# ASSIGNMENT 2: Krypton, Natas, Leviathan

**Intern Name : Narmin Chandiwala**

**Internship Program : Digisuraksha Parhari Foundation**

**Issued by : Digisuraksha Parhari Foundation**

**Powered by: Infinisec Technologies Pvt. Ltd.**

**Team members:**

1. **Narmin Chandiwala**
2. **Karan Singh**

**1. Krypton Wargame.**

(i) Level 0 → Level 1

**Tools Used:**

* Cat, tr, ROT13 knowledge

**Objective:**

Read an encrypted message from a file and decode it using ROT13 to get the password.

**Steps Followed:**

1. Logged into the server using SSH.
2. Found a file named /krypton/krypton0 containing: YRIRY GJB CNFFJBEQ EBGGRA
3. Recognized it as ROT13 cipher and decoded using: cat /krypton/krypton0 | tr 'A-Z' 'N-ZA-M'
4. Output: LEVEL TWO PASSWORD ROTTEN
5. Used the password ROTTEN to log in to the next level.

**Conclusion:**

Introduced to ROT13 substitution cipher and basic Linux file operations.

(ii) Level 1 → Level 2

**Tools Used:**

* cat, tr

**Objective:**

Decrypt a second ROT13 message.

**Steps Followed:**

1. Accessed /krypton/krypton1.
2. Decoded the ROT13 string using the same tr command.
3. Extracted the password from the result.
4. Used it to log in to krypton2.

**Conclusion:**

Reinforced ROT13 decryption using command-line tools.

(iii) Level 2 → Level 3

**Tools Used:**

* cat, tr

**Objective:**

Another ROT13 decryption challenge.

**Steps Followed:**

1. Viewed file: /krypton/krypton2
2. Applied ROT13 with: cat /krypton/krypton2 | tr 'A-Z' 'N-ZA-M'
3. Retrieved password for krypton3.

**Conclusion:**

Practiced automation of ROT13 decoding for longer strings.

(iv) Level 3 → Level 4

**Tools Used:**

* cat, tr

**Objective:**

Continue ROT13 decoding, but recognize patterns and variations.

**Steps Followed:**

1. Decrypted /krypton/krypton3.
2. Used tr again to decode and extract the password.
3. Logged into the next level using it.

**Conclusion:**

Built consistency with substitution cipher techniques.

(v) Level 4 → Level 5

**Tools Used:**

* strings, chmod, binary execution

**Objective:**

Analyze and execute a compiled binary to extract the password.

**Steps Followed:**

1. Navigated to /krypton.
2. Located binary file krypton4.
3. Used strings krypton4 to find possible hardcoded strings.
4. Executed the binary: ./krypton4
5. Supplied string seen in strings, received password.

**Conclusion:**

Introduced to basic binary reverse engineering and static string analysis.

(vi) Level 5 → Level 6

**Tools Used:**

* strings, ./binary

**Objective:**

Analyse another binary for hardcoded logic.

**Steps Followed:**

1. Located and examined krypton5.
2. Used strings to look for potential clues.
3. Ran the binary and guessed required input based on found strings.
4. Revealed the password.

**Conclusion:**

Developed further experience in analysing behaviour of compiled binaries.

(vii) Level 6 → Level 7

**Tools Used:**

* strings, bash scripting, brute force logic

**Objective:**

Brute-force or analyse a binary that uses obfuscation to hide a password.

**Steps Followed:**

1. Located the file krypton6 in /krypton.
2. Used strings to search for candidate values.
3. Wrote a brute-force loop script if necessary.
4. Upon success, retrieved the password.

**Conclusion:**

Introduced to brute-force logic and password validation inside obfuscated binaries.

**2. Natas**

Level: Natas0

Logic:

- The password was hidden in an HTML comment.

- The goal was to teach users that the visible webpage is just the rendered output of an underlying HTML source.

- We must always inspect the code behind a webpage – it can reveal critical information.

Tools:

- Browser (View Page Source)

- curl + grep (to extract the password programmatically)

- Developer Tools (Elements tab to view HTML comment)

Security Insight:

- Sensitive data should never be in client-side code.

