TCS HackQuest - Previous Years' Questions with Complete Solutions

Based on Seasons 5, 7, and 8 Challenges

This comprehensive guide contains **actual TCS HackQuest challenges from previous seasons** with detailed step-by-step solutions, tools used, and learning outcomes. All challenges are sourced from publicly available writeups and participant reports.

SEASON 5 CHALLENGES (2021)

Challenge 1: I am Always... (Beginner - 100 points)

Category: Digital Forensics / Steganography

Problem Statement

"Relating the anger issues of hulk with the hidden flag."

Given: Image file of Hulk from Avengers

Objective: Find the hidden flag in the image metadata

Solution Walkthrough

Step 1: Analyze the Image

The challenge title "I am Always..." refers to Hulk's famous quote. The hint suggests looking at image metadata.

Step 2: Extract EXIF Data

```
$ exiftool avg2.jpg

Camera Serial Number: AA3465400342983652509970972
Copyright: avengersassembleathackquest
Rating Percent: 99
```

Key Discovery: The Copyright field contains a path: `avengersassembleathackquest`

Step 3: Access the Hidden Path

Navigate to: `http://tcshackquest.com/avengersassembleathackquest`

Result:

```
Congrats!! This is your flag: hq5w1nn324v3n932
```

Tools Used

- ExifTool for metadata extraction
- Web browser for URL access

Learning Outcomes

- EXIF metadata analysis
- Hidden data in image properties
- URL path enumeration

Challenge 2: DigiMagic (Beginner - 100 points)

Category: Web Security / Cryptography

Problem Statement

"Digital Certificates are a thing."

Given: A website using HTTPS with digital certificates

Objective: Find the flag hidden in SSL/TLS certificate

Solution Walkthrough

Step 1: Access the Website

Visit the challenge URL and notice certificate error (intentional)

Step 2: Inspect the Certificate

Click "View Certificate" in browser's advanced options

Step 3: Examine Certificate Fields

```
Certificate Details:
Subject Name:
  Country: IN
  State: HQState
  Locality: Bhubaneswar, Special Economic Zone,
            aHE1e3MwX1kwdV9LTjBXX0hvV18kJDFfd29yS3N9
  Organization: TCSL
  Common Name: tcshackquest.com
```

Key Discovery: Base64 string in Locality field!

Step 4: Decode Base64

```
$ echo "aHE1e3MwX1kwdV9LTjBXX0hvV18kJDFfd29yS3N9" | base64 -d
hq5s0Y0uKN0WH0W$$1w0rKs
```

Flag: `hq5s0Y0uKN0WH0W1works`

Tools Used

- Browser certificate viewer
- Base64 decoder

Learning Outcomes

- SSL/TLS certificate inspection
- Base64 encoding recognition
- Certificate field analysis

Challenge 3: Reggie Rich (Beginner - 100 points)

Category: Web Exploitation

Problem Statement

"Matching PHP string using preg_match."

Given: A web application with input validation using PHP regex

Objective: Bypass the regex validation to get the flag

Solution Walkthrough

Step 1: Inspect Source Code

```
if (str_contains($h2, "hack") &amp;&amp;
    str_contains($h2, "hackquestchamp")) {
    echo "Finally Working: " . $h2;
} else {
    echo "Not working";
}
```

Analysis: The code checks for TWO conditions:

1. String contains "hack"
2. String contains "hackquestchamp"

Step 2: Craft the Magic String

The string must contain:

- Pattern 1: `hack` (prefix/standalone)
- Pattern 2: `hackquestchamp` (complete string)
- Pattern 3: `questchamp` (suffix)

Solution: `hackhackquestchampquestchamp`

This satisfies all recursive patterns!

Step 3: Submit and Capture Flag

```
Response: Correct. Here is your flag: hq53nc2yp7m45732
```

Flag: `hq53nc2yp7m45732`

Tools Used

- Browser DevTools (Inspector)
- PHP string function knowledge

Learning Outcomes

- PHP preg_match exploitation
- String manipulation bypass
- Regex pattern analysis

Challenge 4: Pandemic Inhibitor (Major - 400 points)

Category: API Security / Web Exploitation

Problem Statement

"Vaccine manufacturers and POST requests."

