

Cybersecurity Wargame Internship Task- Report

Cybersecurity Internship Report Intern Name: Supriya Thamke, Vishakha Ghadage, Manasi Jagtap

Program: Digisuraksha Parhari

Foundation Internship Issued By: Digisuraksha Parhari Foundation

Supported By: Infinisec Technologies Pvt. Ltd.

Project Report: OverTheWire – Krypton Wargame Analysis

Objective:

The **Krypton wargame** introduces beginners to the world of **classical cryptography**. The goal is to **solve each level by decoding encrypted text** and retrieving passwords to access the next level.

Tools Used:

- **Linux Terminal** (SSH for remote access)
 - **Base64 decoding tool**
 - **Caesar cipher decoder**
 - **Online tools:** CyberChef, dCode
 - **Text editors:** nano, vim, cat, grep
-

Level-by-Level Breakdown:

Krypton Level 0 → Level 1

Challenge:

- SSH access using a provided password.

Solution:

- Connect using:
- `ssh krypton0@krypton.labs.overthewire.org -p 2222`
- Password: `krypton0` (provided on website)

Next password found in:

```
cat /krypton/krypton0/keyfile.dat
```

🔒 Krypton Level 1 → Level 2

Challenge:

- A message is encrypted with a **Caesar cipher** (ROT13 logic).

Logic:

- ROT13 shifts each letter by 13 places.

Command used:

```
cat /krypton/krypton1/keyfile.dat | tr 'A-Za-z' 'N-ZA-Mn-za-m'
```

Explanation:

- `tr` command is used to perform letter substitution for ROT13.
-

🔒 Krypton Level 2 → Level 3

Challenge:

- CIPHERED text in a file
- Hint that the text was shifted and characters can only be uppercase

Steps:

1. View the file:
 2. `cat /krypton/krypton2/keyfile.dat`
 3. Use Caesar cipher brute force:
 - Try all 26 shifts (or use dCode's Caesar solver)
 4. Identify plaintext line and extract password.
-

🔒 Krypton Level 3 → Level 4

Challenge:

- Encrypted password using **monoalphabetic substitution cipher**

Steps:

1. Analyze the cipher file:
2. `cat /krypton/krypton3/keyfile.dat`
3. Frequency analysis:
 - Replace common letters based on English frequency (E, T, A, O...)
 - Use online tools like [dCode](#) or do manual mapping.

Tools:

- CyberChef (Frequency Analysis)
 - Python (for mapping logic, optional)
-

Krypton Level 4 → Level 5

Challenge:

- Similar monoalphabetic cipher, but encoded using a **key-based substitution**

Logic:

- Custom key provided in the script: THEQUICKBROWNFXJMPSVLAZYDG
- Remaining letters filled alphabetically

Python or bash used to decode:

- Create two strings (plain and cipher) and map using tr
-

Krypton Level 5 → Level 6

Challenge:

- Encoded message using **Base64**

Solution:

```
cat /krypton/krypton5/keyfile.dat | base64 -d
```

Password is revealed after decoding.

Krypton Level 6 → Level 7 (Final)

Challenge:

- You are required to break a more complex cipher (e.g., Vigenère cipher or custom logic)

General approach:

- Use known text patterns
 - Use CyberChef's Vigenère solver
 - Guess common keywords if needed
-

□ What We Learned:

- Basics of **classical ciphers**: Caesar, Substitution, Base64, Vigenère

- How to use **Linux tools** (`tr`, `cat`, `base64`, `ssh`)
 - Importance of **pattern recognition** and **frequency analysis**
 - Practiced **reading, decoding, and logical thinking**
-

Conclusion:

The Krypton wargame builds a strong foundation in cryptography and Linux command-line usage. It's beginner-friendly yet intellectually engaging — perfect for those stepping into cybersecurity or ethical hacking.

Certainly! Here's a comprehensive report analyzing the OverTheWire Natas wargame levels 0 through 10. This report includes step-by-step solutions, tools utilized, and the underlying logic for each level.

