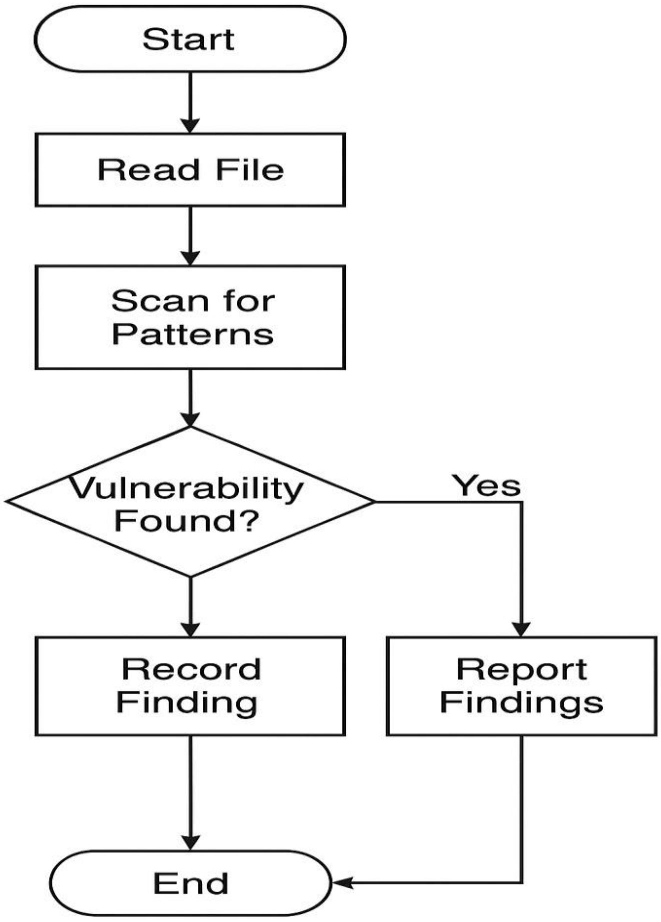
DEPARTMENT	**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**				
Semester: VII	**Course Title: Data Security & Privacy Lab**			**Course Code: 22AI73**	
PROGRAM QUESTION	4. Build a Vulnerability Analyser Tool in Python that can scan source code files (any programming language) for common security vulnerabilities or bad coding patterns.				
STUDENT NAMES	AYUSH MEHTA		USN	1DS22AI008	
LAB MARKS	**Program Execution (10)**	**Source Code (10)**	**Result (5)**	**Innovation/ Extra Features (5)**	**Total**
TOOLS USED	**Programming Language**: Python 3.x **Libraries**: os, re, argparse, json **Editor/IDE**: VS Code **Operating System**: Windows/Linux/MacOS **Testing Dataset**: Sample vulnerable source code files (.py, .js)				
MAP COURSE OUTCOME	**CO1**: Understand the importance of secure coding practices in software development. **CO2**: Apply Python programming to build tools that detect security vulnerabilities. **CO3**: Evaluate the effectiveness of security tools by testing them on sample projects.				
MAP PROGRAM OUTCOME	**PO1** (Engineering Knowledge): Applied knowledge of programming and security principles to develop the analyser tool. **PO2** (Problem Analysis): Identified potential security issues in code by analyzing patterns. **PO3** (Design/Development of Solutions): Designed a tool that helps in secure coding practices. **PO4** (Modern Tool Usage): Used Python as a modern tool to implement static code analysis.				
MAP PROGRAM SPECIFIC OUTCOME	**PSO1**: Ability to apply security concepts to develop reliable and safe software. **PSO2**: Ability to use programming knowledge to design and implement security solutions/tools. **PSO3**: Ability to analyze and mitigate vulnerabilities in real-world applications.				

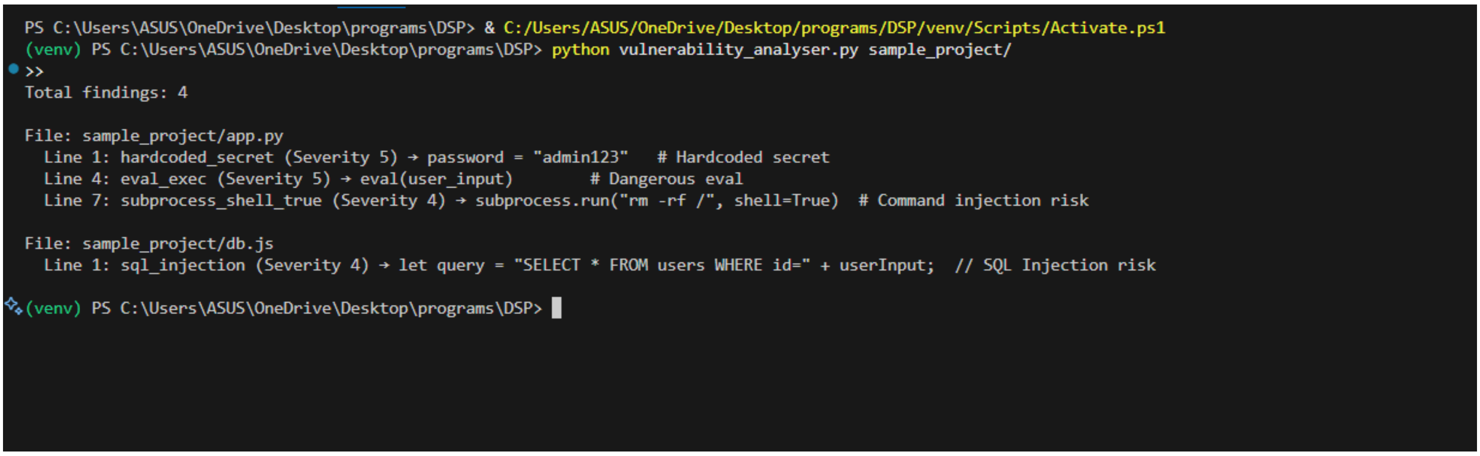
UML / SEQUENCE DIGRAM	Vulnerability Analyser Flow Diagram  Start Read File Scan for Patterns Vulnerability Found? Yes Record Finding Report Findings End
COURSE COORDINATOR	Dr ARUNA M G & Prof ENSTEIH SILVIA