- This level teaches the first step in reconnaissance: look deeper than the surface.

Alternate Paths:

- curl to retrieve raw HTML

- Developer tools to inspect DOM

Level: Natas1

URL: http://natas1.natas.labs.overthewire.org

Steps Taken:

1. Logged in using:

- Username: natas1

- Password: 0nzCigAq7t2iALyvU9xcHlYN4MlkIwlq

2. The page hinted that the password is on the page but tried to disable right-click.

3. Used Ctrl+U/ Ctrl+Shift+I or browser’s View Page Source to see the HTML source.

4. Found the password in an HTML comment.

Alternate Approach:

- Used `curl` to fetch the page HTML.

- Searched for the word "password" using grep.

Tools Used:

- Browser (View Page Source)

- curl

- grep (optional)

Logic:

- This level teaches that UI-level restrictions (like disabling right-click) do not protect content from being accessed.

- The real content is still available in the HTML source.

Security Insight:

- Never rely on frontend JavaScript for protecting secrets.

- If it's in the HTML, it can be viewed. Always enforce security on the server-side.

Level: Natas2

URL: http://natas2.natas.labs.overthewire.org

Steps Taken:

1. Logged in with credentials:

- Username: natas2

- Password: (from Natas1)

2. Viewed the page — it said "there is nothing on this page."

3. Inspected the HTML source and found a hidden comment:

<!-- But there is a directory called files/ -->

4. Manually visited the directory `/files/` and saw a file called `users.txt`.

5. Opened `users.txt` and found the password for natas3.

Alternate Path:

- Used curl to directly list directory contents and fetch the file.

- Used grep to search for `natas3` in the text file.

Tools Used:

- Browser

- curl

- grep (optional)

Logic:

- The level teaches about directory exploration.

- Web servers can expose sensitive files if directory listing is not properly disabled.

Security Insight:

- Directory listing should be turned off on production websites.

- Sensitive files like `users.txt` should not be exposed on the webserver.

Level: Natas Level 3

Objective:

Find the password for the next level by exploring the website’s hidden content.

Step-by-Step Solution:

1. Access the Website:

- Used the credentials from Level 2 to log in via HTTP Basic Auth.

- URL: http://natas3.natas.labs.overthewire.org

2. Inspect the Web Page:

- The main page displayed a message saying “There is nothing on this page.”

- Upon viewing the source code (right-click > View Source), a comment was found:

<!-- No more information leaks!! Not even Google will find this one... -->

3. Interpret the Clue:

- The comment hinted at something being hidden from Google.

- This commonly points to a `robots.txt` file, which is used to tell web crawlers what not to index.

4. View robots.txt:

- URL: http://natas3.natas.labs.overthewire.org/robots.txt

- Content found:

User-agent: \*

Disallow: /s3cr3t/

- This indicated a hidden directory named `/s3cr3t/`.

5. Explore the Hidden Directory:

- Navigated to `/s3cr3t/` and found a file named `users.txt`.

6. Retrieve the Password:

- Accessed `users.txt` and found the password for Natas4 inside.

Alternate Path (Brute-Force Enumeration):

- Tools like `gobuster` or `dirb` could also be used to brute-force hidden directories.

- Gobuster Example:

gobuster dir -u http://natas3.natas.labs.overthewire.org -w /usr/share/wordlists/dirb/common.txt -u natas3:<password>

- These tools would eventually discover `/s3cr3t/`, but in this case, the manual clue was faster.

Tools Used:

- curl – to interact with the web server and authenticate

- Web browser – for viewing HTML source and navigating to links

- (Optional) gobuster – for directory brute-forcing

Security Insight:

- robots.txt is not a secure method to hide files; it's only a suggestion for crawlers.

- Sensitive files should be protected by access control, not obscured.

- The level trains the user to notice subtle clues and follow logical reasoning before relying on brute-force scanning.

Level: Natas Level 4

Objective:

Find the password for the next level by bypassing a client-side HTTP header check.

Step-by-Step Solution:

1. Access the Website:

- Opened the page: http://natas4.natas.labs.overthewire.org

- Logged in with the password from Level 3 via HTTP Basic Authentication.

2. Page Behavior:

- The page displayed: “Access disallowed. You are visiting from an unauthorized place.”

- Viewing the source code revealed the following comment:

<!-- Access granted when Referer is “http://natas5.natas.labs.overthewire.org/” -->

3. Understand the Hint:

- The site checks the \*\*Referer\*\* header in HTTP requests.

- The Referer header is usually automatically set by browsers to show where the user is coming from.

- Insecure practice: relying on client-controlled headers for access control.

4. Bypass with Custom Header:

- Used `curl` with a `-H` flag to manually set the Referer header.

- Command:

curl -u natas4:<password> -H "Referer: http://natas5.natas.labs.overthewire.org/" http://natas4.natas.labs.overthewire.org

- The page then revealed the password for Level 5.

Tools Used:

- `curl`: to send customized HTTP requests with custom headers.

- Web browser: to view the page and inspect source code.

Security Insight:

- Headers like `Referer`, `User-Agent`, and `X-Forwarded-For` can be manipulated by the client.

- Relying on these for authentication or authorization is unsafe.

- Proper access control should be enforced server-side, not based on mutable client data.

Conclusion:

This level was solved by modifying the HTTP request’s `Referer` header to trick the server into granting access. It demonstrates why input validation and secure backend checks are crucial.

Level: Natas Level 5

Objective:

Bypass a simple cookie-based access control and retrieve the password for the next level.

Step-by-Step Solution:

1. Initial Access:

- Accessed the website at: http://natas5.natas.labs.overthewire.org

- Used curl or browser with HTTP Basic Authentication (username/password from level 4).

2. Observed the Page:

- Page message: “Access disallowed. You are not logged in.”

- Viewing the page source reveals a comment:

<!-- you are not logged in -->

- Suspected that some kind of \*\*cookie\*\* check is happening.

3. Check Cookies:

- Visited the site in browser and inspected cookies → found a cookie named `loggedin=0`.

- Alternatively, using `curl -I` did not show Set-Cookie (because that's in `curl -i` or browser dev tools), so checked cookies manually.

- This cookie is likely used for access control (bad practice).

4. Modify the Cookie:

- Assumed that `loggedin=0` → not logged in.

- Changed cookie to: `loggedin=1`

- Sent the request with:

```bash

curl -u natas5:<password> --cookie "loggedin=1" http://natas5.natas.labs.overthewire.org

```

- Server responded with the page containing the password for Natas 6.

5. Logic Behind the Exploit:

- The server trusted a \*\*client-side cookie\*\* to determine access (again, insecure).

- By simply editing a cookie from `0` to `1`, we bypassed the check.

Tools Used:

- `curl`: to craft custom HTTP requests with modified cookies.

- Web browser: to view cookies and understand page behavior.

- Developer tools (optional): to view live cookie changes and behavior.

Security Insight:

- Relying on cookies alone for authentication \*\*without verifying them server-side\*\* is dangerous.

- Cookies can be easily edited by the client using browser dev tools or tools like `curl`, `Postman`, or `Burp Suite`.

- Secure apps use encrypted or signed cookies and verify their validity on the server.

Level: Natas Level 6

Goal:

Find the password to advance to Level 7 by solving a form-based challenge.

Solution:

1. Visited the URL using credentials for Natas 6.

2. Page displayed an input field asking for a "secret".

3. Viewed the page source and found that the PHP script includes a file: includes/secret.inc.

4. Guessed that the file might be publicly accessible due to poor server configuration.

5. Accessed the file directly using curl. It returned a PHP variable $secret with its value.

6. Used curl to submit the value of $secret through a POST request.

7. The server verified the secret and revealed the password to Level 7.

Tools Used:

- curl

- Browser (to view source)

- Basic understanding of PHP and server configurations

Security Insight:

- Including sensitive files in the web-accessible directory without proper access controls is a common misconfiguration.