OverTheWire Natas Levels 0–34: Comprehensive Analysis

Objective: □ The Natas wargame is designed to teach the basics of server-side web security by challenging players to find passwords hidden in web pages, source code, cookies, and more □ □

◆ Level 0: Introduction to Viewing Page Source

- *URL: □`http://natas0.natas.labs.overthewire.org`□
- *Username: □`natas`□
- *Password: □`natas`□ □

Approach:

□ Upon accessing the page, it displays: "You can find the password for the next level on this page" □
□ However, the password isn't visible in the rendered content. □

- *Solution: □ View the page source (Right-click → "View Page Source") to find the password within an HTML comment. □ □

*Tools Used: □ Web browser's "View Page Source" feature. □ □

*Logic: □ Understanding that developers may leave sensitive information in HTML comments, which are not rendered on the page but are visible in the source code. □ □

◆ Level 1: Exploring Developer Tools

- *URL: □`http://natas1.natas.labs.overthewire.org`□
- *Username: □`natas1`□
- *Password: □(Obtained from Level 0)□ □

Approach:

- The page displays: "You can find the password for the next level on this pag." - No visible password in the rendered content.
- *Solution: Use browser developer tools (F12) to inspect the page elements and find the password within an HTML comment.

*Tools Used: Browser Developer Tools (Elements tab).

*Logic: Recognizing that inspecting the DOM can reveal hidden elements or comments containing sensitive information.

◆ Level 2: Directory Enumeration

- URL: http://natas2.natas.labs.overthewire.org
- Username: natas2
- Password: (Obtained from Level 1)

Approach:

- The main page doesn't provide any useful information.
- Solution: View the page source to find a reference to an "images" directory.
- Navigate to http://natas2.natas.labs.overthewire.org/files/ to find a directory listing.
- Locate a file containing the password.

Tools Used: Web browser for manual directory traversal.

Logic: Understanding that web servers may have accessible directories that are not linked on the main page but can be discovered through hints in the source code.

◆ Level 3: Utilizing robots.txt

- UR: http://natas3.natas.labs.overthewire.org
- Username: natas3
- Password: (Obtained from Level 2)

*Approach:

- The main page provides no useful information.
- Solution: Access http://natas3.natas.labs.overthewire.org/robots.txt to find disallowed directories.
- Navigate to the disallowed directory to find a file containing the password.

Tools Used: Web browser to access robots.txt and disallowed directories.

Logic: Recognizing that robots.txt files can reveal hidden directories that are intended to be excluded from search engine indexing but may contain sensitive information.

◆ Level 4: Referer Header Manipulation

- **URL:** http://natas4.natas.labs.overthewire.org
- **Username:** natas4
- **Password:** (Obtained from Level 3)

**Approach:*

- Accessing the page results in a message denying access.
- **Solution:** Use browser developer tools to modify the "Referer" header in the HTTP request to <http://natas5.natas.labs.overthewire.org>.
- Refresh the page with the modified header to gain access and retrieve the password.

Tools Used: Browser Developer Tools (Network tab) or tools like cURL to modify HTTP headers.

Logic: Understanding that some web applications use the "Referer" header for access control, which can be manipulated to bypass restrictions.

◆ Level 5: Cookie Manipulation

- **URL:** http://natas5.natas.labs.overthewire.org
- **Username:** natas5
- **Password:** (Obtained from Level 4)

Approach

- The page indicates that access is denied.
- **Solution:** Inspect the cookies set by the server using browser developer tools.
- Find a cookie named `loggedin` set to `0`.
- Change its value to `1` and refresh the page to gain access and retrieve the password.

Tools Used: Browser Developer Tools (Application tab) to view and modify cookies.

Logic: Recognizing that authentication states can be stored in cookies, which, if not securely implemented, can be manipulated to gain unauthorized access.

◆ Level 6: Hidden Files in Source Code

- **URL:** http://natas6.natas.labs.overthewire.org
- **Username:**

NATAS Level 11: Cookie Manipulation

- **URL:** http://natas11.natas.labs.overthewire.org
- **Username:** natas11
- **Password:** (from Level 10 output)

Approach:

- A cookie named `data` is sent to the browser.
- The site mentions the data is encrypted using XOR and that the code for encryption is available.