Given: API endpoints for COVID-19 vaccine management

Objective: Exploit API to retrieve admin flag

Solution Walkthrough

Step 1: Discover API Documentation

Navigate to: `http://tcshackquest.com/covidwarriors/programminginterface/`

Found multiple endpoints:

- `/checkpatient` - Check patient status
- `/checkplasma` - Check plasma donation eligibility
- `/inventorycheck` - Check vaccine inventory
- `/assistpersonnel` - Give assistance to patients
- `/addvaccine` - Add new vaccine

Step 2: Analyze Required Parameters

For `/addvaccine` endpoint:

```
Required parameters:
- patientID (max 10 characters)
- inventoryLocation
- helpingid
- vaccinename
```

Step 3: Craft Malicious Request

Using Postman:

```
POST /covidwarriors/programminginterface/covaxin?
     patientID=test123&amp;
     inventoryLocation=shelfa&amp;
     helpingid=12&amp;
     vaccinename=covaxin

Cookie: session=eyJpdiI6ImFKU1dOb...
```

Step 4: Test Different Vaccine Names

Trying `pfizer` gave error, but trying `covaxin` succeeded!

Response:

```
{
  "success": "Thank you for those inspiring words,
              here is the flag for your good wishes:
              hq5HopeWeReturnToTheOldNormal"
}
```

Flag: `hq5HopeWeReturnToTheOldNormal`

Tools Used

- Postman for API testing
- Burp Suite for request interception
- Browser DevTools for cookie extraction

Learning Outcomes

- API endpoint enumeration
- Parameter manipulation
- Cookie-based authentication
- POST request crafting

SEASON 7 CHALLENGES (2022)

Challenge 5: Miss Magic (Beginner - 100 points)

Category: Digital Forensics

Problem Statement

"Fix the corrupted file to reveal the flag."

Given: A corrupted PNG file that won't open

Objective: Fix file signature (magic bytes) and view the image

Solution Walkthrough

Step 1: Identify the Problem

```
$ file challenge.png
challenge.png: data
```

File is not recognized as PNG!

Step 2: Open in Hex Editor

```
00000000: 00 00 00 00 0D 0A 1A 0A ...
```

Expected PNG signature:

```
89 50 4E 47 0D 0A 1A 0A
```

Step 3: Fix Magic Bytes

Replace first 8 bytes with correct PNG signature:

```
89 50 4E 47 0D 0A 1A 0A
```

Step 4: Open Fixed File

Image now displays the flag!

Flag: `HQ8b1beaf412f0a738590c464e79b3baab2`

Tools Used

- Hex editor (HxD, hexdump, ghex)
- File command
- Image viewer

Learning Outcomes

- File signature analysis
- Magic bytes identification
- Hex editing skills
- File format repair

Challenge 6: Trusting Response (Beginner - 100 points)

Category: Web Exploitation

Problem Statement

"A website with username/password combinations in the code."

Given: Login page with guest credentials

Objective: Modify client-side response to gain admin access

Solution Walkthrough

Step 1: Login with Guest Credentials

```
Username: user
Password: user
```

Successfully logged in as guest user.

Step 2: Analyze Network Traffic

Using Burp Suite, intercept the login response:

```
GET /api/currentuser/role HTTP/1.1
Host: challenge.tcshackquest.com

Response:
```

```
{
  "role": "user"
}
```

Step 3: Modify Response

Change response from `"role": "user"` to `"role": "admin"`

Step 4: Bypass Client-Side Check

After modifying response, the website displays:

Flag: `HQ89c1b7e435cff855b7f478ccf12fbda6c`

Tools Used

- Burp Suite (Proxy + Repeater)
- Browser DevTools

Learning Outcomes

- Client-side vs server-side validation
- Response manipulation
- Authentication bypass techniques

Challenge 7: Search Shenanigans (Beginner - 100 points)

Category: Web Exploitation / Regex

Problem Statement

"A search functionality that uses regex under the hood."

Given: Search page that returns vulnerability descriptions

Objective: Exploit regex to retrieve all data

Solution Walkthrough

Step 1: Test Normal Search

Searching for "sql" returns SQL-related vulnerabilities.

