VulnERR Security Report

File Name: testcase_1.py

Analysis Result:

Status: vulnerable

Severity: High

Message: Potential taint vulnerability detected

AI Suggestions:

Taint Vulnerabilities Analysis

1. **Command Injection**

The **Command Injection** vulnerability occurs in the `command_injection()` function. This function takes user input and uses it directly in the `os.system()` function without proper sanitization or validation. This allows an attacker to inject malicious commands, potentially leading to unauthorized access, data theft, or system compromise.

Why it is dangerous:

Unvalidated user input can lead to arbitrary command execution.
Malicious commands can be injected, potentially causing harm to the system or data.

How to fix:

```
def command_injection():
filename = input("Enter filename to display: ")
try:
with open(filename, "r") as f:
print(f.read())
except FileNotFoundError:
print("File not found.")
```

2. **SQL Injection**

The **SQL Injection** vulnerability occurs in the `sql_injection()` function. This function takes user input and uses it directly in the SQL query without proper sanitization or validation. This allows an attacker to inject malicious SQL code, potentially leading to unauthorized data access, modification, or deletion.

Why it is dangerous:

Unvalidated user input can lead to arbitrary SQL code execution.

Malicious SQL code can be injected, potentially causing harm to the database or data.

How to fix:

```
def sql_injection():
conn = sqlite3.connect("test.db")
cursor = conn.cursor()

username = input("Enter username: ")
password = input("Enter password: ")

query = "SELECT * FROM users WHERE username=? AND password=?"
cursor.execute(query, (username, password))

print(cursor.fetchall())
```

3. **File Read**

The **File Read** vulnerability occurs in the `file_read()` function. This function takes user input and uses it directly in the `open()` function without proper sanitization or validation. This allows an attacker to read arbitrary files, potentially leading to unauthorized access to sensitive data.

Why it is dangerous:

Unvalidated user input can lead to arbitrary file access.
Malicious files can be read, potentially causing harm to the system or data.

How to fix:

```
def file_read():
path = input("Enter file path: ")
try:
with open(path, "r") as f:
print(f.read())
except FileNotFoundError:
print("File not found.")
```

4. **Dynamic Code Execution**

The **Dynamic Code Execution** vulnerability occurs in the `dynamic_execution()` function. This function takes user input and uses it directly in the `exec()` function without proper