Solution:

- Decode and understand the encryption function from the source code.
- Write a Python script to decrypt the cookie and re-encrypt it after modifying values like admin from 0 to 1.

🛠 Tools Used:

- **Python script for XOR encryption/decryption**
- **Browser Developer Tools** (for cookie management)

❑ Logic:

- XOR encryption is reversible if the key is known.
- Re-encode the modified dictionary with admin:1 using the same XOR function and base64, then set it as the new cookie.

🔒 NATAS Level 7: Directory Traversal

URL: <http://natas7.natas.labs.overthewire.org>

Username: natas7

Password: (from Level 6 output)

🔍 Approach:

The page has links like `page=home`. This is a hint that the application loads pages using a GET parameter.

✓ Solution:

Use directory traversal to load the `/etc/natas_webpass/natas8` file by visiting:

bash

CopyEdit

?page=../../../../etc/natas_webpass/natas8

🛠 Tools Used:

Browser URL manipulation

❑ Logic:

Web applications that include files based on user input are vulnerable to path traversal if they don't **sanitize input**.

🔒 NATAS Level 8: XOR-Based Password Encoding

URL: <http://natas8.natas.labs.overthewire.org>

Username: natas8

Password: (from Level 7 output)

🔍 Approach:

Source code reveals that the correct encoded password is compared against the result of an XOR operation.

✓ Solution:

Use the known encoded string and XOR it against each character in the input to reverse-engineer the original password.

🛠 Tools Used:

Python script to reverse XOR

❑ Logic:

If $A \text{ XOR } B = C$, then $B = A \text{ XOR } C$ — use this to reverse the operation and extract the original password.

🔒 NATAS Level 9: Command Injection via Input

URL: <http://natas9.natas.labs.overthewire.org>

Username: natas9

Password: (from Level 8 output)

Approach:

The app performs a grep on the input value.

Solution:

Inject a command using a separator like ;, e.g., enter:

bash

CopyEdit

; cat /etc/natas_webpass/natas10

Tools Used:

Browser input form

Command injection knowledge

Logic:

Unsanitized input passed to shell commands allows arbitrary command execution.

NATAS Level 10: Filter Bypass in Command Injection

URL: <http://natas10.natas.labs.overthewire.org>

Username: natas10

Password: (from Level 9 output)

Approach:

The input is passed to a shell, but with filters removing ;, |, and &.

Solution:

Use a newline character (\$IFS or actual newline %0A) or logic operator like && encoded differently to bypass filters:

bash

CopyEdit

. /etc/natas_webpass/natas11

Tools Used:

Payload encoding

Burp Suite or URL encoder

Logic:

Filters block specific characters, but logic or encoded versions can bypass them.

NATAS Level 11: Cookie Manipulation & XOR Decryption

URL: <http://natas11.natas.labs.overthewire.org>

Username: natas11

Password: (from Level 10 output)

Approach:

The application sets a cookie named data which stores a base64-encoded, XOR-encrypted JSON object. The JSON includes a showpassword field.

Solution:

1. Decode the cookie using Base64.
2. XOR the decoded value with a known plaintext (like {"showpassword":"no", "bgcolor":"#ffffff"}) to find the encryption key.
3. Modify the JSON to {"showpassword":"yes", "bgcolor":"#ffffff"}, XOR it with the same key, then Base64 encode it.
4. Replace the original cookie with the modified one.

Tools Used:

Python script for XOR encryption/decryption and Base64 encoding/decoding

Logic:

XOR is reversible and if part of the plaintext is known, the key can be derived. By flipping the "showpassword" flag, you gain access to the next level's password.

NATAS Level 12: File Upload Exploit

- **URL:** http://natas12.natas.labs.overthewire.org
- **Username:** natas12
- **Password:** (from Level 11 output)

Approach:

- Web app allows uploading .jpg files.
- Source code shows no real file type validation; only file extension check is done.

Solution:

- Upload a .php file disguised as an image (e.g., shell.php.jpg) with PHP code like:

```
php
CopyEdit
<?php echo shell_exec($_GET['cmd']); ?>
```
- After upload, visit the file URL and use ?cmd=cat /etc/natas_webpass/natas13 to get the password.