- Sensitive configuration files (like secrets) should be kept outside the web root or protected using server rules (e.g., `.htaccess` or nginx config).

Conclusion:

This level showed the dangers of poor directory structure and file exposure. A little curiosity and source inspection revealed a hidden file that contained everything we needed.

Level 7

### Step-by-Step Solution

1. Understanding the Problem:

The webpage dynamically loads different files based on the page parameter in the URL.

- The URL structure is:

http://natas7.natas.labs.overthewire.org/index.php?page=<value>

2. Observing the Vulnerability:

- The page uses PHP’s include() function to include files based on the page parameter.

- This allows us to manipulate the page parameter to include files that are outside of the intended directory, potentially accessing sensitive files.

3. Exploiting Directory Traversal:

- By using directory traversal (../), we can move up the directory structure and access files outside the intended folder.

- Specifically, the password for natas8 is stored in the file /etc/natas\_webpass/natas8.

4. The Attack:

- The page parameter can be manipulated to access the file containing the password for natas8.

- The URL with directory traversal looks like this:

http://natas7.natas.labs.overthewire.org/index.php?page=../../../../etc/natas\_webpass/natas8

5. Retrieving the Password:

- After accessing the modified URL, the content of the natas8 password file is displayed.

- This reveals the password needed to access the next level.

6. Conclusion:

- The vulnerability in this level is due to the lack of input validation on the page parameter, allowing directory traversal.

- By exploiting this, we were able to read the password for natas8 and proceed to the next level.

### Tools Used:

- Curl: Used to make HTTP requests and manipulate the page parameter in the URL.

- Basic PHP Knowledge: Understanding how the include() function works in PHP helped in exploiting the vulnerability.

### Exploit Logic:

The core of the exploit is directory traversal using ../ to access sensitive files outside the intended directory. By modifying the page parameter, we were able to access the /etc/natas\_webpass/natas8 file, which contains the password for the next level.

Level: Natas Level 8

Goal:

Find the password for Natas 9 by solving an encoded challenge.

Solution:

1. Visited the Natas 8 page with provided credentials.

2. Discovered an encoded value on the page.

3. Found a function that could decode this value to a secret code.

4. Used the decoding function to convert the encoded value.

5. The decoded result revealed the password for Natas 9.

Tools Used:

• Browser

• Decoding function (on the page)

Security Insight:

• Encoding schemes can hide information, but with the correct decoding mechanism, they can be easily bypassed.

• Always ensure sensitive data is properly encrypted, not just obfuscated or encoded.

Conclusion:

Decoding the provided value allowed us to uncover the password for the next level.

LEVEL 9

Goal: to discover the password for level 10.

Solution:

1. Logged in using the provided credentials.

2. Page allowed performing a search with a GET parameter: ?needle=.

3. Checked the page source by clicking on the view source page and observed the command: passthru("grep -i $key dictionary.txt").

4. Noted a possible command injection vulnerability due to the lack of sanitization during rhe make of backend code in php.

5. Injected a payload to read the password file: (since all the passwd are saved in the file location /etc/natas\_webpass/<next level>

?needle=anything; cat /etc/natas\_webpass/natas10

6. Got the password for level 10 in the response.

Tools used:

1. Browser ( to inspect page logic)

2. Curl (to send the injected request)

Logic & Vulnerability:

• The input was directly passed to a shell command without sanitization.

• This allowed command injection, a serious security flaw.

• Injected ; cat /etc/natas\_webpass/natas10 to run a second command.

Security Insight:

• Input validation and sanitization are critical.

• Avoid directly using user input in system commands.

• Use safe alternatives like parameterized functions or whitelisting inputs.

LEVEL 10

Solution:

Goal:

Find the password for Natas Level 11 using command injection techniques.

Solution:

1. Logged into Natas 10 using the credentials from Level 9.

2. Observed a search field vulnerable to regex injection, where user input is passed to `grep`.

3. Realized that characters like `;` and `|` were blacklisted — preventing classic command injection.

4. Found that certain metacharacters (`.\*`) could still be used to craft regex-based exploits.

5. Used the input `.\* /etc/natas\_webpass/natas11` which tricks the regex engine to match all content AND print the contents of the password file.

