Deflate) data. We must extract the base64 substrings and attempt Base64 decode + raw DEFLATE decompression.

Commands (robust extraction, preserves candidates):

```
# extract candidate base64 runs (>= 16 chars) to a file
grep -oE '[A-Za-z0-9+/=]{16,}' stream3_follow_ascii.txt > stream3_b64_candidates.txt
wc -l stream3_b64_candidates.txt
```



Then use the helper script (this is the same script you used interactively — it decodes every candidate and tries raw deflate and zlib):

```
cat > decode_stream3_responses.py <<'PY'
#!/usr/bin/env python3
import re, base64, zlib, sys
fn = 'stream3_follow_ascii.txt'
s = open(fn, 'r', errors='ignore').read()
cands = re.findall(r'[A-Za-z0-9+/=]{16,}', s)
seen=set(); uniq=[]
for c in cands:
    if c in seen: continue
    seen.add(c); uniq.append(c)
for idx, b64 in enumerate(uniq, 1):
    print("="*60)
    print("Candidate", idx, "len", len(b64))
    try:
        raw = base64.b64decode(b64, validate=False)
    except Exception as e:
        print(" base64 decode error:", e); continue
    for name,w in [('raw_deflate', -zlib.MAX_WBITS), ('zlib', zlib.MAX_WBITS)]:
        try:
            out = zlib.decompress(raw, w)
```

```
            print("Decompressed via", name, "→", len(out), "bytes")
            print(out.decode('utf-8', errors='replace'))
            raise SystemExit(0)
        except Exception as e:
            print(" decompress", name, "failed:", e)
    print("First 120 bytes hex:", raw[:120].hex())
print("done")
PY
chmod +x decode_stream3_responses.py
./decode_stream3_responses.py | sed -n '1,2000p'
```

Why: PowerShell used `.NET DeflateStream` (raw DEFLATE). In Python `zlib.decompress()` needs `wbits=-zlib.MAX_WBITS` to decompress raw deflate streams; `zlib` fallback uses normal zlib header handling.

You discovered several successful decompressions. One of them (Candidate #9) decompressed to the `config.php` content — which contained the DB password and the flag.

---

6) The important decompressed outputs (interpretation)

From the successful candidates you saw readable output:

- `whoami` result (example): `desktop-pmoil0d\usr01` — confirms command execution context.
- `Get-ChildItem` listings from `C:\xampp` — shows the XAMPP directory structure, confirming the webroot and presence of `config.php`.

- `config.php` content (the crucial artifact) looked like:

```
<?php define('DB_SERVER', 'db'); define('DB_USERNAME', 'db_user'); define('DB_PASSWORD', 'HTB{f06_d154pp34r3d_4nd_fl46_w4s_f0und!}'); define('DB_DATABASE', 'a5BNadf');  $mysqli = new mysqli(...); ?>
```

So the flag is:

```
HTB{f06_d154pp34r3d_4nd_fl46_w4s_f0und!}
```

Why we had to do decompress + UTF-16 checks (short technical notes)

- The webshell ran **PowerShell** commands on Windows. To return output safely in HTTP, the attacker compressed the textual output with `.NET DeflateStream` (raw DEFLATE) and then Base64 encoded it.
- Raw DEFLATE has **no zlib header**, so Python needs `zlib.decompress(raw, -zlib.MAX_WBITS)`. If you use default zlib parameters you'll get header errors.
- PowerShell output is often encoded in **UTF-16LE**; if decoded bytes look garbled in UTF-8, try `utf-16le`.