Tools Used:

- **Custom PHP payload**
- **Browser upload interface**

Logic:

- If the server saves uploaded files in a web-accessible directory, and doesn't validate MIME type or content, PHP can be executed.

NATAS Level 13: File Upload with MIME Check

- **URL:** http://natas13.natas.labs.overthewire.org
- **Username:** natas13
- **Password:** (from Level 12 output)

Approach:

- Upload still allowed, but this time with an extra MIME type check (image).

Solution:

- Use a tool like **Burp Suite** or **Postman** to intercept the request.
- Change the Content-Type to image/jpeg while uploading a .php payload.

Tools Used:

- **Burp Suite or Postman**
- PHP reverse shell or simple command execution payload

Logic:

- MIME type can be spoofed; web servers don't always validate it deeply.

NATAS Level 14: SQL Injection – Login Bypass

- **URL:** http://natas14.natas.labs.overthewire.org
- **Username:** natas14
- **Password:** (from Level 13 output)

Approach:

- Login form with username and password fields.
- Source code uses raw SQL queries with user input.

Solution:

- Classic SQL Injection payload:

vbnnet

CopyEdit

username: ' OR 1=1 --

password: (anything)

Tools Used:

- **Browser**
- **SQL injection techniques**

Logic:

- The payload makes the SQL query always return true, bypassing the login mechanism.

NATAS Level 15: Blind SQL Injection (Boolean-Based)

- **URL:** <http://natas15.natas.labs.overthewire.org>
- **Username:** natas15
- **Password:** (from Level 14 output)

Approach:

- No visible response, but backend behavior changes with different inputs.

Solution:

- Use a script to perform blind SQL injection:

```
sql  
CopyEdit  
natas16" AND password LIKE BINARY "a%'' --
```

- Automate the character-by-character brute-force.

Tools Used:

- **Python script with requests module**
- **Boolean-based SQL injection**

Logic:

- Even without visible output, the timing or content of the response reveals success.

Would you like me to continue from **Level 16 to 20** next in the same format? Let me know if you'd like a downloadable version or code snippets included too!

40

You said:

yes till 34

ChatGPT said:

Perfect! Here's the continuation of your **OverTheWire Natas Levels 16 to 34** walkthrough in the same structured format. This will complete the documentation for your internship submission.

NATAS Level 16: Blind Command Injection

- **URL:** <http://natas16.natas.labs.overthewire.org>
- **Username:** natas16
- **Password:** (from Level 15 output)

Approach:

- Web app filters input but allows command execution via grep.

Solution:

- Inject a command using:

```
bash
```

```
CopyEdit
```

```
$(grep ^a /etc/natas_webpass/natas17)
```

- Use a script to perform character-by-character brute-force.

Tools Used:

- **Command injection automation with Python**
- **requests module**

Logic:

- Even though dangerous characters are filtered, command substitution using \$() is allowed.

NATAS Level 17: Blind Time-Based SQL Injection

- **URL:** <http://natas17.natas.labs.overthewire.org>

- **Username:** natas17
- **Password:** (from Level 16 output)

🔍 Approach:

- Blind SQLi with no visible response changes. Use SLEEP() to determine correctness.

✓ Solution:

- Use Python to automate requests with payloads like:

sql

CopyEdit

natas18" AND IF(password LIKE BINARY "a%", SLEEP(2), 0) --

🛠 Tools Used:

- **Python + requests + time module**

□ Logic:

- Time delay (via SLEEP()) lets us extract data when the app gives no output.

🔒 NATAS Level 18: Session ID Enumeration

- **URL:** http://natas18.natas.labs.overthewire.org
- **Username:** natas18
- **Password:** (from Level 17 output)

🔍 Approach:

- Admin login depends on session ID.

✓ Solution:

- Brute-force session IDs from 1 to 640 (or 1024 depending on setup).
- Set PHPSESSID cookie and check if admin access is granted.

🛠 Tools Used:

- **Python + requests**
- **Burp Suite** (optional)

□ Logic:

- Predictable session IDs = vulnerability.

🔒 NATAS Level 19: Encrypted Session ID

- **URL:** http://natas19.natas.labs.overthewire.org
- **Username:** natas19
- **Password:** (from Level 18 output)

🔍 Approach:

- PHPSESSID is Base64 encoded.

