

# **Hero's Journey: From Newbie to Pro**

**Omnibus Edition**

Hasan Coşkun  
23169575

# Contents

<b>1</b>	<b>Beginnings in Turkey</b>	<b>2</b>
<b>2</b>	<b>The Cybersecurity Path</b>	<b>3</b>
2.1	Starting Out: The White Belt . . . . .	3
2.2	The Challenges: Learning and Growing . . . . .	4
2.3	Getting Better: Skills and Tools . . . . .	4
2.4	Looking Forward: Keeping On . . . . .	4
2.5	Ending Thoughts: A Big Journey . . . . .	5
<b>3</b>	<b>Advanced Learning and Challenges</b>	<b>6</b>
3.1	The Return of the Hero . . . . .	6
3.1.1	The Text Weavers: SED and ED . . . . .	6
3.1.2	The Rulers of Permissions: CHOWN and CHMOD . . . . .	6
<b>4</b>	<b>Mastery of Tools and Techniques</b>	<b>7</b>
4.0.1	The Twin Movers: CP and MV . . . . .	7
4.0.2	The Scripting Sorcerers: PERL-PYTHON-RUBY . . . . .	7
4.0.3	The Command Line King: BASH . . . . .	7
4.0.4	The Keepers of Time and Space: DATE and DMESG . . . . .	7
4.0.5	The Counter of Words: WC . . . . .	7
4.0.6	The Craftsmen's Tools: GCC and AS . . . . .	8
4.0.7	The Network Warrior: WGET . . . . .	8
4.0.8	The Keymaster's Challenge: SSH-KEYGEN . . . . .	8
4.1	Program Interaction . . . . .	8
4.1.1	Initial Challenges: Mastering Basic Interactions . . . . .	8
4.1.2	Refining Techniques and Efficiency . . . . .	8
4.1.3	Scripting and Binary Exploitation . . . . .	9
<b>5</b>	<b>The Struggle in Pwn Village</b>	<b>10</b>
5.1	The Struggle in Pwn Village is Gaining Momentum . . . . .	10
5.1.1	Embryoio_Level34 and the Fork Spell . . . . .	10
5.1.2	Embryoio_level41 Makes Heads Spin . . . . .	11
5.1.3	Embryoio_level50 with Interactive Python Sword . . . . .	12
<b>6</b>	<b>Hero's Journey: The Trials of Pwn Village</b>	<b>13</b>
6.1	Gaining Momentum . . . . .	13
6.1.1	Embryoio Level 54 and the Python Spell . . . . .	13
6.1.2	Embryoio Level 55 and the Grep Enchantment . . . . .	13
6.1.3	Embryoio Level 56 and the Sed Sorcery . . . . .	14
6.1.4	Embryoio Level 57 and the Reversal Ritual . . . . .	14
6.1.5	Embryoio Level 58 and the Secret Password . . . . .	15
6.1.6	Embryoio Level 60 and the Custom Binary . . . . .	15
	<b>References</b>	<b>17</b>

# Chapter 1

## Beginnings in Turkey

My journey started in Turkey. It's had its fair share of twists and turns. Right now, I'm cruising through the second chapter. Let me rewind a bit: it all kicked off when I graduated from the Turkish National Police Academy. Spent a good eight years there, learning the ropes and serving on the force for another four. But then, I hit a crossroads and decided to flip the script. Packed up my courage, waved goodbye to my badge, and dove headrst into a bachelor's degree. Fast forward to today, and here I am, knee-deep in part two of my adventure.

Over the past few years, I've been hustling hard, leveling up my skills in English, and diving deep into the world of web programming and data analysis. Every challenge I tackle is a chance to learn something new. Right now, I'm posted up in The Hague, chasing my dreams of higher education. Before coming here, I clocked in some serious hours interning at the Critical Infrastructure Protection Center of Istanbul Kadir Has University. Mostly, I was knee-deep in research and getting my hands dirty with machine learning models. That's when it hit me: cybersecurity is where I want to be.

When I stumbled upon the Software Revising and Explanations course at Hague University, I knew it was my ticket to a cybersecurity career. Sure, it's gonna be a wild ride, but I'm ready to roll up my sleeves and dive in. And that's how this chapter of my journey began. Right now, I'm just gearing up for the road ahead. Spent the past few weeks prepping for the course, starting with a crash course in Docker to get my M1 processor MacOS up to speed. Now, I'm knee-deep in the world of C programming, soaking in all the nitty-gritty details. Next up? Tackling Assembly and honing in on those Addressing and Saving areas. Hoping that no matter what bumps lie ahead, I'll find a way to rise, even if it means crawling through the mud sometimes...

## Chapter 2

# The Cybersecurity Path

### 2.1 Starting Out: The White Belt

Starting a new class is always exciting and a bit scary, especially when it's about something as complex as cybersecurity. My newest challenge is the pwn.college course. It's a tough class that teaches us how to protect and attack computer systems.