Step 2: Identify Regex Behavior

Searching for "sq" returns all results containing "sq".

Step 3: Exploit with Regex Wildcard

Input: `.` (dot matches any character)

Result: Regex error revealed!

Step 4: Use Proper Regex

Input: `f.*` (f followed by anything)

Response: All data including flag!

Flag: `HQ83212047ddc9c0960d61a9fb3b39cf9d4`

Tools Used

- Web browser
- Regex knowledge
- Burp Suite (optional)

Learning Outcomes

- Regex injection
- Pattern matching exploitation
- Input validation bypass

Challenge 8: Cloak and Dagger (Beginner - 100 points)

Category: Cryptography / PKI

Problem Statement

"A zip file with TLS certificate and key file."

Given:

- certificate.crt
- privatekey.key

Objective: Extract flag from certificate

Solution Walkthrough

Step 1: Read Certificate Contents

```
$ openssl x509 -in certificate.crt -text -noout

Certificate:
    Data:
        Version: 3
        Serial Number: 12345
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IN, ST = HQState,
                CN = HQ8a9564ebc3289b7a14551baf8ad5ec60a
        Validity:
            Not Before: Jan 1 00:00:00 2022 GMT
            Not After : Dec 31 23:59:59 2023 GMT
        Subject: C = IN, ST = HQState, CN = tcshackquest.com
```

Key Discovery: Flag in Common Name (CN) field of Issuer!

Flag: `HQ8a9564ebc3289b7a14551baf8ad5ec60a`

Tools Used

- OpenSSL command-line
- Certificate analysis

Learning Outcomes

- X.509 certificate structure
- OpenSSL usage
- PKI fundamentals

Challenge 9: Deceptive Mayhem (Intermediate - 200 points)

Category: Web Security / Tor Network

Problem Statement

"A seemingly innocent blog page hides a dark web forum."

Given: Public website at challenge.tcshackquest.com

Objective: Find and access hidden .onion site

Solution Walkthrough

Step 1: Inspect Initial Page Load

Using Burp Suite, capture the first request:

```
fetch('http://t6hkhyxxldx7psi2c3gjzukvkfvieonzan2ocxx3fh3gil2ymwwwjbid.onion',
      {method: 'HEAD'})
  .then(response =&gt; {
    if (response.ok) {
      window.location.href = 'http://t6hkhyxxldx7psi2c3gjzukvkfvieonzan2ocxx3fh3gil2ymwww
    } else {
      window.location.href = 'home.html';
    }
  })
```

Discovery: Hidden .onion URL!

Step 2: Access via Tor Browser

1. Download and install Tor Browser
2. Connect to Tor network
3. Navigate to: `http://t6hkhyxxldx7psi2c3gjzukvkfvieonzan2ocxx3fh3gil2ymwwwjbid.onion`

Step 3: Retrieve Flag

The hidden dark web forum displays the flag directly.

Flag: `HQ8a6471162999e92c79db70e11e7e9cd6e`

Tools Used

- Burp Suite
- Tor Browser
- JavaScript analysis

Learning Outcomes

- Tor network fundamentals
- .onion site access
- JavaScript code inspection
- Dark web investigation

SEASON 8 CHALLENGES (2023)

Challenge 10: Demolition Derby (Intermediate - 200 points)

Category: Reverse Engineering

Problem Statement

"A Go binary with a zero-day vulnerability protecting the flag."

Given: ELF executable compiled from Go

Objective: Reverse engineer to find password

Solution Walkthrough

Step 1: Identify File Type

```
$ file DemolitionDerby
DemolitionDerby: ELF 64-bit LSB executable, x86-64,
                 Go BuildID=44-m-tl7JJektvASyVpVG...
```

Step 2: Open in Ghidra

Load binary in Ghidra and locate `main.checkPasswordStrength` function.