Code Implementation

vulnerability_analyser.py

```
import os
import re
import argparse
import json

# ==============================
#  Vulnerability Detector Rules
# ==============================
DETECTORS = [
    {
        "name": "hardcoded_secret",
        "pattern": re.compile(r"(password|passwd|pwd|secret|api[_-]?key)\s*=\s*['\"].+['\"]", re.I),
        "severity": 5,
    },
    {
        "name": "eval_exec",
        "pattern": re.compile(r"\b(eval|exec)\s*\("),
        "severity": 5,
    },
    {
        "name": "subprocess_shell_true",
        "pattern": re.compile(
            r"subprocess\.Popen\(|subprocess\.call\(|subprocess\.run\([\s\S]*shell\s*=\s*True|Runtime\
.getRuntime\(\)\.exec\(|system\("
        ),
        "severity": 4,
    },
    {
        "name": "sql_injection",
        "pattern": re.compile(r"(SELECT|INSERT|UPDATE|DELETE).*\+.*", re.I),
        "severity": 4,
    },
    {
        "name": "insecure_deserialization",
        "pattern": re.compile(r"(pickle\.load|yaml\.load|ObjectInputStream)"),
        "severity": 4,
    },
]

# ==============================
#  Scan a single file
# ==============================
def scan_file(filepath):
    findings = []
    with open(filepath, "r", errors="ignore") as f:
        for lineno, line in enumerate(f, start=1):
            for det in DETECTORS:
                if det["pattern"].search(line):
                    findings.append({
                        "detector": det["name"],
                        "severity": det["severity"],
                        "line": lineno,
                        "code": line.strip()
                    })
    return findings

# ==============================
#  Scan a folder or file
# ==============================
def scan_path(path):
    results = {}
    if os.path.isfile(path):
        results[path] = scan_file(path)
    else:
        for root, _, files in os.walk(path):
            for f in files:
                if f.endswith((".py", ".js", ".java", ".php", ".c", ".cpp")):
                    filepath = os.path.join(root, f)
                    results[filepath] = scan_file(filepath)
    return results

# ==============================
#  Report results
# ==============================
def report_findings(findings, json_out=None):
    total = sum(len(v) for v in findings.values())
    print(f"Total findings: {total}\n")
    for f, issues in findings.items():
        if not issues:
            continue
        print(f"File: {f}")
        for issue in issues:
            print(f"  Line {issue['line']}: {issue['detector']} "
                  f"(Severity {issue['severity']}) → {issue['code']}")
        print()
    if json_out:
        with open(json_out, "w") as jf:
            json.dump(findings, jf, indent=2)
        print(f"Findings saved to {json_out}")

# ==============================
#  Main function
# ==============================
def main():
    parser = argparse.ArgumentParser(description="Source Code Vulnerability Analyser")
    parser.add_argument("path", help="File or directory to scan")
    parser.add_argument("--json", help="Save findings to JSON file")
    args = parser.parse_args()

    findings = scan_path(args.path)
    report_findings(findings, args.json)

if __name__ == "__main__":
    main()
```

Output



```
PS C:\Users\ASUS\OneDrive\Desktop\programs\DSP> & C:/Users/ASUS/OneDrive/Desktop/programs/DSP/venv/Scripts/Activate.ps1
(venv) PS C:\Users\ASUS\OneDrive\Desktop\programs\DSP> python vulnerability_analyser.py sample_project/
>>
Total findings: 4

File: sample_project/app.py
  Line 1: hardcoded_secret (Severity 5) → password = "admin123"   # Hardcoded secret
  Line 4: eval_exec (Severity 5) → eval(user_input)        # Dangerous eval
  Line 7: subprocess_shell_true (Severity 4) → subprocess.run("rm -rf /", shell=True)  # Command injection risk

File: sample_project/db.js
  Line 1: sql_injection (Severity 4) → let query = "SELECT * FROM users WHERE id=" + userInput;  // SQL Injection risk

(venv) PS C:\Users\ASUS\OneDrive\Desktop\programs\DSP>
```