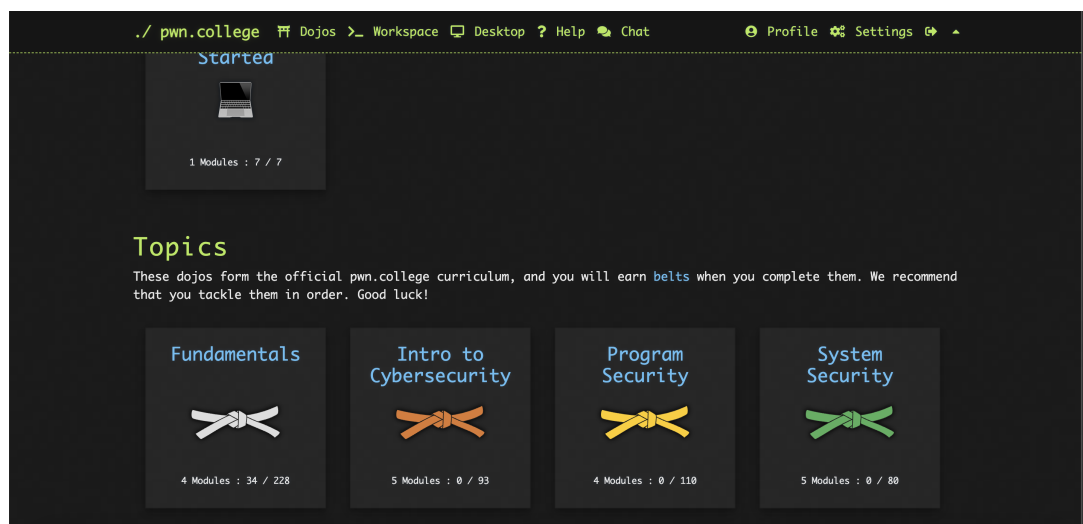


Figure 2.1: pwn.college

I want to share my story of how I began, what I'm doing now, and where I hope to be. In pwn.college, we use belts like in martial arts to show how much we've learned. Right now, I have a white belt, which means I'm just starting. My journey didn't begin here, though. I first tried learning Assembly language through a Udemy course by Paul Chin to get better at reversing software. But I realized that wasn't enough for what I really wanted to do.

So, I decided to do both the Udemy course and pwn.college at the same time. The first part of pwn.college felt like meeting the older students or "seniors" and learning the basics. This part is called "Getting Started" or the "dojos." After that, I really began my learning journey.

## 2.2 The Challenges: Learning and Growing

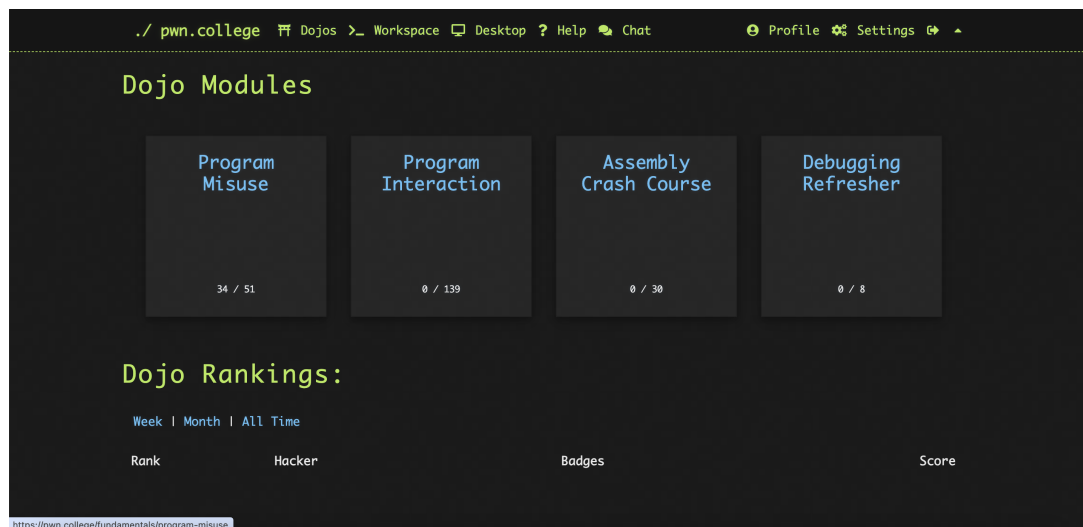


Figure 2.2: pwn.college

As a white belt, I'm at the very start, but there's a lot to learn. The course has videos and slides to teach us, and then we get to try real challenges. I've finished some and got the flags (which means I did it right), but there are still many left. Getting those flags feels great—it's like winning a little victory every time.

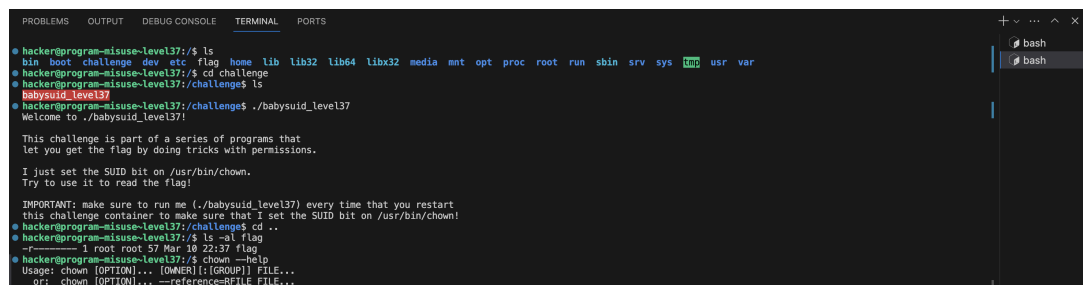


Figure 2.3: pwn.college

We learn by doing things with different computer programs, like whiptail, ed, socat, and genisoimage, using a system called Kali Linux. I start by looking for the challenge files, then I see what I need to do. A lot of it is about understanding how these programs work. Sometimes, just reading the program's help command is enough, but other times I have to search the internet for answers.