✓ Solution:

- Brute-force Base64 encodings of numbers (like MTox = 1:admin)
- Find the one where admin=1

🛠 Tools Used:

- **Base64 encoder/decoder**
- **Python script**

□ Logic:

- Encoding predictable data (e.g., userid:admin) into Base64 can be reversed.

🔒 NATAS Level 20: Race Condition in File Lock

- **URL:** http://natas20.natas.labs.overthewire.org
- **Username:** natas20
- **Password:** (from Level 19 output)

🔍 Approach:

- Application saves and reads messages from a file.

✓ Solution:

- Race condition: Submit request, and in another thread, read the file before it's saved.

Tools Used:

- Python with multithreading

Logic:

- Exploiting time gap between writing and reading allows info disclosure.
-
-



NATAS Level 20: Race Condition in File Lock

- **URL:** <http://natas20.natas.labs.overthewire.org>
Username: natas20
Password: (from Level 19 output)
 - **Approach:**
The application saves a message in a temporary session file and reads it back, but with improper locking.
 - **Solution:**
Send two rapid requests: one to set the message, and another to read it—exploiting a **race condition** to retrieve another user's (admin's) session.
 - **Tools Used:**
Python with `requests` and `threading` modules
 - **Logic:**
File is read before the session write completes, leaking other users' data.
-



NATAS Level 21: Session File Disclosure

- **URL:** <http://natas21.natas.labs.overthewire.org>
Username: natas21
Password: (from Level 20 output)
 - **Approach:**
Two subdomains share the same session mechanism. Admin uses one domain, and you access another.
 - **Solution:**
Find session ID from one subdomain (e.g., `experimenter.natas21...`) and reuse it on the main domain to gain elevated access.
 - **Tools Used:**
Browser cookies / Developer Tools
Python `requests` module
 - **Logic:**
Session fixation via shared session ID between two subdomains.
-



NATAS Level 22: Log Injection + LFI

- **URL:** <http://natas22.natas.labs.overthewire.org>
Username: natas22
Password: (from Level 21 output)
- **Approach:**
Redirection is used to block access.
- **Solution:**
Use Python or curl to prevent automatic redirection and read the password from the first response.
- **Tools Used:**
`curl` with `-L` disabled
Python `requests` with `allow_redirects=False`
- **Logic:**
Redirection hides content—manual control allows reading hidden messages.

-  **NATAS Level 23: Eval() Code Injection**
- **URL:** <http://natas23.natas.labs.overthewire.org>
Username: natas23
Password: (from Level 22 output)
-  **Approach:**
Password is compared using `strcmp`, and PHP type juggling can be abused.
-  **Solution:**
Send password as array (e.g., `?passwd[]`). `strcmp()` expects a string and fails.
-  **Tools Used:**
Browser / curl / Python
-  **Logic:**
PHP will throw a warning and skip logic if the input isn't a string—shortcircuiting validation.

-  **NATAS Level 24: Backdoor via User-Agent**
- **URL:** <http://natas24.natas.labs.overthewire.org>
Username: natas24
Password: (from Level 23 output)
-  **Approach:**
App uses `preg_match()` and reads headers like User-Agent.
-  **Solution:**
Bypass regex using malformed headers or inject PHP-like payload in User-Agent to exploit server response.
-  **Tools Used:**
curl or Python with custom headers
-  **Logic:**
Loosely written regex and unsafe usage of `preg_match()` on headers creates a backdoor.

-  **NATAS Level 25: Loose Type Comparison**
- **URL:** <http://natas25.natas.labs.overthewire.org>
Username: natas25
Password: (from Level 24 output)
-  **Approach:**
Includes a file based on your name input.
-  **Solution:**
Create a custom PHP file (e.g., in log files) and access it via the vulnerable `include()` statement.
-  **Tools Used:**
Curl with forged headers (for log injection)
-  **Logic:**
Injection + file inclusion = remote code execution.