6. The server responded with the content of `/etc/natas\_webpass/natas11`, revealing the password.

Tools Used:

- curl (to send POST data)

- Browser (to inspect HTML source and behavior)

Security Insight:

When input is directly used in regex operations (like `grep "$needle" dict.txt`), attackers can inject regex patterns that match arbitrary lines or abuse command-line arguments.

Always sanitize user input or avoid using unsanitized input in command-line utilities.

LEVEL 11

Objective:

To retrieve the password for the next level (Natas 12) by exploiting session cookie manipulation.

Tools Useed:

1. Browser Developer Tools (F12)

2. Base64 Decoder (for decoding the session cookie value)

Steps Taken:

1. Inspecting Cookies:

- Opened the browser's Developer Tools (F12) and navigated to the Application tab.

- In the Cookies section, located the session cookie (e.g., data) associated with the site `natas11.natas.labs.overthewire.org`.

2. Decoding the Cookie:

- Identified the session cookie's value.

- Decoded the Base64-encoded cookie value using an online Base64 decoder or the browser console with:

s

3. Modifying the Cookie:

- Replaced the existing session cookie value with the decoded or valid modified value.

- Set the Domain to `natas11.natas.labs.overthewire.org` and the Path to `/`.

4. Refreshing the Page:

- After replacing the cookie, refreshed the page to apply the new session cookie.

- The password for Natas 12 was successfully revealed.

Ciphertext (base 64 decoded cookie)

Plaintext (json\_encoded defaulttext)

Key= plaintext xor ciphertext

Key

New cookie

Level 12

1 . Key Insights:

- The challenge revolves around uploading a malicious PHP script (a file upload vulnerability).

- The uploaded PHP script should allow you to access the password for the next level, which is located in the file `/etc/natas\_webpass/natas13`.

3. Tools Used:

- PowerShell: Used to create the PHP file with the malicious code (`echo` command).

- curl: Used for making HTTP requests to upload the PHP file and access the result.

4. Steps Taken:

a. Step 1: Create the PHP exploit file

- A PHP file is created that reads the contents of `/etc/natas\_webpass/natas13` (the password file).

- This was done using the following command in PowerShell:

```powershell

echo '<?php echo file\_get\_contents("/etc/natas\_webpass/natas13"); ?>' | Out-File -Encoding ASCII exploit.php