## 2.3 Getting Better: Skills and Tools

This isn't just about finishing tasks. I'm also learning a lot of important skills, like Assembly language, and using tools like xdbg64-32 and GNN. It's a lot to take in, but I'm getting better little by little. I'm also keeping notes and documents about what I learn, especially about Linux.

## 2.4 Looking Forward: Keeping On

I know I have a long way to go. There are five belts in pwn.college, and I'm only on the first one. But every challenge I solve and every flag I get makes me feel closer to my goal of being really good at cybersecurity.

This class is teaching me not just how to do specific tasks, but how to think and solve problems like a cybersecurity expert. Even though I'm just starting, I believe that if I keep working hard, I'll get through all the belts and become someone who really knows their stuff.

## **2.5 Ending Thoughts: A Big Journey**

There's an old saying, "A journey of a thousand miles begins with a single step." That's how I feel about pwn.college. Every challenge and every new thing I learn is a step toward my goal. It's not just about learning; it's about getting better and gaining real skills.

I'm excited and hopeful about the future. Yes, it's going to be tough, but I'm ready for it. Pwn.college isn't just a class for me; it's the path to becoming a pro in cybersecurity, and I can't wait to see where it leads.

## Chapter 3

# Advanced Learning and Challenges

### 3.1 The Return of the Hero

#### 3.1.1 The Text Weavers: SED and ED

```
● hacker@program-misuse~level35:/challenge$ /usr/bin/sed -n 'p' /flag
pwn.college{Y0J3X-leYvHMftN6090Wnw18ZIR.01N3EDL5QTM5QzW}
○ hacker@program-misuse~level35:/challenge$
```

Figure 3.1: Flag of level35

Our hero starts his journey with SED and ED, armed with SUID permissions that elevate his command execution to root-level capabilities. With SED, he swiftly executes `/usr/bin/sed 'p' /flag`, which pipes the entire content of the `/flag` file to his terminal screen.

In the realm of ED, he delves deeper, using `/usr/bin/ed /flag` to load the file and then issuing the `p` command to print the active line, revealing hidden data within the metadata of the document.

```
● hacker@program-misuse~level36:/challenge$ /usr/bin/ed /flag
57
p
pwn.college{YTGA1LrSLHf0uBPfW0qWGP0UH_S.0F03EDL5QTM5QzW}
```

Figure 3.2: flag of level36

#### 3.1.2 The Rulers of Permissions: CHOWN and CHMOD

```
● hacker@program-misuse~level37:/ $ /usr/bin/chown hacker /flag
● hacker@program-misuse~level37:/ $ cat flag
pwn.college{QWJpGIyPcweF6QjVaQWk0grX2wD.0V03EDL5QTM5QzW}
```

Figure 3.3: Flag of level37

Navigating the chambers of permissions, our hero exploits the SUID configuration of CHOWN and CHMOD. With precise commands like `chown hacker /flag && cat /flag`, he changes the ownership of the `flag` file to his user, ensuring access. He then alters the file permissions with `chmod 666 /flag && cat /flag`, setting it to be globally readable and writable, which allows unrestricted access to its contents.

## Chapter 4

# Mastery of Tools and Techniques

### 4.0.1 The Twin Movers: CP and MV

In the domain of file management, our hero uses CP and MV with strategic brilliance. He copies the `/flag` file to his current directory with `cp -no-preserve=all /flag . && cat flag`, effectively bypassing any attribute restrictions. Using MV, he cleverly reassigns the functionality of 'cat' to 'mv', executing a move command that also outputs the file content:

```
mv /usr/bin/cat /usr/bin/mv || ./challenge/babysuid\_level40 || mv /flag | grep pwn.college
```

### 4.0.2 The Scripting Sorcerers: PERL-PYTHON-RUBY

Equipped with scripting languages, he tackles challenges with PERL, PYTHON, and RUBY. Each script is tailored to parse and extract data stealthily. With PERL, he employs text processing to manipulate strings and file contents dynamically. In PYTHON, he scripts a small function to open and read `/flag`, handling exceptions adeptly. RUBY is used for its elegant syntax to quickly script and execute a read operation on the `/flag` file.

### 4.0.3 The Command Line King: BASH

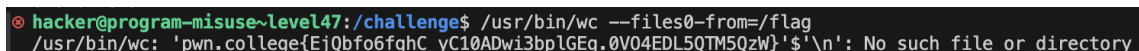
Mastering the command line with BASH, our hero utilizes its scripting capabilities to execute a high-privilege shell using `-p` to preserve the root-level permissions.

```
/usr/bin/bash -p /flag
```

### 4.0.4 The Keepers of Time and Space: DATE and DMESG

Utilizing DATE, he discovers its ability to parse each line of a file as dates with `-f /flag`, which misinterprets the flag data as erroneous dates, thus leaking it via error messages. With DMESG, he redirects the tool to output the contents of the `/flag` file by misusing the `-F` option, creatively repurposing kernel message outputs to reveal hidden information.