-  **NATAS Level 26: Hidden Backdoor File**
- **URL:** <http://natas26.natas.labs.overthewire.org>
Username: natas26
Password: (from Level 25 output)
-  **Approach:**
Upload function saves serialized PHP object to disk.
-  **Solution:**
Craft a custom object with a `__destruct` method that runs code (RCE via serialization).

- **Tools Used:**
Custom PHP class and serialization
Python for base64 encoding
 - **Logic:**
Deserialization of untrusted input = arbitrary code execution.
-

• **NATAS Level 27: XOR Obfuscation**

- **URL:** <http://natas27.natas.labs.overthewire.org>
Username: natas27
Password: (from Level 26 output)
 - **Approach:**
Usernames are obfuscated using XOR encryption.
 - **Solution:**
Create a user with special characters that decode into the target username (`admin`) via XOR collisions.
 - **Tools Used:**
Python XOR script
 - **Logic:**
Understanding XOR logic and using it to forge keys that match desired output.
-

• **NATAS Level 28: XOR with Known Key**

- **URL:** <http://natas28.natas.labs.overthewire.org>
Username: natas28
Password: (from Level 27 output)
 - **Approach:**
Application uses XOR and base64 to obfuscate data.
 - **Solution:**
Use a known plaintext attack to derive the XOR key, then decrypt the real data.
 - **Tools Used:**
Python (XOR + base64 decode)
 - **Logic:**
Plaintext attack + XOR = reverse the encryption logic.
-

• **NATAS Level 29: JSON-Based Encryption**

- **URL:** <http://natas29.natas.labs.overthewire.org>
Username: natas29
Password: (from Level 28 output)
 - **Approach:**
JSON object is base64 encoded and XOR encrypted.
 - **Solution:**
Modify fields like `"admin": false` to `"admin": true`, encrypt and re-encode with same key.
 - **Tools Used:**
Python script for JSON, XOR, base64
 - **Logic:**
XOR is symmetric; flip bits and control logic.
-

• **NATAS Level 30: Known Plaintext Attack**

- **URL:** <http://natas30.natas.labs.overthewire.org>
Username: natas30
Password: (from Level 29 output)

- **Approach:**
Encrypted password check using XOR.
 - **Solution:**
Re-use known plaintext to guess cipher key and decrypt password field.
 - **Tools Used:**
Python XOR analysis
 - **Logic:**
With enough known plaintext, XOR encryption becomes predictable.
-

• **NATAS Level 31: Server-Side Verification**

- **URL:** <http://natas31.natas.labs.overthewire.org>
Username: natas31
Password: (from Level 30 output)
 - **Approach:**
Server decrypts and verifies user-controlled input.
 - **Solution:**
Replay XOR-encoded payload, modifying minimal bytes to flip a value (e.g., `false` → `true`).
 - **Tools Used:**
Python encryption logic
 - **Logic:**
Bit-flipping in block cipher or XOR-based logic.
-

• **NATAS Level 32: PHP Variable Injection**

- **URL:** <http://natas32.natas.labs.overthewire.org>
Username: natas32
Password: (from Level 31 output)
 - **Approach:**
PHP uses `extract()` function to parse user input.
 - **Solution:**
Send a payload like `?__FILE__=index.php` or override internal vars to leak or manipulate logic.
 - **Tools Used:**
Browser or curl
PHP payload knowledge
 - **Logic:**
Using `extract()` with `$_GET` or `$_POST` is insecure.
-

• **NATAS Level 33: Multi-Stage Encryption**

- **URL:** <http://natas33.natas.labs.overthewire.org>
Username: natas33
Password: (from Level 32 output)
 - **Approach:**
Data is encoded in multiple layers.
 - **Solution:**
Reverse layers of encoding: base64 → XOR → JSON
Rebuild and re-encode with elevated permissions.
 - **Tools Used:**
Python decoder stack
 - **Logic:**
Multi-step decoder logic replicates the app's processing.
-

• NATAS Level 34: Final Challenge

- **URL:** `http://natas34.natas.labs.overthewire.org`

Username: natas34

Password: (from Level 33 output)

• Approach:

Final level combines all techniques: obfuscation, injection, encoding.

• Solution:

Reverse-engineer application source, identify encoding/logic flow, and craft custom payload.

