

OBJECTIVE

I Develop a software framework in Python that simplifies the process of performing cybersecurity security analysis on a variety of devices.

BACKGROUND

- I Modern power systems are protected, controlled, and automated through a vast selection of digital products.
- I This opens the door to malicious actors trying to take advantage of these products to compromise critical systems for financial or political gain.
- I Protecting these critical systems requires cybersecurity testing to analyze vulnerabilities present within devices.
- I The issue is that security testing tools have a wide range of input requirements, output styles, and are not beginner friendly or efficient.

VALUE PROPOSITION

- I Purpleframe aims to standardize the use of these tools.
- I This is accomplished by allowing users to choose what tests to run, providing a simple user interface, and producing a formatted output that is easy to understand.
- I Using Purpleframe, users can confidently test, track, and analyze their digital systems for possible attack vectors.

KEY REQUIREMENTS

- I Must have both scripting and automatic options for initiating security tests.
- I Be able to run tests via Nmap, Nikto, and Slowhttptest.
- I Support incremental testing based on previous results.
- I Parse and archive results and provide an output containing user and machine-readable information.

CONCEPT DEVELOPMENT

I The initial plan included different client/server models using Docker containers.

FINAL DESIGN

1. A CLI takes input from the user.
2. The Core sends the Session Data to the Test Runner or the Script Engine.
3. The Test Runner triggers plugins to run the desired tests.
4. The Plugins are used to perform the security tests. As of right now, we have implemented 3 plugins: Nmap, Nikto, and Slowhttptest.
5. The results are returned as Test Artifacts. Any additional tests that can be ran automatically based on the results will be triggered.
6. The Test Artifacts are sent to the Artifact Manager. They are archived in an SQLite database and a Report will be generated.

I The Script Engine is an alternative way to provide input via a yaml configuration file. The data flow is the same as using the Test Runner.

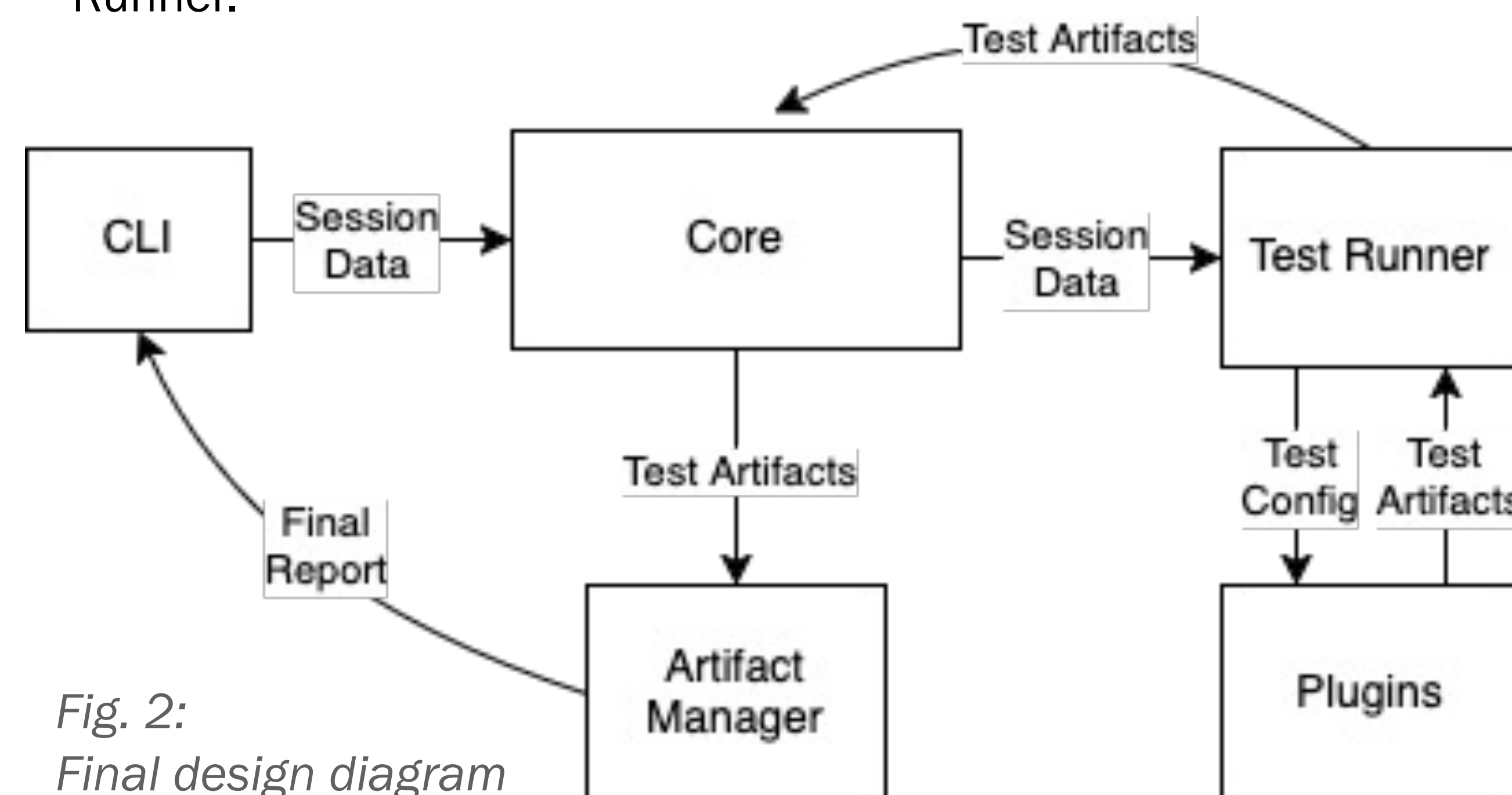
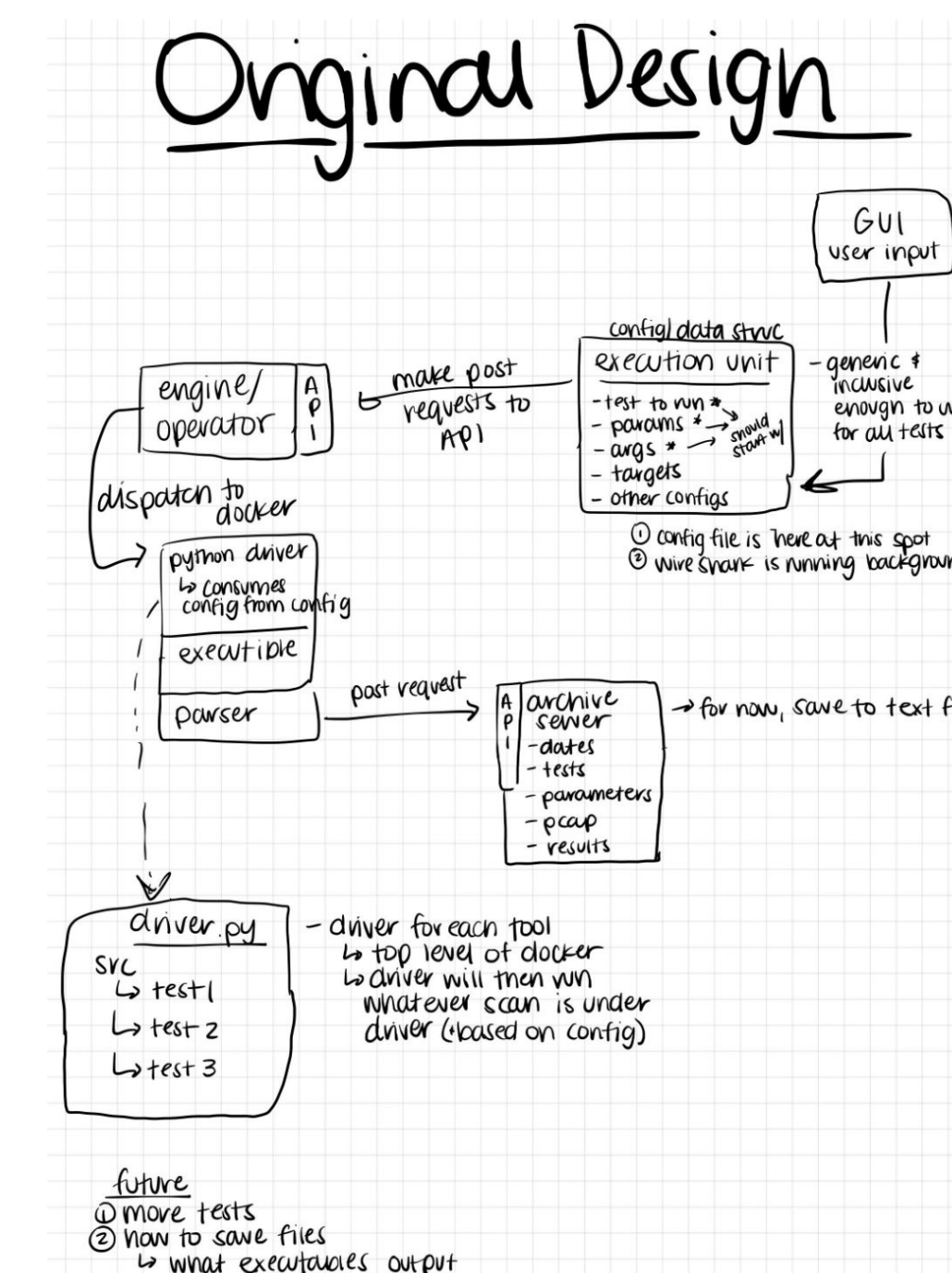


Fig. 2:
Final design diagram

Fig. 1: The original dataflow



VALIDATION & TESTING

- I We have been using Pytest to develop automated validation tests that ensure our variables have the correct values at runtime.
- I Below (Fig. 3) is an example of a Pytest that validates that a registered plugin and returns a compilation of test results.

```

from slowhttptest_parser import SlowhttptestParser
import os

def test_slowhttptest_parser():
    filepath = "/workspaces/PurpleFrame/plugins/slowhttptest/purpleframe_slowhttptest/slowhttp_outputs/2025-04-14_04:56:53/2025-04-14_04:56:53.csv"
    assert os.path.exists(filepath), f"CSV file not found: {filepath}"

    parser = SlowhttptestParser()
    artifact = parser.parse_slowhttptest_output(filepath)
    assert artifact is not None
  
```

- I Our software has met the key requirements and Nmap, Nikto, and Slowhttptest plugins are fully implemented.

Fig. 4: Purpleframe running a test

```

(.venv) [root@aa353fd7360b] - [workspaces/PurpleFrame]
└─ purpleframe script ./test/samples/sample_script.yaml -vv
2025-04-16 19:17:48 - purpleframe - INFO - Running single test: nmap.default_scan
2025-04-16 19:17:50 - purpleframe.scripting - INFO - Test Nmap Port Scan result: {
  "hosts": [
    {
      "host_addr": {
        "ip_address": "127.0.0.1"
      },
      "hostnames": [
        {
          "hostname": "localhost"
        }
      ],
      "ports": [],
      "os_info": null
    }
  ]
}
  
```

CONCLUSION & RECOMMENDATIONS

- I Purpleframe can automate security testing to make secure development of devices more efficient and accessible.
- I More plugins can be added in the future to create more comprehensive testing abilities.

ACKNOWLEDGEMENTS

- I Thanks to Joe Ferguson and Tracy Young from SEL for providing guidance on our project.



2025 Capstone Project