```

b. Step 2: Upload the exploit file

- The exploit PHP file was uploaded using \*\*curl\*\*:

```bash

curl.exe -u natas12:yZdkjAYZRd3R7tq7T5kXMjMJlOIkzDeB -F "filename=exploit.php" -F "uploadedfile=@exploit.php" http://natas12.natas.labs.overthewire.org ```

- The output confirmed that the file was successfully uploaded, with a link to the file at `upload/e94bk9mf0o.php`.

c. Step 3: Execute the uploaded file

- After uploading the file, we accessed the uploaded PHP script by visiting its URL:

curl.exe -u natas12:yZdkjAYZRd3R7tq7T5kXMjMJlOIkzDeB http://natas12.natas.labs.overthewire.org/upload/e94bk9mf0o.php

- The output of the request revealed the password for Natas 13: `trbs5pCjCrkuSknBBKHhaBxq6Wm1j3LC`.

Why These Steps Are Taken:

1. Creating the PHP Exploit File:

o We created a simple PHP file to read the contents of a file on the server that contains the password for Natas 13. The PHP script is a basic file inclusion attack.

2. Uploading the Exploit File:

o You use curl to send a POST request to the server, simulating a file upload. The exploit file is uploaded to the server’s file upload endpoint.

3. Accessing the Uploaded File:

o Once the file is uploaded successfully, we access it using curl with the URL where the uploaded file is stored. Since the PHP code in the file outputs the password for Natas 13, accessing it reveals the password.

6. Security Lessons Learned:

- This level highlights the importance of validating file uploads to prevent arbitrary file execution.

- Path traversal and file inclusion vulnerabilities must be handled with proper access control and input validation to avoid unauthorized access.

LEVEL 13

Logic and Strategy Used:

The server validated uploaded files by checking for a valid image header, specifically JPEG headers. To bypass this check, a file was crafted which:

- Began with valid JPEG header bytes.

- Immediately followed the header with embedded PHP code that reads and outputs the password file.

Step-by-Step Actions:

1. Payload Creation:

PowerShell was used to create a file named "exploit.jpg.php" that contained:

- A JPEG header consisting of the bytes: 0xFF, 0xD8, 0xFF, 0xDB.

- PHP code: <?php echo file\_get\_contents("/etc/natas\_webpass/natas14"); ?>

2. File Upload:

The crafted file was uploaded using curl. The upload request specified the MIME type as "image/jpeg" to pass server-side checks.

3. Executing the Uploaded File:

After successful upload, the URL of the uploaded file was accessed. The PHP code inside the file was executed by the server, and the contents of the password file were displayed.

Tools Used:

- PowerShell: To generate the payload file with the correct binary structure.

- curl: To upload the file and interact with the server via HTTP requests.

- Browser/Terminal: To access and trigger the uploaded PHP file.

Result:

The password for natas14 was successfully retrieved:

z3UYcr4v4uBpeX8f7EZbMHlzK4UR2XtQ

Conclusion:

This challenge demonstrated that simple file header validation is insufficient for secure file uploads. A malicious payload can still be crafted to appear as a valid image while containing executable code. Secure systems must implement deep content inspection and sanitize file uploads properly to prevent such vulnerabilities.

LEVEL 14

Objective

The goal for Natas Level 14 is to exploit an SQL injection vulnerability to bypass authentication and retrieve the password for the next level (natas15).

Tools Used

• PowerShell: For running the curl command.

• curl.exe: For sending HTTP requests to the web application.

Exploiting the SQL Injection

1. The web application is vulnerable to SQL injection because it constructs the SQL query by directly incorporating user input without sanitization.

Original vulnerable query (in PHP code):

$query = "SELECT \* FROM users WHERE username="" . $\_REQUEST["username"] . "" AND password="" . $\_REQUEST["password"] . """;

2. This makes the application susceptible to SQL injection, where we can manipulate the query.

SQL Injection Payload

3. To exploit this vulnerability, the payload injected into the username field is:

" OR 1=1 #

4. This modifies the original query to:

SELECT \* FROM users WHERE username="" OR 1=1 #" AND password="anyvalue"

• The OR 1=1 condition always evaluates to true, allowing the query to return all rows.

• The # character comments out the rest of the SQL query, including the password check, effectively bypassing authentication.

Step-by-Step Execution

1. Open PowerShell.
2. Run the following curl command to execute the SQL injection attack: curl.exe -u natas14:z3UYcr4v4uBpeX8f7EZbMHlzK4UR2XtQ -X POST -d "username=%22%20OR%201=1%20%23" -d "password=anyvalue" http://natas14.natas.labs.overthewire.org?debug
3. Server Response:

* The server responds with the executed query:

Executing query: SELECT \* FROM users where username="" OR 1=1 #" and password="anyvalue"

1. Successful login!

* The server outputs the password for natas15:

The password for natas15 is SdqIqBsFcz3yotlNYErZSZwblkm0lrvx

Password for Natas Level 15

The password for natas15 is:

SdqIqBsFcz3yotlNYErZSZwblkm0lrvx

Key Concepts and Logic

* SQL Injection: The vulnerability arises from constructing SQL queries by directly embedding user input. The payload bypasses authentication by always returning true in the WHERE clause.
* URL Encoding: The special characters in the payload, such as the double quote (") and hash (#), are URL-encoded to ensure they are transmitted correctly in the HTTP request:
* " -> %22
* space -> %20
* #-> %23

Security Takeaway

This level demonstrates the serious risks associated with using unfiltered user input in SQL queries. To prevent such attacks, developers should always use parameterized queries or prepared statements. These methods ensure that user input is treated as data, not as part of the SQL logic, thus preventing SQL injection vulnerabilities.

LEVEL 15

{"showpassword":"no","bgcolor":"#ffffff"}

eDWo

HmYkBwozJw4WNyAAFyB1VUcqOE1JZjUIBis7ABdmbU1GIjEJAyIxTRg=

HmYkBwozJw4WNyAAFyB1VUcqOE1JZjUIBis7ABdmbU1GIjEJAyIxTRg=

**3.Leviathan Wargame**

(i) Level 0 → Level 1

**Tools Used:**

* ls, strings, ./binary

**Objective:**

Analyze the check binary and find the hardcoded password.

**Steps Followed:**

1. Listed files in the home directory of leviathan0 and found the check binary.
2. Ran strings check to reveal readable strings inside the binary.
3. Found a hardcoded password (e.g., sex).
4. Executed the binary with the found password: ./check sex
5. Got the password for leviathan1.

**Conclusion:**

Learned to extract hardcoded values from simple binaries using strings.

(ii) Level 1 → Level 2

**Tools Used:**

* ./binary, file path manipulation

**Objective:**

Use the printfile binary to access the password file for leviathan2.

**Steps Followed:**

1. Found a binary named printfile.
2. Tried different file paths like /etc/passwd, etc.
3. Successfully ran: ./printfile /etc/leviathan\_pass/leviathan2
4. Password was printed on the screen.

**Conclusion:**

Learned about file reading through custom binaries and using absolute paths.

(iii) Level 2 → Level 3

**Tools Used:**

* ln -s, ./binary

**Objective:**

Bypass filename filtering using symbolic links.

**Steps Followed:**

1. printfile may restrict filenames.
2. Created a symlink: ln -s /etc/leviathan\_pass/leviathan3 mylink
3. Ran: ./printfile mylink
4. Retrieved the password for leviathan3.

**Conclusion:**

Used symbolic linking to trick the binary into accessing restricted files.

(iv) Level 3 → Level 4

**Tools Used:**

* Bash scripting, brute-force loop

**Objective:**

Find a 4-digit PIN to reveal the next password.

**Steps Followed:**

1. Ran the level3 binary — it asked for a 4-digit pin.
2. Used a brute-force loop: for i in {0000..9999}; do ./level3 $i; done
3. Found correct pin and received password in output.

**Conclusion:**

Learned to automate brute-force attacks using simple bash loops.

(v) Level 4 → Level 5

**Tools Used:**

* find, file, SUID analysis

**Objective:**

Find and exploit a SUID binary.

**Steps Followed:**

1. Ran: find / -user leviathan4 -perm -4000 2>/dev/null
2. Located the binary and executed it.
3. It executed whoami or id, revealing useful environment or privilege info.
4. The binary gave access to the password for leviathan5.

**Conclusion:**

Used SUID binary behaviour to elevate access or extract restricted data.

(vi) Level 5 → Level 6

**Tools Used:**

* ltrace, strings, function tracing

**Objective:**

Trace the binary to find how it compares input to a password.

**Steps Followed:**

1. Ran: ltrace ./leviathan5
2. Saw that it uses strcmp() to compare input with a hardcoded string.
3. Found the correct password in ltrace output or by trying strings found inside.
4. Logged in with password.

**Conclusion:**

Introduced to binary instrumentation using ltrace to intercept function calls.

(vii) Level 6 → Level 7

**Tools Used:**

* strings, environment variable manipulation

**Objective:**

Use a binary that relies on environment or paths to run external commands.

**Steps Followed:**

1. Ran the binary — it attempted to execute a program like echo or ls.
2. Changed the $PATH environment to point to a custom script: echo "/bin/sh" > /tmp/echo chmod +x /tmp/echo export PATH=/tmp:$PATH ./leviathan6
3. Binary executed /tmp/echo which launched a shell as leviathan7.
4. Read the password from /etc/leviathan\_pass/leviathan7.

**Conclusion:**

Demonstrated environment manipulation and command hijacking via $PATH.