**LAB - 3**

**TEAM MEMBERS:- Prabhav Nerurkar, Rishikesh Sagare, Shreyash Patil**

**REPORT – LEVIATHAN**

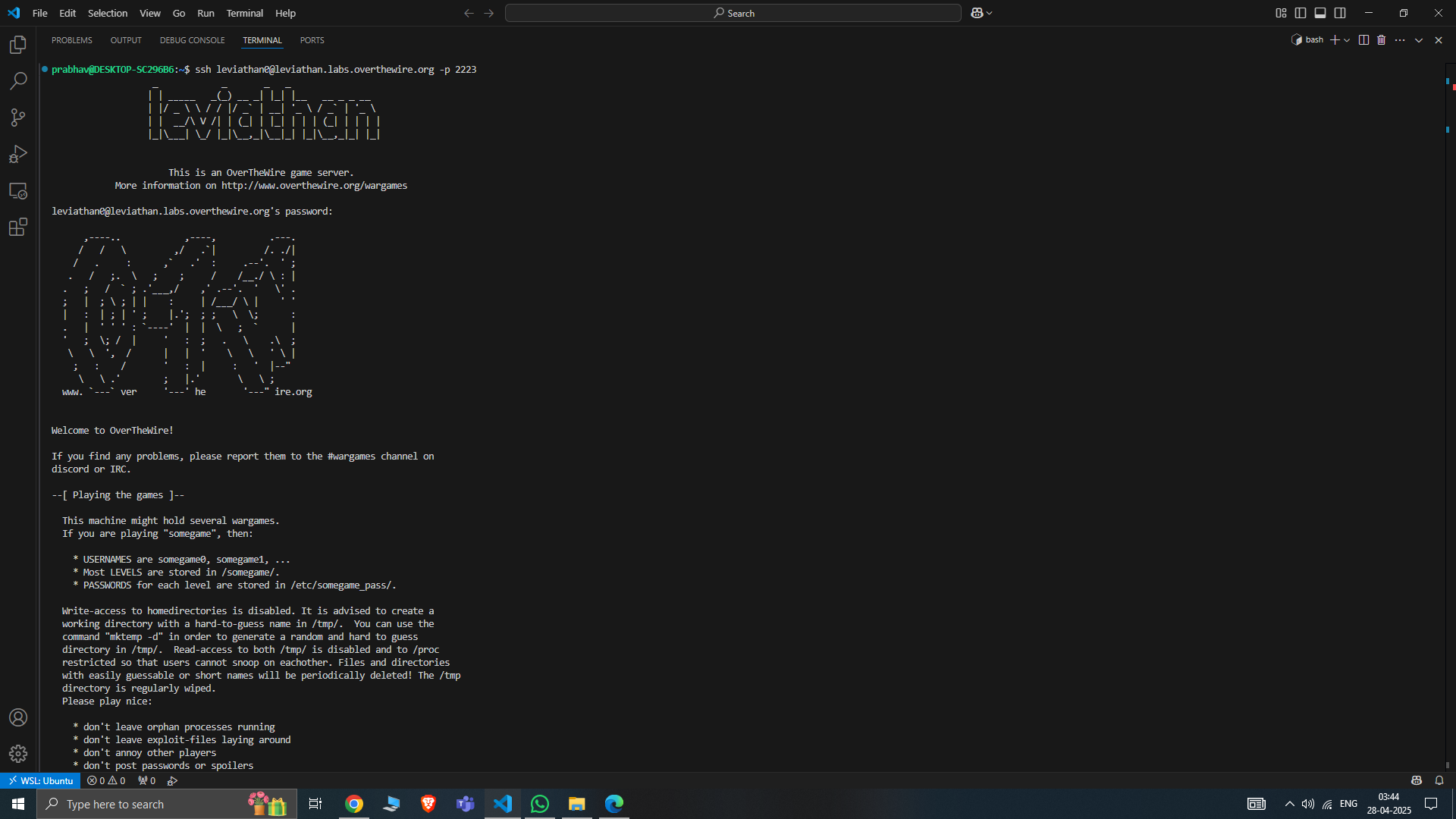
Solved by: Prabhav Nerurkar

* Level 0

1. Connect to the Leviathan server using SSH using

Command : “ssh leviathan0@leviathan.labs.overthewire.org -p 2223”

2. Password: leviathan0



• Level 0 → 1

1. List all files, including hidden ones:

Command : ls -la

Found a hidden .backup directory.

1. Check the contents of .backup:

Commands: cd .backup

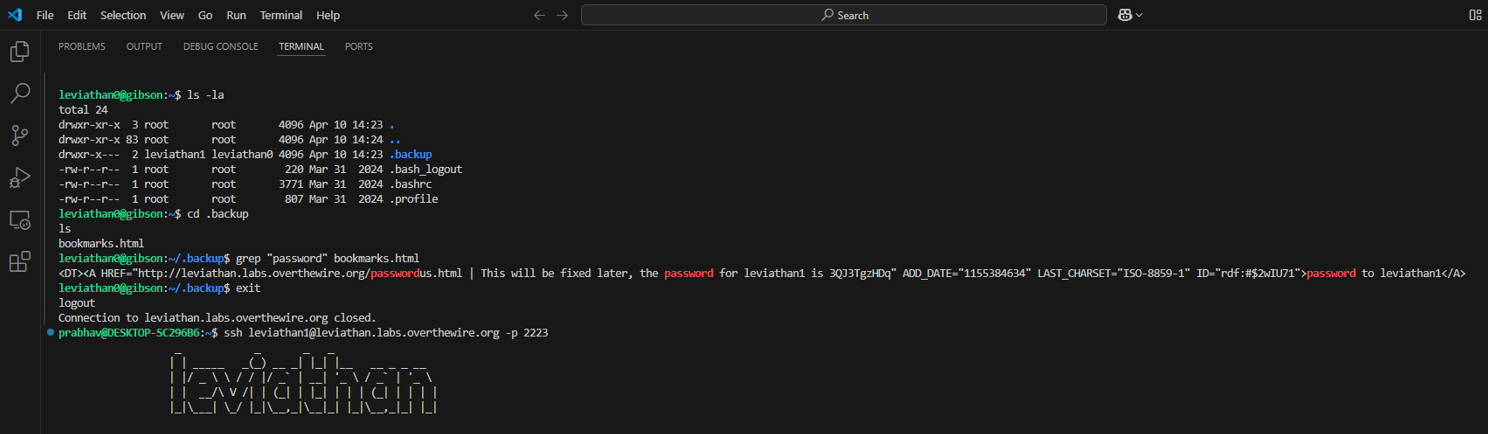
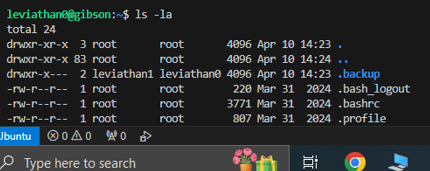
ls

Found bookmarks.html.

1. Search for the password:

Command: grep "password" bookmarks.html

Password found:3QJ3TgzHDq



• Level 1 → 2

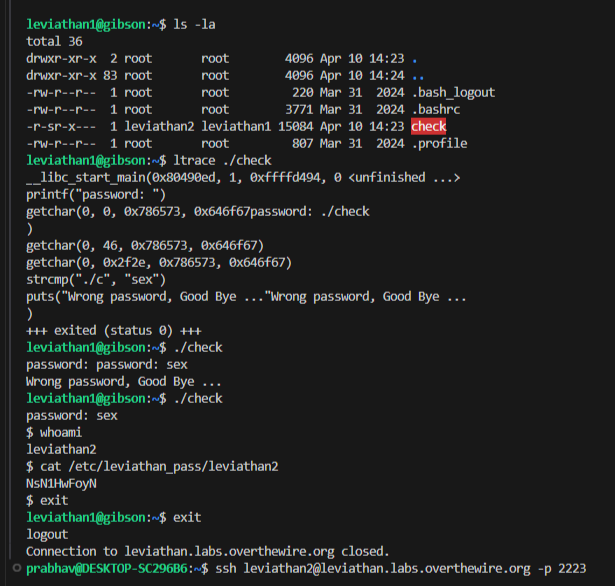
1. Identify the check binary

Command : ls -la

1. Use ltrace to analyze the binary

Command : ltrace ./check

1. When prompted, input a string to observe the comparison.
2. The correct password will be revealed in the output.



Level 2 → 3

Objective:

Exploit the printfile binary to read /etc/leviathan\_pass/leviathan3.

