Stub Testing for Vulnerability Detection in LLM Training

Sara Gomez

ARISE Lab – The Fu Foundation School of Engineering & Applied Science

Introduction

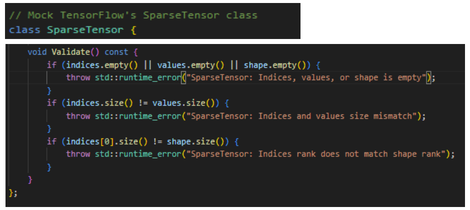
- PrimeVUL was previously introduced as a dataset for training and evaluating large language models (LLMs) for vulnerability detection (VD), but research revealed **the considerable gap between capabilities and practical requirements** for deploying LLMs in security roles.
- This **project aims to enhance the detection and fixing of security vulnerabilitie**s in open-source codebases through **stub testing.**
- By utilizing stub tests to recreate a sample vulnerabilities in TensorFlow codebase, this project looks to **validate fixes and build a test case codebase.**
- Hope to underscore the **value of automated test generation for training LLMs for VD** via **dynamic vulnerability tracing.**

Stub Test Method

Example shown is a vulnerability caused by lack of input validation on 'AddManySparseToTensorsMap' function in TensorFlow.

1. Create **mock classes** to simulate objects & their behaviors

Pre-processing to set simulated environment

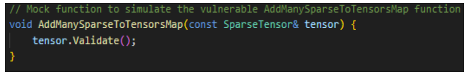


```cpp
// Mock TensorFlow's SparseTensor class
class SparseTensor {
```

```cpp
    void Validate() const {
        if (indices.empty() || values.empty() || shape.empty()) {
            throw std::runtime_error("SparseTensor: Indices, values, or shape is empty");
        }
        if (indices.size() != values.size()) {
            throw std::runtime_error("SparseTensor: Indices and values size mismatch");
        }
        if (indices[0].size() != shape.size()) {
            throw std::runtime_error("SparseTensor: Indices rank does not match shape rank");
        }
    }
};
```

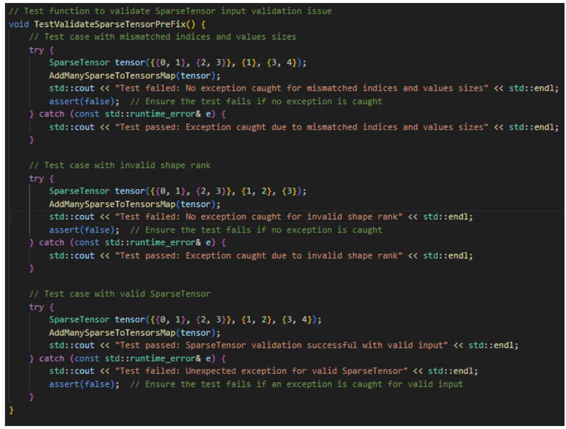
2. Create **mock functions** to simulate function calls & expected results

Focusing on vulnerable functions



```cpp
// Mock function to simulate the vulnerable AddManySparseToTensorsMap function
void AddManySparseToTensorsMap(const SparseTensor& tensor) {
    tensor.Validate();
}
```

3. Develop **test cases** to create vulnerable scenarios

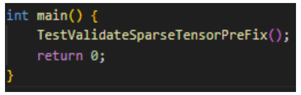


```cpp
// Test function to validate SparseTensor input validation issue
void TestValidateSparseTensorPreFix() {
    // Test case with mismatched indices and values sizes
    try {
        SparseTensor tensor({{0, 1}, {2, 3}}, {1}, {3, 4});
        AddManySparseToTensorsMap(tensor);
        std::cout << "Test failed: No exception caught for mismatched indices and values sizes" << std::endl;
        assert(false);  // Ensure the test fails if no exception is caught
    } catch (const std::runtime_error& e) {
        std::cout << "Test passed: Exception caught due to mismatched indices and values sizes" << std::endl;
    }

    // Test case with invalid shape rank
    try {
        SparseTensor tensor({{0, 1}, {2, 3}}, {1, 2}, {3});
        AddManySparseToTensorsMap(tensor);
        std::cout << "Test failed: No exception caught for invalid shape rank" << std::endl;
        assert(false);  // Ensure the test fails if no exception is caught
    } catch (const std::runtime_error& e) {
        std::cout << "Test passed: Exception caught due to invalid shape rank" << std::endl;
    }

    // Test case with valid SparseTensor
    try {
        SparseTensor tensor({{0, 1}, {2, 3}}, {1, 2}, {3, 4});
        AddManySparseToTensorsMap(tensor);
        std::cout << "Test passed: SparseTensor validation successful with valid input" << std::endl;
    } catch (const std::runtime_error& e) {
        std::cout << "Test failed: Unexpected exception for valid SparseTensor" << std::endl;
        assert(false);  // Ensure the test fails if an exception is caught for valid input
    }
}
```

Test cases with mismatched indices and value sizes.

Test case with valid input

4. Implement in **main function** to execute test cases



```cpp
int main() {
    TestValidateSparseTensorPreFix();
    return 0;
}
```

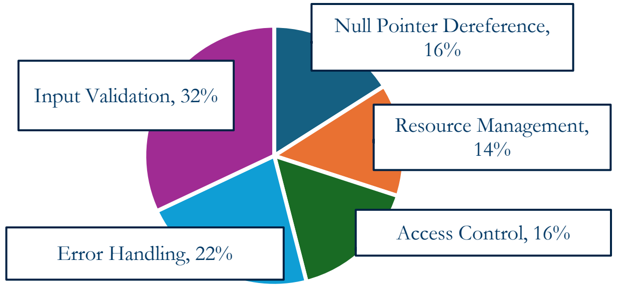
5. Run in **isolation** to validate vulnerabilities

Test cases written in C++ and run in isolation VisualStudio Code. Want to determine if the vulnerability was successfully recreated or not. Test cases with successful recreations added to database.

Results

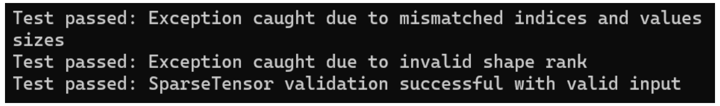
Initially, used a sample of 50 confirmed vulnerabilities across codebases to create a graph of **common vulnerabilities.**

Common Vulnerabilities in Open-Source Codebases



Utilized the **stub test method** to recreate 22 vulnerabilities from TensorFlow. Example of successful vulnerability recreation shown using same example of validation check added – shows appropriate error handling & input validation. **Compiled database of testcases** of the TensorFlow sample.

Two edge cases tested & both vulnerabilities caught.



```
Test passed: Exception caught due to mismatched indices and values
sizes
Test passed: Exception caught due to invalid shape rank
Test passed: SparseTensor validation successful with valid input
```

Validation added, function successful.

Conclusion & Next Steps

Stub tests proved effective at identifying and addressing security vulnerabilities. Provides modality of information – allowing for **dynamic vulnerability tracing** during development.

- Experimentally created a **test case database** to recreate vulnerabilities by simulating edge cases.
- Value proven in using automated test generation to train **LLMs for VD.**

Acknowledgements

ARiSE Lab
Prof Baishakhi Ray & Yangrubio (Robin) Ding

My Contact

sara.m.gomez@vanderbilt.edu
4013021176
Scan for LinkedIn: