Cyber Escape Room: Shared Task Description

Introduction and Description

An **escape room** (also called escape game, puzzle room, exit game, or riddle room) is a collaborative experience where a team of players must discover clues, solve puzzles, and complete tasks across one or more rooms to achieve a defined objective within a limited time frame, often “escaping” from the room [1] [2]. This format supports teamwork, lateral thinking, and layered puzzle design.

You assume the role of a blue-team defender responding to a simulated incident in a facility, and the task to collect and create a report of clues from the incident(s). The **Cyber Escape Room** you will build and run is a **single Python CLI application** that runs a sequence of commands to solve puzzles and collect clues from the rooms. Each room corresponds to a module or concept from the course and simulates a realistic slice of defensive work.

Why this Escape Room?

In real security operations (blue team), defenders face many different tasks:

- **Triage logs** to spot brute-force or scanning activity
- **Check configurations** for weak or suspicious entries
- **Search large dumps** for hidden indicators
- **Trace processes** to find how malware moves through a system and reconstruct process trees to detect lateral movement
- **Verify findings and report** in a consistent way and sign the report

This escape room compresses that spectrum into five simplified "rooms"
Each room stands for one type of defender task, that cover the **spectrum of blue-team work**:

Room	Represents	Real-world parallel
SOC Triage Desk	Log parsing & anomaly detection	Analysts review auth logs for brute force attacks
DNS Closet	Config analysis & decoding	Incident responders decode obfuscated hints in configs
Vault Corridor	Regex search & validation	Forensics sift through dumps for valid artifacts

Room	Represents	Real-world parallel
Malware Lab	Graph traversal	Threat hunters trace malware process trees
Final Gate	Verification & reporting	Teams must package findings into proofs

Below is a high-level flow of the game. Use it as a mental map of the tasks that follow.

Course Session	Python Concepts	Escape Room Step	Token
1	Syntax, control flow, CLI, Git, venv	Game engine skeleton (Lobby)	None
2	Data structures, error handling	Room 1: SOC Triage Desk	KEYPAD
3	Classes, functions, modularization	Room 2: DNS Closet	DNS
4	Algorithms, regex, complexity	Room 3: Vault Corridor	SAFE
5	Recursion, DFS/BFS, graph traversal	Room 4: Malware Lab	PID
6	Linting, coding good practices	Room 5: Final Gate + polish	ESCAPE
7	Demos	Full run + presentation	—

Learning outcomes: by the end of the project you will be able to:

- Decompose a complex problem into modular components and implement them in Python.
- Use core Python constructs (data structures, control flow, functions, classes, I/O).
- Implement and explain algorithms (regex search, DFS/BFS graph traversal, hashing/HMAC).
- Write robust code that tolerates malformed inputs and logs evidence.
- Present and defend design choices in a live demo.

2. Project overview

What you build: a CLI game `escape.py` that interacts with small provided log files. Your program must run deterministically and be implemented in pure Python, using functions, classes, and the standard built-in string, files and data processing libraries.

Deliverables:

- A working repository with code and modules.
- A transcript file (`run.txt`) containing all required token lines and evidence: TOKEN[...], EVIDENCE[...], and FINAL_GATE
- Tests, types, and linting by Session 6 (minimum checks).

- A live demo on Session 7 (last session)

Important constraints:

- No external network calls. Use only Python standard libraries (built-in string, file, and data processing).
- No hardcoded tokens.
- Fail gracefully on malformed input, log warnings rather than crashing.

The five core rooms are:

1. Intro Lobby (engine)
2. SOC Triage Desk (`auth.log`)
3. DNS Closet (`dns.cfg`)
4. Vault Corridor (`vault_dump.txt`)
5. Malware Lab (`proc_tree.jsonl`)

After completing Rooms 2–5, you will have collected four tokens.
The Final Gate requires you to combine these tokens in the order specified in `final_gate.txt` .