### 4.0.5 The Counter of Words: WC



```
hacker@program-misuse~level47:~$ ./challenge$ /usr/bin/wc --files0-from=/flag
/usr/bin/wc: 'pwn.college{EjQbfo6fghC_yC10ADwi3bp1GEg.0V04EDL5QTM5QzW}' '$'\n': No such file or directory
```

Figure 4.1: Word Counter

By invoking WC with `-files0-from=/flag`, our hero ingeniously causes WC to interpret the contents of `/flag` as filenames, leading to error outputs that include the data from `/flag`, revealing sensitive information unintentionally.



## 4.0.6 The Craftsmen's Tools: GCC and AS

In the compilers' lair, he uses GCC and AS to deconstruct and analyze the `/flag` file under the guise of compiling. GCC is run with `-x c -E /flag` to preprocess C directives in `/flag`, revealing any macros or includes as plaintext. With AS, he assembles the contents, misusing the tool to parse non-assembly code, causing leakage of file contents through error messages.

## 4.0.7 The Network Warrior: WGET

Facing WGET, our hero sets up a rogue server that captures incoming data. He misuses WGET with `-post-file="/flag"` `localhost:8888`, forcing it to send the contents of `/flag` to his server, cleverly capturing the flag through network protocols.

## 4.0.8 The Keymaster's Challenge: SSH-KEYGEN

Finally, with SSH-keygen, he ventures into key generation. Using his knowledge, he crafts a plugin in C that, when loaded into SSH-keygen using `-D lib.so`, performs an unauthorized operation to read and print the contents of `/flag`, showcasing a novel exploit of SSH-keygen's functionality.

# 4.1 Program Interaction

Our hero embarks on a new chapter, delving into the intricacies of program interaction. The journey requires a thorough understanding of processes, parent and child relationships, and how to manipulate programs to meet his needs.

## 4.1.1 Initial Challenges: Mastering Basic Interactions

In the early stages of his adventure, our hero encounters a series of challenges that test his proficiency in managing bash shells, manipulating input and output, and handling environment variables. He adeptly navigates these by starting new bash sessions, redirecting outputs, and dynamically adjusting the environment settings to meet the requirements of each task. This demonstrates his foundational skills in process handling, which are critical for the more complex challenges that lay ahead.

```
hacker@program-interaction~level2:/challenge$ /bin/bash
hacker@program-interaction~level2:/challenge$ echo "iykduqib" | ./embryoio_level2
```

Figure 4.2: Program interaction

Using a variety of scripting techniques, he progresses through the initial fifteen levels. He showcases his ability to script interactions using simple bash commands, redirect file contents cleverly, and manipulate environmental variables to exploit the programs' vulnerabilities. These tasks highlight his strategic use of command-line tools to bridge the gap between user actions and program responses, ensuring success at each stage.

## 4.1.2 Refining Techniques and Efficiency

Our hero then refines his techniques to enhance efficiency and reduce redundancy. Notably, during 'embryoio-level13', he initially directs the output to a temporary file, but later optimizes the process with a single command:

```
/challenge/embryoio_level13 > /tmp/umcqp n && cat /tmp/umcqp n
```

### 4.1.3 Scripting and Binary Exploitation

```
GNU nano 4.8                                pyt.py
#!/usr/bin/python3.8
import subprocess
with open("/tmp/qymjzu", "r") as file:
    subprocess.run(["/challenge/embryoio_level19", cwd=".", stdin=file, text=True)
```

Figure 4.3: Script for embryoio<sub>level19</sub>

As the challenges increase in complexity, our hero turns to IPython for scripting tasks requiring sophisticated control over subprocesses and file manipulations. He also crafts a C program to manipulate low-level system operations. This program uses the `fork()` function to manage control flow between parent and child processes:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5 void pwncollege() {}
6 int main() {
7     pid_t pid = fork();
8     if (pid == 0) {
9         execl("/challenge/embryoio_level29", "embryoio_level29", NULL);
10    } else if (pid > 0) {
11        waitpid(pid, NULL, 0);
12    } else {
13        perror("fork");
14        return 1;
15    }
16    return 0;
17 }
```

After compiling this into a binary:

```
gcc -o pwn_college pwn_college.c
mv /tmp/pwn_college /home/hacker/
cd /home/hacker
./pwn_college
```

By compiling his C program into a binary and executing it, he deeply engages with the system's architecture, exploiting nuances of process creation and execution to capture flags. This marks a significant leap in his journey, showcasing his advanced understanding of system internals and his ability to integrate this knowledge to overcome challenges.

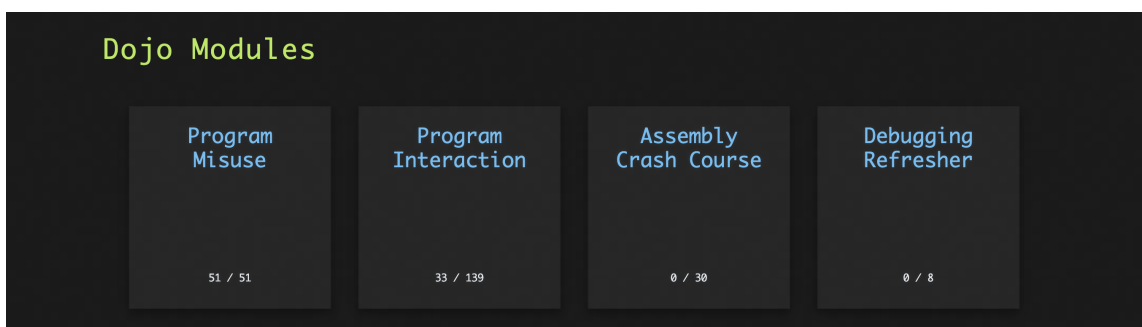


Figure 4.4: Dojo Dashboard

## Chapter 5

# The Struggle in Pwn Village

### 5.1 The Struggle in Pwn Village is Gaining Momentum



Figure 5.1: Dojo Dashboard

Our hero has rested in Pwn village, and then decided to move from room 31 to room 52. During his journey, he realized the need to understand the relationships between parent and child processes. After a thorough research ((TutorialsPoint, )), he understood that processes can be multiple and can communicate with each other. He discovered that creating a new subprocess with each process was possible using the `fork()` system call, and he started using it right away.

With the `fork()` system, he also learned to manage these processes' tasks using the `exec()` family of functions. After experiencing how to track a child process operation using `wait()` or `waitpid()` functions, our hero is now equipped and ready for the adventure.

#### 5.1.1 Embryoio\_Level34 and the Fork Spell

Here, our hero creates a child process using the `fork()` spell and names it "pid". If the pid value equals zero, he redirects its standard output to the `/tmp/csumctf` file. Then, the parent process waits for the completion of the child process and finally prints stdout.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h> // Wait function
#include <fcntl.h> //File Control function

void pwncollege() {
    // It is just for the requirement...
}

int main() {
    pwncollege();
    int pid = fork(); // Creating a new process with fork()

    if (pid == 0) { // Child process --if pid equals 0, it is child process. If pid 1, it is parent and if it is lower than 0, there will be an error
        int fileDescriptor = open("/tmp/csuwtc", O_WRONLY | O_CREAT | O_TRUNC, 0644); //Open or create file for writing only and permissions...
        if (fileDescriptor < 0) {
            perror("Failed to open file");
            return 1;
        }

        dup2(fileDescriptor, STDOUT_FILENO); // Redirecting standard output of the child process to file
        close(fileDescriptor);

        char *args[] = {"/challenge/embryoio_level34", NULL}; // Executable to run
        execve(args[0], args, NULL); // Execute a file
    } else if (pid > 0) { // Parent process
        int status;
        waitpid(pid, &status, 0); // Wait for the child process to finish
        if (WIFEXITED(status)) {
            printf("Child process exited with code %d\n", WEXITSTATUS(status)); // Print child's exit code
        }
    }

    return 0;
}
```

Figure 5.2: script for embryoio<sub>level34</sub>

*Note: Supported by ChatGPT*

```
● hacker@program-interaction~level34:~$ cat /tmp/csuwtc
WELCOME! This challenge makes the following asks of you:
- the challenge checks for a specific parent process : binary
- the challenge will check that output is redirected to a specific file path : /tmp/csuwtc

ONWARDS TO GREATNESS!
```

Figure 5.3: embryoio<sub>level34</sub>

After discovering ways to manage data flow between processes, our hero gains strength in the basic functions of stdin and stdout. Especially with stdin, he enables a process to receive data from the outside world, and with stdout, he experiences outputting the data processed by the processes. Through these two streams, processes can interact with each other and with users. Also, he discovers that he can connect these two streams with a pipe, thus directing the output of one process directly as input to another.

### 5.1.2 Embryoio\_level41 Makes Heads Spin

When our hero reviews the necessities for the challenge, he realizes the needs to create a parent process. In addition, he rolls up his sleeves to store a password in another box and read it with the "cat" spell. However, he discovers that the password needs to be reversed.

After accomplishing this with the "rev" spell, he senses another oddity in the challenge. Since the password was reversed, the parent process did not accept it. But how could this be because the requirement was using the "rev" spell?

```
hacker@program-interaction~level41:/challenge$ /bin/bash
hacker@program-interaction~level41:/challenge$ mkfifo /tmp/pipe
hacker@program-interaction~level41:/challenge$ cat /tmp/pipe | rev | rev | ./embryoio_level41 &
[1] 671
```

Figure 5.4: 'rev' usages

Just then, the answer came to our hero's mind, and he smiled. This spell was reversing the password, but the parent process wanted the state of the password. He performed this once more with the "rev" spell. Moreover, he placed a pipe (|) between these spells to make them affect each other.

Now it was time to transfer the password to the box named pipe because he had added one more magic to his last spell. This magic, the '&' spell, continued to check the contents of the box constantly. Eventually, he found the desired password and presented our hero with his reward.

```
echo "fixjrlva" > /tmp/pipe
hacker@program-interaction~level41:/challenge$ [GOOD] You successfully passed the password!
[GOOD] Success! You have satisfied all execution requirements. Here is your flag:
pwn.college{UVyvk0DMd6ypGwKNtYBmBeFgDVG.QXxQDL5QTM5QzW}
```

Figure 5.5: Flag of level41

Among the challenges our hero faced, there were not only topics like child and parent processes but also he needed to use the script sword. In these challenges, our hero had to properly set up and use processes with Bash, Shell, and iPython swords. Especially while using shell scripts in non-interactive mode and often preferring the interactive version, iPython, for Python.

### 5.1.3 Embryoio\_level50 with Interactive Python Sword

Here, our hero faces the task of manipulating Linux processes ((Pwntools, )) and Pipes using the Python sword. His task consists of arranging a parent process's output in a certain order to process the result.

```
● hacker@program-interaction~level50:/challenge$ cd /tmp && nano pyt.py
● hacker@program-interaction~level50:/tmp$ chmod +x pyt.py
○ hacker@program-interaction~level50:/tmp$ ipython
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.12.3 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %run /tmp/pyt.py
```

Figure 5.6: run ipython

#### Python Script:

```
from subprocess import Popen, PIPE
challenge_command = ["/challenge/embryoio_level50"] #parent process
sed_command = ["sed", "s/pattern/replacement/"] #child process
p_challenge=Popen(challenge_command, stdout=PIPE) #the output of it will be using for p_sed
p_sed = Popen(sed_command, stdin=p_challenge.stdout, stdout=PIPE, stderr=PIPE)
p_challenge.stdout.close()
output, errors = p_sed.communicate()
print("Output: ", output.decode())
if errors:
    print("Errors:", errors.decode())
```

Figure 5.7: Script for embryoio\_level50

And here's our hero claiming his reward:

```
pwn.college{YDW7Ty1uSGKe_14P5KNgmMXVcDK.QXwUDL5QTM5QzW}
```

## Chapter 6

# Hero's Journey: The Trials of Pwn Village

### 6.1 Gaining Momentum

The Struggle in Pwn Village is Gaining Momentum Our hero, a tenacious and skilled warrior, embarks on a journey through the enchanted realms of Pwn Village. With every step, he faces trials that test his knowledge and skills in the mystical arts of process manipulation and inter-process communication. Each level he conquers brings him closer to ultimate mastery and the treasures that await.

#### 6.1.1 Embryoio Level 54 and the Python Spell

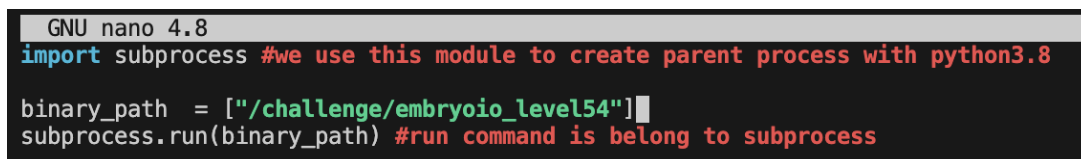
In the realm of Level 54, the guardian demands a demonstration of our hero's ability to manipulate processes with precision:

Requirements

- The parent process must be 'python'.
- The process at the other end of 'stdout' must be 'cat'.

Solution:

With a swift motion, our hero wields the Python spell to summon the necessary processes. He crafts a script, 'pyt.py', using the 'subprocess' module to call upon the '/challenge/embryoio\_level54' binary. With a deft incantation, he executes the script through Python 3.8 and channels its output to 'cat':



```
GNU nano 4.8
import subprocess #we use this module to create parent process with python3.8

binary_path = ["/challenge/embryoio_level54"]
subprocess.run(binary_path) #run command is belong to subprocess
```

Figure 6.1: script for embryoio\_level54



```
● hacker@program-interaction~level54:/tmp$ python3.8 pyt.py | cat
```

Figure 6.2: parent process

#### 6.1.2 Embryoio Level 55 and the Grep Enchantment

As the steps into Level 55, our hero encounters a challenge that requires harmonious interaction between processes:

- Requirements-

- The parent process must be 'python'.

- The process at the other end of 'stdout' must be 'grep'.

Solution: Once again, our hero conjures a Python script ('pyt.py') to run the '/challenge/embryoio\_level55' binary. He deftly directs the output through 'grep' using the following command:

```
GNU nano 4.8
import subprocess
binary_path = "/challenge/embryoio_level55"
subprocess.run(binary_path)
```

Figure 6.3: script for embryoio<sub>level55</sub>

```
hacker@program-interaction~level55:/challenge$ cd /tmp && nano pyt.py
hacker@program-interaction~level55:/tmp$ nano pyt.py
hacker@program-interaction~level55:/tmp$ chmod +x pyt.py
hacker@program-interaction~level55:/tmp$ python3.8 pyt.py | grep ""
WELCOME! This challenge makes the following asks of you:
- the challenge checks for a specific parent process : python
- the challenge checks for a specific process at the other end of stdout : grep
```

### 6.1.3 Embryoio Level 56 and the Sed Sorcery

Continuing to Level 56, the guardian tests our hero's ability to manipulate streams with dexterity:

- Requirements:

- The parent process must be 'python'.
- The process at the other end of 'stdout' must be 'sed'.