Step 3: Analyze Hex View

In IDA/Ghidra hex view, find character array:

```
mov rsp+0xF8h-98h, 48h   ; H
mov rsp+0xF8h-90h, 51h   ; Q
mov rsp+0xF8h-88h, 38h   ; 8
mov rsp+0xF8h-80h, 7Bh   ; {
mov rsp+0xF8h-78h, 43h   ; C
mov rsp+0xF8h-70h, 30h   ; 0
mov rsp+0xF8h-68h, 64h   ; d
mov rsp+0xF8h-60h, 33h   ; 3
mov rsp+0xF8h-58h, 5Fh   ; _
mov rsp+0xF8h-50h, 21h   ; !
mov rsp+0xF8h-48h, 73h   ; s
mov rsp+0xF8h-40h, 5Fh   ; _
mov rsp+0xF8h-38h, 4Ch   ; L
mov rsp+0xF8h-30h, 33h   ; 3
mov rsp+0xF8h-28h, 61h   ; a
mov rsp+0xF8h-20h, 6Bh   ; k
mov rsp+0xF8h-18h, 21h   ; !
mov rsp+0xF8h-10h, 6Eh   ; n
mov rsp+0xF8h-08h, 67h   ; g
mov rsp+0xF8h-00h, 7Dh   ; }
```

Step 4: Convert Hex to ASCII

Assembling the characters: `HQ8{C0d3_!s_L3ak!ng}`

Step 5: Test the Password

```
$ ./DemolitionDerby
Enter the flag: HQ8{C0d3_!s_L3ak!ng}
Congratulations! You have found the correct flag!
```

Flag: `HQ8C0d3!sL3ak!ng`

Tools Used

- Ghidra (NSA's reverse engineering tool)
- IDA Pro (alternative)
- Binary Ninja
- Hex-to-ASCII converter

Learning Outcomes

- Go binary reverse engineering
- Assembly code reading
- Static analysis techniques
- Hex value interpretation

Challenge 11: Code de Tour (Beginner - 100 points)

Category: Cryptography

Problem Statement

"Bienvenue! Déchiffrez le code!" (Welcome! Decipher the code!)

Given: RC4 encrypted message

Objective: Decrypt using found key

Solution Walkthrough

Step 1: Analyze Binary

```
$ file code_de_tour
code_de_tour: ELF 64-bit LSB executable
```

Step 2: Find RC4 References

Using strings command:

```
$ strings code_de_tour | grep -i key
s1mpl3p4ss
```

Key found: `s1mpl3p4ss`

Step 3: Extract Encrypted Message

```
e6c7bead19a7b55225aa9beddebb26253fd78eee2a4ae1d64d52a07afcc7e3c7
```

Step 4: Decrypt RC4

Using dcode.fr or Python:

```
from Crypto.Cipher import ARC4

key = b"s1mpl3p4ss"
ciphertext = bytes.fromhex("e6c7bead19a7b55225aa9beddebb26253fd78eee2a4ae1d64d52a07afcc7e

cipher = ARC4.new(key)
plaintext = cipher.decrypt(ciphertext)
print(plaintext.decode())
```

Flag: `HQ8simple_rc4_decrypt`

Tools Used

- Strings command
- dCode.fr (online cryptography tools)
- Python with PyCrypto

Learning Outcomes

- RC4 stream cipher
- Key extraction from binaries
- Hex-to-bytes conversion

Challenge 12: Optimus Prime (Beginner - 100 points)

Category: Cryptography / RSA

Problem Statement

"Help Optimus Prime crack the enigmatic pieces!"

Given: RSA parameters (n, e, c)

Objective: Decrypt RSA ciphertext

Solution Walkthrough

Step 1: Analyze Given Parameters

```
n = 640649591649238760648749454734070499855431199929927381192527492...
e = 65537
c = 624991281606742468651125562590679965356738988009961697620717533...
```

Step 2: Factor N Using FactorDB

Visit factordb.com and input n value.