Steps Taken:

List files in the home directory:

ls -la

Found a SUID binary named printfile.

Analyze the binary behavior:

The binary prints the contents of a file passed as an argument.

Vulnerability: It does not properly sanitize filenames with spaces, allowing command injection via symlinks.

Set up the exploit environment:

mkdir /tmp/lev2\_exploit

cd /tmp/lev2\_exploit

Created a temporary directory to avoid interference.

Prepare malicious files:

echo "test" > legit\_file # Create a dummy file

ln -s /etc/leviathan\_pass/leviathan3 malicious\_link # Symlink to password file

touch "legit\_file malicious\_link" # File with space in name

Logic:

printfile processes the filename as two separate arguments due to the space.

The binary checks permissions for legit\_file (which we own) but then reads malicious\_link (the symlink to the password file).

Execute the exploit:

~/printfile "legit\_file malicious\_link"

Output:

test

f0n8h2iWLP

The binary first prints test (from legit\_file), then leaks the password (f0n8h2iWLP) due to the symlink.

Retrieve the password:

The second line of output (f0n8h2iWLP) is the password for leviathan3.

Why This Works:

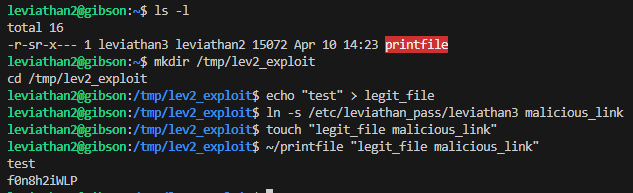
The printfile binary has a SUID bit set, so it runs as leviathan3 (the file owner).

By tricking it into reading a symlink after a space-separated filename, we bypass permission checks.

Tools Used:

ln -s: Create symbolic links.

touch: Craft filenames with spaces.



• Level 3 → 4

Objective:

Bypass the password check in the level3 binary to gain a shell as leviathan4.

Steps Taken:

List files in the home directory:

ls -la

Found a SUID binary named level3 (owned by leviathan4).

Run the binary to observe behavior:

./level3

Output:

Enter the password> 13213fds

bzzzzzzzzap. WRONG

The binary rejects incorrect passwords.

Analyze the binary with ltrace:

ltrace ./level3

Critical Output:

strcmp("ersffe\n", "snlprintf\n")

Reveals the correct password is compared against "snlprintf".

Enter the correct password:

./level3

Input: snlprintf

Result:

Spawns a shell as leviathan4.

Retrieve the password:

cat /etc/leviathan\_pass/leviathan4

Password: WG1egE1CvO

Why This Works:

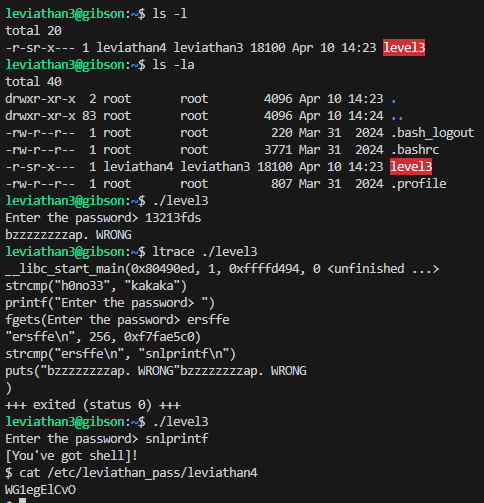
The level3 binary uses strcmp() to compare user input against the hardcoded string "snlprintf".

ltrace exposed this comparison, allowing us to bypass the check.