Solution:

Displaying his growing prowess, our hero scripts another Python incantation ('pyt.py') to invoke the '/challenge/embryoio\_level56' binary, ensuring the output flows through 'sed':

```
BrokenPipeError: [Errno 32] Broken pipe
hacker@program-interaction~level56:/tmp$ nano pyt.py
hacker@program-interaction~level56:/tmp$ python3.8 pyt.py | sed ""
WELCOME! This challenge makes the following asks of you:
- the challenge checks for a specific parent process : python
- the challenge checks for a specific process at the other end of stdout : sed
```

Figure 6.4: embryoio<sub>level56</sub>

### 6.1.4 Embryoio Level 57 and the Reversal Ritual

Level 57 presents a convoluted challenge, where the guardian demands precise output manipulation:

- Requirements:

- The parent process must be 'python'.
- The process at the other end of 'stdout' must be 'rev'.

Solution:

Utilizing his deepening understanding, our hero writes a Python script ('pyt.py') and channels the output through the 'rev' spell twice:

```

}WzQ5MTQ5LDU3XQ.nd7sNSAE_RN-uKykyBTb2KE02R8{egelloc.nwp
● hacker@program-interaction~level57:/tmp$ python3.8 pyt.py | rev | rev
WELCOME! This challenge makes the following asks of you:
- the challenge checks for a specific parent process : python
- the challenge checks for a specific process at the other end of stdout : rev

```

### 6.1.5 Embryoio Level 58 and the Secret Password

In the penultimate trial, Level 58, the guardian demands a complex interaction involving a hidden password:

- Requirements:

- The parent process must be 'python'.
- The process at the other end of 'stdin' must be 'cat'.
- A hardcoded password ('kaeuhyse') must be provided over 'stdin'.

Solution:

Displaying his mastery, our hero scripts a Python solution ('pyt.py') that orchestrates the processes, using 'cat' to pass the password to the binary:

```

import subprocess

# Create a subprocess for the persistent process (cat)
# This process will read from stdin and write to stdout.
cat_process = subprocess.Popen(["cat"], stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

# Start the embryoio_level58 process and pass the cat process as stdin
# This means the embryoio_level58 program will get its input from the cat process's stdout.
process = subprocess.Popen(["/challenge/embryoio_level58"], stdin=cat_process.stdout, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

# Send the hardcoded password to the cat process.
# This password is what the embryoio_level58 program expects to read from its stdin.
cat_process.stdin.write(b"kaeuhyse\n")
# Flush the stdin buffer to ensure the data is sent immediately.
cat_process.stdin.flush()

# Wait for the embryoio_level58 process to complete and capture its output.
output, error = process.communicate()

# Print the output of the embryoio_level58 process.
print(output.decode())

# Close the stdin of the cat process since we are done sending input.
cat_process.stdin.close()
# Terminate the cat process.
cat_process.terminate()
# Wait for the cat process to fully terminate.
cat_process.wait()

```

Figure 6.5: script and comments for embryoio\_level58

```

● hacker@program-interaction~level58:/challenge$ cd /tmp && nano pyt.py
● hacker@program-interaction~level58:/tmp$ chmod +x pyt.py
● hacker@program-interaction~level58:/tmp$ python3 pyt.py

```

Figure 6.6: running of the pyt.py

### 6.1.6 Embryoio Level 60 and the Custom Binary

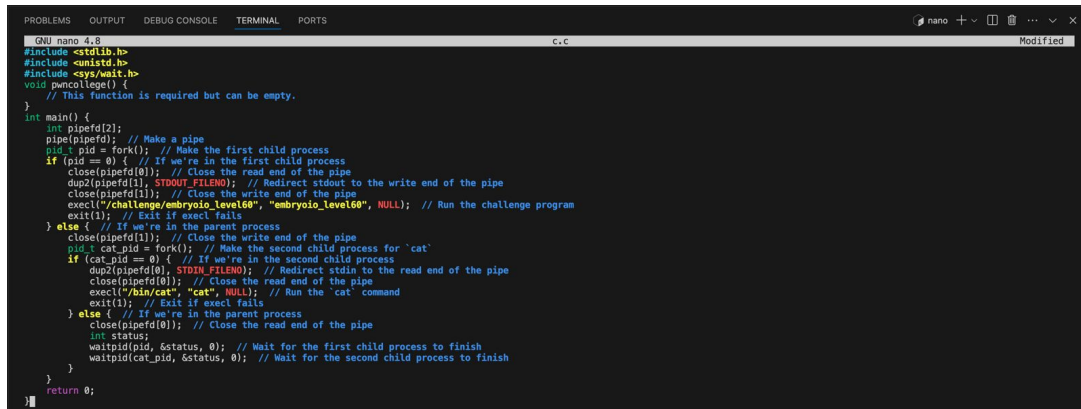
In the final trial, Level 60, the guardian sets a challenge requiring the creation of a custom binary:

- Requirements:

- The parent process must be a custom binary that you created by compiling a C program.
- The process at the other end of 'stdout' must be 'cat'.
- The custom binary must be in your home directory.
- The custom binary must include a function called 'pwncollege'.