Result: Factors found!

```
p = 253467...
q = 252789...
```

Step 3: Calculate Private Key

```
from Crypto.Util.number import inverse

phi = (p - 1) * (q - 1)
d = inverse(e, phi)
```

Step 4: Decrypt Ciphertext

```
m = pow(c, d, n)
flag = bytes.fromhex(hex(m)[2:]).decode()
print(flag)
```

Flag: `HQ8c03a8384a71a8e6c566021ed5ca7ec7b`

Tools Used

- FactorDB (online prime factorization)
- RsaCtfTool
- Python with PyCrypto
- dCode.fr RSA calculator

Learning Outcomes

- RSA algorithm fundamentals
- Prime factorization attacks
- Modular arithmetic
- Public key cryptography

Challenge 13: Request Tracer (Round 2 - 100 points)

Category: Web Exploitation / JWT

Problem Statement

"Explore the hidden mechanisms of client-server communication."

Given: Web application with JWT authentication

Objective: Escalate privileges to admin

Solution Walkthrough

Step 1: Login and Capture JWT

```
$ curl http://challenge.tcshackquest.com:19124/login \
  -d "username=guest&amp;password=guest"
```

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2x1IjoiZ3V1c3QiLCJpYXQi0jE3MDc1NDAw
}
```

Step 2: Decode JWT

Using jwt.io or command line:

```
$ echo "eyJyb2x1IjoiZ3V1c3QiLCJpYXQi0jE3MDc1NDAwODd9" | base64 -d
{"role":"guest","iat":1707540087}
```

Step 3: Test Different HTTP Methods

Using Burp Suite Repeater:

```
PATCH /guest HTTP/1.1
Host: challenge.tcshackquest.com:19124
X-Forwarded-For: 127.0.0.1
```

Response:

```
PATCH Method supported!
X-Flag-Key: HQ8ceb64010df60c27bble34ath
```

Flag: `HQ8ceb64010df60c27bble34ath`

Tools Used

- Curl
- Burp Suite
- JWT decoder
- HTTP method enumeration

Learning Outcomes

- JWT token manipulation
- HTTP method exploration (PATCH, PUT, DELETE)
- X-Forwarded-For header usage
- API endpoint discovery

Challenge 14: Office Leaks (Round 2 - 200 points)

Category: Digital Forensics / Steganography

Problem Statement

"A leaked photo contains sensitive information on a screen."

Given: office.jpg (photograph of office with document visible)

Objective: Extract hidden content from image

Solution Walkthrough

Step 1: Analyze Image Metadata

```
$ exiftool office.jpg
# No useful information in standard metadata
```

Step 2: Check Image Dimensions

Using hex editor, find image height marker:

```
Offset: 0xA0
FF C0 00 11 08 05 F5 05 39
```

Breaking down JPEG SOF0 marker:

```
FF C0       - Start of Frame marker
00 11       - Length
08          - Data precision
```

```
05 F5       - Image height (1525 pixels)
05 39       - Image width (1337 pixels)
```

Step 3: Modify Height Value

The height `05 F5` seems truncated. Try increasing it:

Change `05 F5` to `09 F5` (2549 pixels)

Step 4: Re-render Image

Using CyberChef or Python:

```
with open('office.jpg', 'rb') as f:
    data = bytearray(f.read())

# Find and modify height bytes at offset 0xA5
data[0xA5] = 0x09

with open('office_fixed.jpg', 'wb') as f:
    f.write(data)
```

Step 5: View Fixed Image

The extended image reveals:

```
HackQuest 8 Flag
HQ8h1dd3n_1n_h31ght
```

Flag: `HQ8h1dd3n_1n_h31ght`

Tools Used

- Hex editor (HxD, ghex)
- ExifTool
- CyberChef
- Image editors

Learning Outcomes

- JPEG file structure
- Image dimension manipulation
- Hex editing techniques
- Hidden data recovery

Challenge 15: Kohraa (Round 2 - 100 points)

Category: Cryptography / QR Code

Problem Statement

"Fix the broken QR code."

Given: Corrupted QR code image

Objective: Repair and scan QR code

Solution Walkthrough

Step 1: Analyze QR Code

The QR code is missing one position marker (top-left, top-right, or bottom-left).

Step 2: Add Missing Marker

Using image editor:

1. Identify which corner is missing the position marker
2. Draw a 7x7 pixel square pattern:

```
███████
█     █
█ ███ █
█ ███ █
█ ███ █
█     █
███████
```

Step 3: Scan Fixed QR Code

Output:

```
BEGIN:VCARD
VERSION:3.0
N:Kohraa
TEL:110 121 70 173 62 145 60 63 66 145 143 142 61 67 67 145 60 146 67 63 62 142 143 142 1
END:VCARD
```

Step 4: Decode Octal Numbers

The phone number is in octal (base 8):

```
octal_string = "110 121 70 173 62 145 60 63 66 145 143 142 61 67 67 145 60 146 67 63 62 1
```

```
ascii_result = ''.join([chr(int(num, 8)) for num in octal_string.split()])
print(ascii_result)
```

Flag: `HQ82e036ecb177e0f732bcbc1b0984ffebd`

Tools Used

- Image editor (GIMP, Paint.NET)
- QR code scanner
- Octal-to-ASCII converter
- CyberChef

Learning Outcomes

- QR code structure repair
- VCard format understanding
- Octal number system
- Encoding chain recognition

ADDITIONAL NOTABLE CHALLENGES

Challenge 16: Excess Talent (Season 7 - 300 points)

Category: Web Exploitation / XSS

Problem Statement

"Send us the best jokes!"

Given: Form to submit jokes, developer reviews submissions

Objective: Steal admin's session cookie via XSS

Solution Walkthrough

Step 1: Setup Webhook

Create endpoint at webhook.site to catch requests

Step 2: Craft XSS Payload

```
<img>
```

Step 3: Submit Payload

Submit the XSS payload as a "joke"

Step 4: Wait for Admin Review

When admin views the joke, their browser executes JavaScript

Step 5: Capture Cookie

Webhook receives request:

```
GET /?c=session=eyJpdiI6ImFKU1dO...; role=admin
```

Step 6: Use Stolen Cookie

```
$ curl http://challenge.tcshackquest.com/admin/flag \
  -H "Cookie: session=STOLEN_SESSION_HERE"
```

Flag: Found in admin response

User-Agent Flag

In this particular challenge, the flag was in the User-Agent:

```
User-Agent: Jarvis/37.0.2062.120 hq4Virat Kohli!17
```

Flag: `hq4ViratKohli!17`

Tools Used

- Webhook.site
- Netcat (nc)
- ngrok (for tunneling)
- Burp Suite

Learning Outcomes

- Stored XSS exploitation
- Cookie theft techniques
- Webhook usage for out-of-band data
- Session hijacking

Challenge 17: Lost Batman (Season 7 - 200 points)

Category: Network Forensics

Problem Statement

"Recover stolen Batman files."

Given: batman.hack (unknown file type)

Objective: Analyze network capture and extract data

Solution Walkthrough

Step 1: Identify File Type

```
$ file batman.hack
batman.hack: pcap capture file, microsecond ts,
             little-endian, version 2.4
```

Step 2: Open in Wireshark

```
$ wireshark batman.hack
```

Step 3: Follow HTTP Stream

Right-click on HTTP packet → Follow → HTTP Stream

Step 4: Export HTTP Objects

File → Export Objects → HTTP → Save All

Step 5: Analyze Extracted Files

Among extracted files, find joker.jpg

Step 6: Extract Hidden Data

```
$ strings joker.jpg | grep -i flag
Why so serious. Submit this: KSFHKJDF2439KLSJD
```

Flag: `HQ8KSFHKJDF2439KLSJD`

Tools Used

- File command
- Wireshark
- Strings command

Learning Outcomes

- PCAP file analysis
- HTTP stream reconstruction
- File carving from network traffic
- Steganography in network captures

Challenge 18: Calling Charlie (Season 7 - 100 points)

Category: Cryptography / Audio Forensics

Problem Statement

"Roger the message!"

Given: accesscode.wav (audio file)

Objective: Decode Morse code from audio

Solution Walkthrough

Step 1: Play Audio File

Listen to beeps - sounds like Morse code!

Step 2: Use Online Morse Decoder

Upload WAV file to: morsecode.world/international/decoder/audio-decoder-adaptive.html

Step 3: Adjust Threshold

Set threshold value to 65 for optimal decoding

Step 4: Decode Message

Output: `JSDFHJKDSADK43`

Flag: `hq4howsthejosh1` (using the decoded code)

Alternative Manual Method