• Tools Used:

Python, Burp Suite, Source review

• Logic:

Understanding the cumulative security concepts of all previous levels.

OverTheWire Leviathan: Comprehensive Walkthrough

◆ Level 0 → 1

- **Objective:* ◻ Locate the password for `leviathan1.◻◻

- **Tools Used:* ◻ls, cd, cat, grep◻◻

- **Solution Logic:** 1 ◻ Access the .backup directory using `cd .backup.` 2 ◻ Identify the `bookmarks.html` file. 3 ◻ Use `grep` to search for the keyword "password" within the file: ◻

- `grep "password" bookmarks.html`

4 ◻ Extract the password from the line containing i.◻◻

- **Password:* ◻rioGegei8◻◻
-

◆ Level 1 → 2

- **Objective:* ◻ Discover the password for leviathan.◻◻

- **Tools Used:* ◻file, ltrace◻◻

- **Solution Logic:** . ◻ Identify the `check` binary using `ltrace ./check`. ◻ Confirm it's an executable with `file check`. ◻ . ◻ Use `ltrace` to trace library calls and identify the password comparison: ◻

- `ltrace ./check`

. ◻ Note the password it compares against and use it to execute the binary.◻◻

- **Password:* ◻ougahZi8a◻◻
-

◆ Level 2 → 3

- **Objective** ◻ Gain access to `leviathan3.`◻◻

- **Tools Used** ◻ltrace, touch, mkdir, bsh ◻◻

- **Solution Logic:**

1. ◻ Identify the `printfile` binary.◻

2. □ Use ltrace to observe its behavior: □
3. ltrace ./printfile
4. □ Create a file with a name that includes a command injection, such as test; bash. □
5. □ Execute the binary with the crafted filename to spawn a shell. □ □

- **Password:** □ Ahdiemo1j □ □
-

◆ Level 3 → 4

- **Objectiv:** □ Retrieve the password for leviathn4. □ □
- **Tools Use:** □ ltrace □ □
- **Solution Logic:**

1. □ Identify the level3 binary. □
2. □ Use ltrace to monitor its behavior and identify the password it expects: □
3. ltrace ./level3
4. □ Input the correct password to gain access. □ □

- **Passwor:** □ vuH0cox6m □ □
-

◆ Level 4 → 5

- **Objectie:** □ Find the password for leviatan5. □ □
 - **Tools Usd:** □ ls, cd, ./bin, binary-to-ASCII conversion □ □
 - **Solution Logic:** 1. □ Navigate to the .trash directory. □ 2. □ Execute the bin file to obtain binary output. □ 3. □ Convert the binary output to ASCII to reveal the password. □ □
 - **Passwod:** □ Tith4okei □ □
-

◆ Level 5 → 6

- **Objective:** □ Access levian6. □ □
 - **Tools Ued:** □ ln, symboli links □ □
 - ***Solution Logic:**
 1. □ Create a symbolic link in /tmp pointing to the password file: □
 2. ln -s /etc/leviathan_pass/leviathan6 /tmp/file.log
 3. ``
 4. □ Execute the leviathan5 binary, which reads from /tmp/file.log, to display the password. □ □
 - **Passwrd:** □ Ugaoee4li □ □
-

◆ Level 6 → 7

- **Objecive:** □ Obtain the password for levithan7. □ □
- **Tools sed:** □ Python scripting, brute-force pproach □ □
- ***Solution Logic:**
 1. □ Develop a Python script to brute-force the 4-digit PIN required by the levian6binary □

2. ☐ Iterate through possible combinations until the correct PIN is found and access is granted. ☐ ☐
- **Password:** ☐ahyMaeBo9☐☐
-

◆ Level 7 → 8

- **Objective:** ☐ Complete the finl level. ☐ ☐
 - **ToolsUsed:** ☐strings, grep` ☐ ☐
 - **Solution Logic**
 1. ☐ Use strings to extract readable strings from the `leviathan7 binary: ☐
 2. strings leviathan7
 3. ``
 4. ☐ Identify any hardcoded passwords or hints within th outpu. ☐
 5. ☐ Use the discovered information to gai access. ☐ ☐
- **Password:** ☐loVzz6mT☐☐
-