Solution: 1. C Program: Creating a file named 'c.c' in my home directory with the following code:

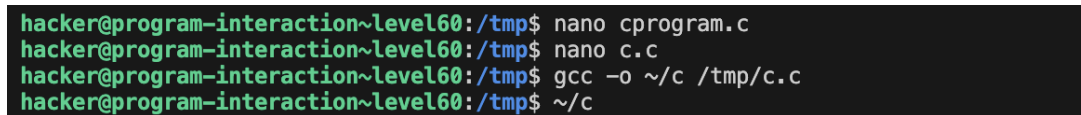


```
GNU nano 4.8 c.c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
void perrorlege() {
    // This function is required but can be empty.
}

int main() {
    int pipefd[2]; // Make a pipe
    pipe(pipefd); // Make a pipe
    pid_t pid = fork(); // Make the first child process
    if (pid == 0) { // If we're in the first child process
        close(pipefd[0]); // Close the read end of the pipe
        dup2(pipefd[1], STDOUT_FILENO); // Redirect stdout to the write end of the pipe
        close(pipefd[1]); // Close the write end of the pipe
        execl("/challenge/embryoio_level60", "embryoio_level60", NULL); // Run the challenge program
        exit(1); // Exit if execl fails
    } else { // If we're in the parent process
        close(pipefd[1]); // Close the write end of the pipe
        pid_t cat_pid = fork(); // Make the second child process for 'cat'
        if (cat_pid == 0) { // If we're in the second child process
            dup2(pipefd[0], STDIN_FILENO); // Redirect stdin to the read end of the pipe
            close(pipefd[0]); // Close the read end of the pipe
            execl("/bin/cat", "cat", NULL); // Run the 'cat' command
            exit(1); // Exit if execl fails
        } else { // If we're in the parent process
            close(pipefd[0]); // Close the read end of the pipe
            int status;
            waitpid(pid, &status, 0); // Wait for the first child process to finish
            waitpid(cat_pid, &status, 0); // Wait for the second child process to finish
        }
    }
    return 0;
}
```

Figure 6.7: Script with C

2. Compile and execute the program: Saving the file and compiling it to create the custom binary.



```
hacker@program-interaction~level60:/tmp$ nano cprogram.c
hacker@program-interaction~level60:/tmp$ nano c.c
hacker@program-interaction~level60:/tmp$ gcc -o ~/c /tmp/c.c
hacker@program-interaction~level60:/tmp$ ~/c
```

Figure 6.8: Compiling and executing c program

By following these steps, our hero successfully meets all the requirements set forth by the guardian of Level 60, demonstrating his mastery in process manipulation and inter-process communication.

Throughout this journey, the hero Hasan Coskun has faced numerous challenges that tested his abilities in process management and inter-process communication. Each level demanded unique strategies and precise execution, pushing him to improve his skills and knowledge.

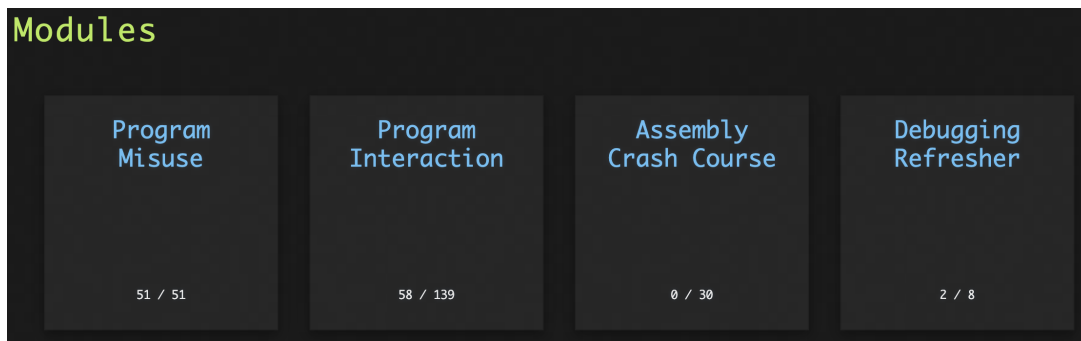


Figure 6.9: Resting place for the Hero

# References

- Coalfire. (2020). The basics of exploit development 2: Seh overflows. Retrieved from <https://coalfire.com/the-coalfire-blog/the-basics-of-exploit-development-2-seh-overflows> (Accessed: 2024-05-12)
- Database, E. (2020). Shellcodes for windows/x86-64 - 64-bit null-free shell bind tcp shellcode. Retrieved from <https://www.exploit-db.com/shellcodes/48116> (Accessed: 2024-05-12)
- Pwntools. (2021). Processes — pwntools 4.7.0 documentation. Retrieved from <https://docs.pwntools.com/en/stable/tubes/processes.html> (Accessed: 2024-05-12)
- TutorialsPoint. (2021). Process vs. parent process vs. child process. Retrieved from <https://www.tutorialspoint.com/process-vs-parent-process-vs-child-process> (Accessed: 2024-05-12)