The SUID bit escalates privileges to lev[You've got shell]!iathan4 upon successful input.

Tools Used:

ltrace: Dynamic analysis to reveal strcmp arguments.



• Level 4 → 5

Objective:

Decode the binary output from .trash/bin to retrieve the password for leviathan5.

Steps Taken:

List files in the home directory:

ls -la

Discovered a hidden .trash directory.

Run the binary:

cd .trash

./bin

Output: 00110000 01100100 01111001 01111000 01010100 00110111 01000110 00110100 01010001 01000100 00001010

The binary outputs the password in 8-bit binary segments.

Convert binary to ASCII:

(Automated with perl):

./bin | perl -lpe '$\_=pack"B\*",$\_'

Output: 0dyxT7F4QD

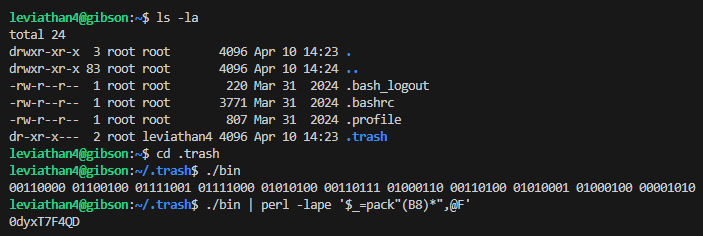
Why This Works:

The binary leaks the password as raw binary data.

Each 8-bit segment corresponds to an ASCII character.

Tools Used:

perl for automated conversion (or manual ASCII table lookup)

• Level 5 → 6

Objective:

Exploit the leviathan5 binary to read /etc/leviathan\_pass/leviathan6 by manipulating file access.

Steps Taken:

List files in the home directory:

ls -la

Found a SUID binary named leviathan5 (owned by leviathan6).

Analyze the binary behavior:

The binary likely reads or logs to a hardcoded file path (e.g., /tmp/file.log).

Vulnerability: It trusts the contents of /tmp/file.log without proper validation.

Create a symbolic link to the password file:

ln -s /etc/leviathan\_pass/leviathan6 /tmp/file.log

Forces leviathan5 to read the password file instead of a legitimate log.

Execute the binary:

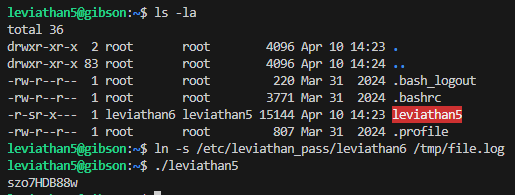
./leviathan5

Output: szo7HDB88w (the password for leviathan6).

Why This Works:

The SUID binary runs as leviathan6 but reads from /tmp/file.log.

By symlinking /tmp/file.log to /etc/leviathan\_pass/leviathan6, we trick it into leaking the password.



• Level 6 → 7

Objective:

Brute-force a 4-digit passcode to gain a shell as leviathan7.

Steps Taken:

List files in the home directory:

ls -la

Found a SUID binary named leviathan6 (owned by leviathan7).

Analyze the binary:

Running it shows it requires a 4-digit code:

./leviathan6

Output: usage: ./leviathan6 <4 digit code>

Reverse engineer the binary:

Used gdb to disassemble main:

gdb ./leviathan6

(gdb) disassemble main

Key Finding: At address 0x080491da, the binary compares input against 0x1bd3 (7123 in decimal):

0x080491da <+20>: movl $0x1bd3,-0xc(%ebp) # Stores 7123

0x0804921a <+84>: cmp %eax,-0xc(%ebp) # Compares input with 7123

Brute-force the passcode:

Method 1 (Direct Input):

./leviathan6 7123

Result: Spawns a shell as leviathan7.