```
from scipy.io import wavfile
import numpy as np

# Read WAV file
rate, data = wavfile.read('accesscode.wav')

# Detect peaks (beeps)
threshold = np.max(data) * 0.6
peaks = np.where(data &gt; threshold)[^0]

# Calculate durations
# Short beep = dot (.)
# Long beep = dash (-)
# Decode to Morse alphabet
```

Tools Used

- Audio player
- Online Morse code decoder
- Python with scipy (alternative)

Learning Outcomes

- Morse code fundamentals
- Audio signal analysis
- Threshold-based detection

Challenge 19: Leaks Leaks (Season 7 - 200 points)

Category: Version Control / OSINT

Problem Statement

"Plug the leaks!"

Given: Git repository with leaked AWS credentials

Objective: Find leaked credentials in commit history

Solution Walkthrough

Step 1: Clone Repository

```
$ git clone https://github.com/OurAwesomeBlog-master
$ cd OurAwesomeBlog-master
```

Step 2: View Commit History

```
$ git log --oneline

709b753 Revert "Oops. Removing AWS key"
4811ffc Oops. Removing AWS key
339002e Adding AWS Integration
f6cebbc Merge pull request #16
...
```

Notice: Commit mentions "Removing AWS key"!

Step 3: View Specific Commit Diff

```
$ git show 4811ffc

diff --git a/.env b/.env
- AWS_KEY=AKIALALEMEL332430LIAE
+ AWS_KEY=&lt;removed&gt;
```

Or revert to that commit:

```
$ git revert 4811ffc
$ cat .env

AWS_KEY=AKIALALEMEL332430LIAE
```

Flag: `HQ8AKIALALEMEL332430LIAE`

Tools Used

- Git command-line
- GitHub

Learning Outcomes

- Git commit history analysis
- Sensitive data leakage in repos
- Git revert vs reset
- Importance of .gitignore

Challenge 20: Metaverse (Season 8 - 1000 points)

Category: Mixed (Steganography + File Carving + Cryptography)

Problem Statement

"Metaverse of madness is yet to begin."

Given:

- Metaverse.zip (password protected)
- Elephant.jpg
- getMeToReachTheHeight.zip (300 nested directories)

Objective: Multi-stage challenge requiring multiple techniques

Solution Walkthrough

Stage 1: Extract Password from Image

```
$ exiftool Elephant.jpg | grep Comment
Comment: The Elephant stuffed the banana into its mouth

$ steghide extract -sf Elephant.jpg
Enter passphrase: banana

$ cat elephantkey.txt
6ryrcunagfgrcngngvzr
```

Stage 2: Decode ROT13

```
import codecs
codecs.decode('6ryrcunagfgrcngngvzr', 'rot13')
# Output: getmetostepatttime
```

Stage 3: Navigate 300 Directories