Your program must output the combined message, the `expected_hmac` value from the file, and mark the gate as: `FINAL_GATE=PENDING` :

```
FINAL_GATE=PENDING
MSG=<group_id|token1-token2-token3-token4>
EXPECTED_HMAC=<hex from final_gate.txt>
```

You solution will be evaluated during the demo

3. The dataset (you will get)

Each group receives a small **data pack** containing the files below. Files are intentionally small and contain noise to encourage robust parsing.

- `auth.log` — about 120 lines of SSH authentication events, contains both `Failed password` and `Accepted password` lines and 3–5 malformed lines.
- `dns.cfg` — a key=value style config with several values are base64-encoded hints (decoys included), and a `token_tag` indicates which `hintX` is relevant.
- `vault_dump.txt` — a text dump with a single valid `SAFE{a-b-c}` code hidden among distractors; the true code satisfies `a + b = c` (e.g. 1(a)+2(b)=3(c)).
- `proc_tree.jsonl` — one JSON-per-line representing process records `{pid, ppid, cmd}` ; includes a malicious chain ending in `curl` or `scp` . Variant B may contain a cycle.
- `final_gate.txt` — defines `token_order` , the `group_id` , and the `expected_hmac` (used for verification).

Note: Each group receives a variant (A/B/C) with different seeds so tokens differ per group.

4. Room-by-room specification

Below is a description of each room, the exact student task, and the required transcript evidence. Read carefully, your transcript is the primary artifact you will be graded on.

Room 1 — Intro Lobby (engine boot)

Purpose: Ensure the CLI and engine are present

Must-have behavior:

- Running `python escape.py --start intro --transcript run.txt` starts an interactive REPL (Read–Eval–Print Loop)
- Commands supported: `look`, `move <room>`, `inspect <item>`, `use <item>`, `inventory`, `hint`, `save`, `load`, `quit`.

Example REPL:

```
$ python escape.py --start intro --transcript run.txt
[Game] Cyber Escape Room started. Type 'help' for commands.

> look
You are in the Intro Lobby.
A terminal blinks in the corner. Doors lead to: soc, dns, vault,
malware, final.

> move soc
You enter the SOC Triage Desk.
A cluttered screen shows failed SSH login attempts.
Items here: auth.log

> inspect auth.log
[Room SOC] Parsing logs...
17 failed attempts found in 203.0.113.0/24
Top IP is 203.0.113.42 (last octet=42)
Token formed: 4217

TOKEN[KEYPAD]=4217
EVIDENCE[KEYPAD].TOP24=203.0.113.0/24
EVIDENCE[KEYPAD].COUNT=17
EVIDENCE[KEYPAD].SAMPLE=2025-08-09T12:02:11Z lab1 sshd[2331]:
Failed password for root from 203.0.113.42 port 50432 protocol 2
EVIDENCE[KEYPAD].MALFORMED_SKIPPED=3

> inventory
You currently hold: KEYPAD
```

```
> move dns
You enter the DNS Closet.
The walls are covered with scribbled key=value pairs.
Items here: dns.cfg

> inspect dns.cfg
[Room DNS] Decoding hints...
Decoded line: "The code is not in the roots but near the closet."
Token formed: closet

TOKEN[DNS]=closet
EVIDENCE[DNS].KEY=hint2
EVIDENCE[DNS].DECODED_LINE=The code is not in the roots but near
the closet.

> move final
You stand before the Final Gate. The console asks for proof.

> use gate
Collected tokens: DNS=closet, SAFE=?, PID=?, KEYPAD=4217
Not all tokens found. The gate remains locked.

> save save1.json
[Game] Progress saved.

> quit
[Game] Goodbye. Transcript written to run.txt
```

No token produced here.

Room 2 — SOC Triage Desk (file: `auth.log`)

The SSH logs show repeated authentication failures. Your task is to identify the most likely attacking subnet.

Student task:

1. Parse `auth.log` line-by-line.
2. Tolerate malformed lines, skip them but count how many were skipped.
3. For each `Failed password` line, extract the source IP and group by `/24` (first three octets).
4. Identify the `/24` with the largest number of failures.
5. Within that `/24`, choose the IP that occurred most frequently, take its **last octet** `L`.
6. Form the keypad code token: `"{L}{COUNT}"` where `COUNT` is the failure count in that `/24`.

Transcript contract (required lines):

```
TOKEN[KEYPAD]=<code>
EVIDENCE[KEYPAD].TOP24=<cidr>/24
EVIDENCE[KEYPAD].COUNT=<int>
EVIDENCE[KEYPAD].SAMPLE=<full original log line>
EVIDENCE[KEYPAD].MALFORMED_SKIPPED=<int>
```

Skills practiced: file I/O, dictionaries, string splitting, exception handling.

Room 3 — DNS Closet (file: `dns.cfg`)

DNS hints are encoded, most are decoys (false hints)

Student task:

1. Parse a `key=value` file robustly (allow extra spaces and `#` comments)
2. For keys `hint1..hintN`, attempt to decode values as base64 (catch exceptions where decoding fails)
3. `token_tag=X` indicates the **single** `hintX` whose decoded last word is the token

Transcript contract:

```
TOKEN[DNS]=<word>
EVIDENCE[DNS].KEY=hintX
EVIDENCE[DNS].DECODED_LINE=<decoded sentence>
```

Skills practiced: base64, parsing, functions, testing

Room 4 — Vault Corridor (file: `vault_dump.txt`)

A noisy dump contains a safe code `SAFE{a-b-c}`, only one candidate satisfies the checksum **`a+b==c`**

Student task:

1. Precompile a regex that captures `SAFE{a-b-c}` tolerant to whitespace and newlines.
2. Find candidates and validate `a+b==c` (integers)

Transcript contract:

```
TOKEN[SAFE]=a-b-c
EVIDENCE[SAFE].MATCH="SAFE{a-b-c}"
EVIDENCE[SAFE].CHECK=a+b=c
```

Skills practiced: regex, validation, text processing

Room 5 — Malware Lab (file: `proc_tree.jsonl`)

A process tree contains a malicious chain ending with an exfil command (`curl` or `scp`).

Student task:

1. Read JSON-lines, build adjacency children map `children[ppid] -> list[pid]` .
2. Implement DFS (recursive) and BFS (iterative) routines, use a `visited` set to avoid cycles
3. Starting from a given PID (stated in the room text), find any path that ends in a `cmd` containing `curl` or `scp` .
4. Token is the terminal PID of that path.

Transcript contract:

```
TOKEN[PID]=<pid>
EVIDENCE[PID].PATH=[p0->p1->...->pid]
EVIDENCE[PID].CMD="<matched command>"
```

Skills practiced: graphs, recursion, complexity analysis, robust JSON parsing.

Final Gate (file: `final_gate.txt`)

The exit checks cryptographic integrity of tokens.

Student task:

- Read `final_gate.txt` for the required `token_order` , `group_id` , and `expected_hmac` .
- Combine your tokens in the correct order to form the message: `group_id|token1-token2-token3-token4`
- Output the message and the expected HMAC from the file.
- Mark the gate as pending. The instructor will verify during the demo.

Transcript contract:

```
FINAL_GATE=PENDING
MSG=<group_id|token1-token2-token3-token4>
EXPECTED_HMAC=<hex from final_gate.txt>
```

5. Deliverables & Grading (what you hand in)

You must provide a repository with the required modules and a `run.txt` transcript produced by running:

```
python escape.py --start intro --transcript run.txt
```