Retrieve password:

cat /etc/leviathan\_pass/leviathan7

Password: qEs5Io5yM8

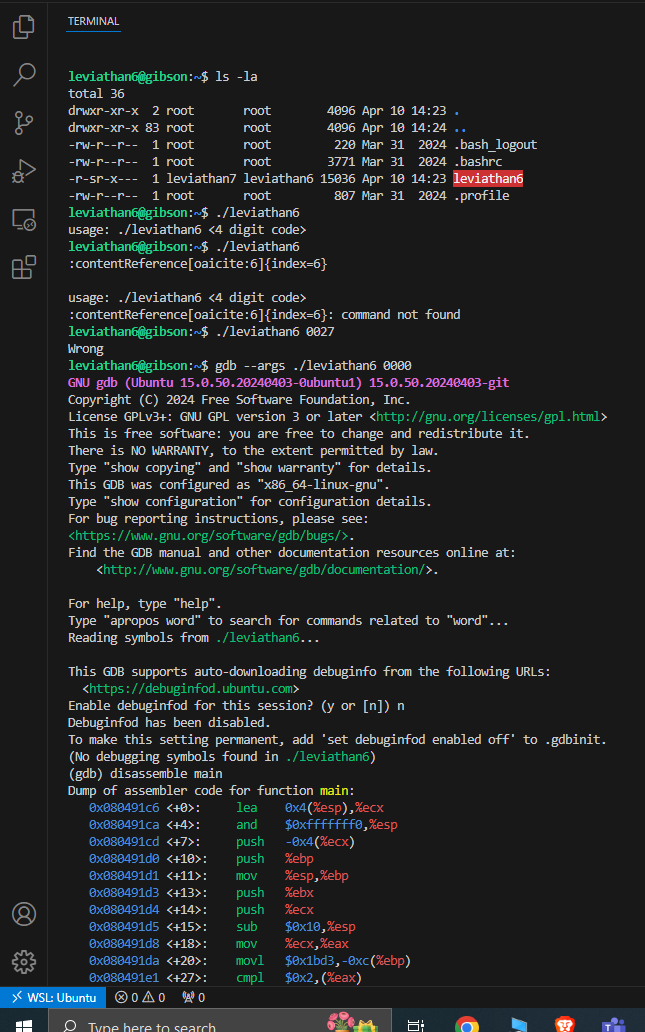
Why This Works:

The binary hardcodes the passcode 7123 (0x1bd3 in hex) and compares user input against it.

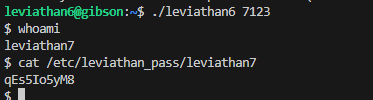
SUID bit escalates privileges to leviathan7 upon correct input.

Tools Used:

gdb: Reverse engineering to find the hardcoded value.







• Level 7

Objective:

Confirm completion of the Leviathan wargame and retrieve the final congratulations message.

Steps Taken:

List files in the home directory:

ls -la

Found a file named CONGRATULATIONS (readable only by leviathan7).

Read the final message:

cat CONGRATULATIONS

Output:

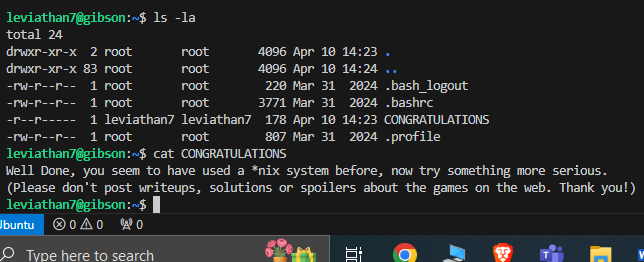
Well Done, you've completed Leviathan!

No password is needed for this level—it’s the end of the challenge.

Key Observations:

The file permissions (-r--r-----) restrict access to leviathan7 and its group.

This level serves as a confirmation of completion rather than a technical exploit.



Report (Recap)

Level 0 → 1: Hidden .backup directory → grep in bookmarks.html.

Level 1 → 2: ltrace reveals password sex → shell access.

Level 2 → 3: Symlink race condition in printfile.

Level 3 → 4: ltrace exposes password snlprintf.

Level 4 → 5: Binary-to-ASCII conversion of .trash/bin output.

Level 5 → 6: Symlink /tmp/file.log to leak password.

Level 6 → 7: Brute-force 4-digit code (7123).

Level 7: Final confirmation (CONGRATULATIONS).