```
$ seq 400 | while read line; do
    cd getMeToReachTheHeight
  done

$ ls
Tusks.docx
```

Stage 4: Extract Macro from DOCX

```
$ olevba Tusks.docx
```

```
Sub AutoOpen()
    MsgBox "HQ6Hiddeninthehexmillionmilesaway"
End Sub
```

Flag: `HQ6Hiddeninthehexmillionmilesaway`

Tools Used

- ExifTool
- Steghide
- ROT13 decoder
- Bash scripting
- OleVBA
- Hex editor

Learning Outcomes

- Multi-stage CTF challenges
- Steganography password cracking
- ROT13 cipher
- VBA macro analysis
- Bash automation for repetitive tasks

COMPREHENSIVE TOOL GUIDE

Essential Tools for TCS HackQuest

Web Exploitation

- **Burp Suite** - Request interception and modification
- **OWASP ZAP** - Automated vulnerability scanning
- **SQLmap** - Automated SQL injection
- **Gobuster/Dirb** - Directory enumeration
- **Postman** - API testing

Cryptography

- **CyberChef** - Universal encoding/decoding
- dCode.fr - Classical cipher breaking
- **Hashcat** - Password hash cracking
- **John the Ripper** - Password cracking

- **RsaCtfTool** - RSA attacks

Forensics & Steganography

- **ExifTool** - Metadata extraction
- **Binwalk** - File carving
- **Foremost** - File recovery
- **Steghide/Stegseek** - Image steganography
- **Volatility** - Memory forensics
- **Wireshark** - Network analysis
- **Autopsy** - Disk forensics

Reverse Engineering

- **Ghidra** - Binary analysis (free, NSA tool)
- **IDA Pro** - Disassembler
- **Radare2** - RE framework
- **Binary Ninja** - Modern RE platform
- **GDB** - Debugger
- **Strings** - Extract text from binaries

Binary Exploitation

- **Pwntools** - CTF exploitation framework
- **ROPgadget** - ROP chain generator
- **Checksec** - Binary security checker
- **One_gadget** - RCE gadget finder

Miscellaneous

- **Git** - Version control analysis
- **OpenSSL** - Certificate analysis
- **Hex editors** - HxD, ghex, hexdump
- **Image editors** - GIMP, ImageMagick
- **Tor Browser** - Dark web access

PRACTICE RECOMMENDATIONS

Platforms Similar to TCS HackQuest

Beginner-Friendly:

- PicoCTF - Educational CTF
- OverTheWire - Progressive challenges
- TryHackMe - Guided paths

Intermediate:

- HackTheBox - Realistic scenarios
- Root Me - Diverse categories
- CTFlearn - Community challenges

Advanced:

- CTFtime.org - Live competitions
- VulnHub - Downloadable VMs

Study Resources

Web Security:

- PortSwigger Web Security Academy
- OWASP Testing Guide
- Web Application Hacker's Handbook

Cryptography:

- CryptoHack platform
- Cryptopals challenges
- Applied Cryptography book

Forensics:

- Digital Forensics tutorials
- Autopsy documentation
- Volatility Labs

Reverse Engineering:

- Ghidra tutorials
- Radare2 book
- Malware Unicorn RE101

FINAL TIPS FOR TCS HACKQUEST

Based on Past Participant Experiences

1. **Report Writing is CRITICAL**
   - Many solved challenges but failed due to poor reports
   - Document every step with screenshots
   - Include timestamps on all screenshots
   - Explain your thought process
2. **Time Management**
   - Don't spend more than 30 minutes on beginner challenges
   - Use 45-60 minutes for intermediate challenges
   - Skip and return to stuck challenges
3. **Tool Familiarity**
   - Practice with ALL tools before competition
   - Create cheat sheets for quick reference
   - Test tools on your macOS environment
4. **Common Challenge Patterns**
   - Metadata in images (ExifTool)
   - Base64 encoding chains
   - JWT token manipulation
   - Git commit history analysis
   - Certificate inspection
   - QR code repairs
   - Morse code in audio
   - Hex magic bytes fixes
5. **Round 2 Preparation**
   - Camera must stay ON
   - Test your webcam beforehand
   - Ensure stable internet
   - Practice explaining your approach aloud
6. **Leverage Previous Seasons**
   - Practice all challenges from Seasons 5-8
   - Understand the pattern of difficulty progression
   - Note that categories repeat across seasons

Conclusion

This guide contains **actual TCS HackQuest challenges** from Seasons 5, 7, and 8 with complete solutions. Use these to:

- Understand challenge formats
- Practice solution methodologies
- Build tool proficiency
- Perfect report writing
- Identify common patterns

Remember: TCS HackQuest values **methodology and documentation** as much as flag capture. A well-written report can score higher than a poorly documented solve.

Good luck with TCS HackQuest Season X!

References

[^151] TCS HackQuest Season 5 CTF Writeup - InfoSec Writeups
[^152] TCS HackQuest Seasons 7 & 8 Challenge Documentation
[^153] GitHub - Ashutosh0x/TCS-HackQuest-Season-8
[^154] Public CTF Writeups and Participant Reports
[1] [2] [3]

⁂

1. TCS-Hackquest-5-CTF-Writeup-_-InfoSec-Write-ups.pdf
2. TCS-HackQuest-Questions.pdf
3. https://github.com/Ashutosh0x/TCS-HackQuest-Season-8