You must structure your project as the following:

```
escape-room/
├─ README.md
├─ escape.py                      # CLI entry point (REPL)
├─ escaperoom/
│  ├─ __init__.py
│  ├─ engine.py                  # REPL loop + GameState + command
routing
│  ├─ transcript.py              # Transcript logger (writes into
run.txt)
│  ├─ utils.py                   # small helpers: IP checks, cfg
parsing, etc.
│  └─ rooms/
│     ├─ __init__.py
│     ├─ base.py                 # Room ABC (shared interface)
│     ├─ soc.py                  # Room 2: SOC Triage Desk
│     ├─ dns.py                  # Room 3: DNS Closet
│     ├─ vault.py                # Room 4: Vault Corridor
│     └─ malware.py              # Room 5: Malware Lab
├─ data/                         # data files provided per group
│  ├─ auth.log
│  ├─ dns.cfg
│  ├─ vault_dump.txt
│  ├─ proc_tree.jsonl
│  └─ final_gate.txt
└─ .gitignore
```

Grading scheme

- Correctness: 40% — all rooms produce tokens and the Final Gate opens
- Design & decomposition: 25% — clear classes/functions and module boundaries
- Demo & defence: 25% — live run and answers to questions
- Code and documentation quality: 10% — names, docstrings, README, error handling, PEP8 cleanliness

In [ ]:

```python
# Starter skeleton (this is only illustrative, write your own)
from dataclasses import dataclass, field
from typing import Dict, Set, List

@dataclass
class GameState:
    current_room: str = 'intro'
    inventory: Set[str] = field(default_factory=set)
    tokens: Dict[str, str] = field(default_factory=dict)
    flags: Dict[str, str] = field(default_factory=dict)

class Room:
    def __init__(self, name: str, description: str):
        self.name = name
        self.description = description

    def solve(self, state: GameState):
        raise NotImplementedError("Implement puzzle logic in subclasses")
```

7. Tips, common pitfalls, and good practices

- **Do not hardcode answers.** The dataset is seeded per group.
- **Log warnings** for malformed lines, do not raise uncaught exceptions.
- **Break tasks into small functions** and test them independently.
- **Use type hints** and small docstrings on public functions.

6. Transcript format (strict schema)

For automated evaluation and grading, the transcript **must** include the following tagged lines (exact prefixes):

- `TOKEN[KEYPAD]=...`
- `EVIDENCE[KEYPAD].TOP24=...`
- `EVIDENCE[KEYPAD].COUNT=...`
- `EVIDENCE[KEYPAD].SAMPLE=...` (at least one)
- `EVIDENCE[KEYPAD].MALFORMED_SKIPPED=...`
- `TOKEN[DNS]=...`
- `EVIDENCE[DNS].KEY=hintX`
- `EVIDENCE[DNS].DECODED_LINE=...`
- `TOKEN[SAFE]=a-b-c`
- `EVIDENCE[SAFE].MATCH="SAFE{a-b-c}"`
- `EVIDENCE[SAFE].CHECK=a+b=c`
- `TOKEN[PID]=...`
- `EVIDENCE[PID].PATH=[p0->p1->...->pid]`
- `EVIDENCE[PID].CMD="..."`
- Final gate lines:
    - `FINAL_GATE=PENDING`, `MSG=...`, `EXPECTED_HMAC=...`

Make sure the transcript is human-readable and contains exactly these tags so it can be parsed it reliably.

7. Tips, common pitfalls, and good practices

- **Do not hardcode answers.** The dataset is seeded per group.
- **Log warnings** for malformed lines, do not raise uncaught exceptions.
- **Break tasks into small functions** and test them independently.
- **Use type hints** and small docstrings on public functions.

References

1. ^ Nicholson, S. (2015). *Peeking Behind the Locked Door: A Survey of Escape Room Facilities*. White Paper. Retrieved from http://scottnicholson.com/pubs/erfacwhite.pdf

2. ^ Hall, L.E. (2021). *Planning Your Escape: Strategy, Secrets, and Habits of Successful Escape Room Players*. Simon & Schuster. ISBN 9781982140342.

ENJOY YOUR GAME AND HAPPY ESCAPING

SEE YOU AT THE FINAL GATE!

```
In [ ]:
```