

CHAPTER 3 - RESEARCH METHODS

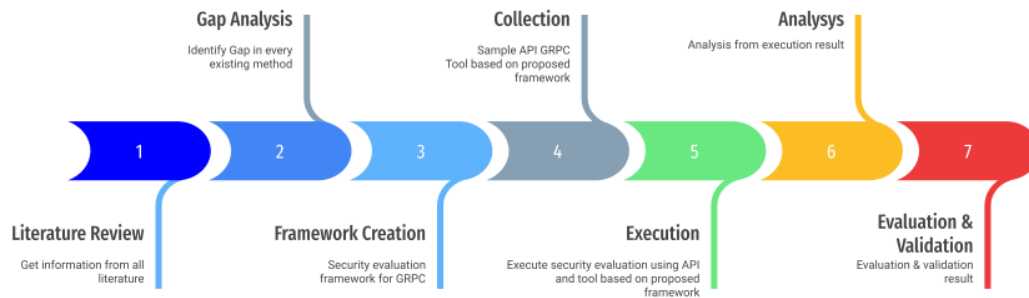


Figure 3.1 Research methodology

3.1. Research Methodology

The research methodology in this case involves conducting a literature review and performing a security evaluation using the proposed framework developed in this thesis. The focus of these activities is outlined in the research framework, as depicted in Figure 3.1, which guides the process to achieve the results in the security evaluation of gRPC.

3.2. Research Framework

The GSTF (gRPC Security Testing Framework) identified to be proposed at this stage can be seen in Figure 3.2. In this framework, it is divided into several parts including Extraction of Payload, Code Generation, Test Case Creation and Execution. In the "**Execution**" phase, automation will be implemented since the required objects, such as payloads, stubs, and test cases, have already been created in before process and the payload will be adjusted based on its data type based that will be created as a test case. At this stage, the process involves simply running these objects and validating the results. Regarding the success criteria for implementing this framework, since the primary issue in this research problem is the inefficiency of manual testing, which results in prolonged testing times, it is expected that this framework will significantly

reduce testing time by 60% to 80% or even more depending on the context. The exact percentage of improvement depends on factors such as:

- **Test Case Complexity:** Automation is more effective for repetitive, time-consuming tasks, leaving complex, exploratory tests to manual efforts.
- **Coverage:** Automated tools can increase test coverage, catching more issues than manual testing would in the same timeframe.
- **Speed:** Automation speeds up execution, especially for regression testing and large test suites.
- **Consistency:** Automation reduces human error, ensuring tests are run in the same way every time.

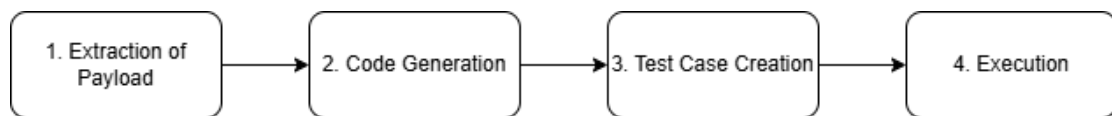


Figure 3.2 GSTF (gRPC Security Testing Framework)

3.2.1. Extraction of Payload

This stage involves extracting the payload for testing, compiling it into a JSON/YAML file, and mapping important attributes such as data type, attack name, payload data, and expected results according to the Figure 3.3. The process includes:

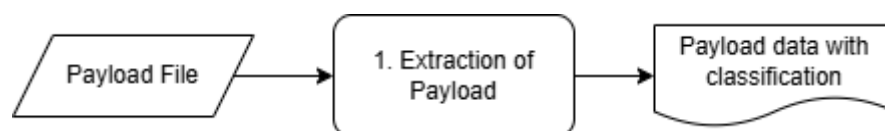


Figure 3.3 Phase 1 - Extraction of payload

- **Input:** Collecting payloads for common vulnerabilities such as SQL injection (SQLi), cross-site scripting (XSS), local file inclusion (LFI), and server-side request forgery (SSRF) in JSON/YAML format.

- **Process:** In this process, the input file will be extracted by mapping the payload on each attribute including data type, attack name, payload data, and expected results.
- **Output:** The output that will be provided by the payload extraction process is an object that has been mapped based on attributes.

3.2.2. Code Generation

The proto file is processed to generate stubs, gRPC functions that handle requests and responses according to the Figure 3.4. Using Python, automation ensures all request methods are seamlessly integrated with the test payload. The steps are:

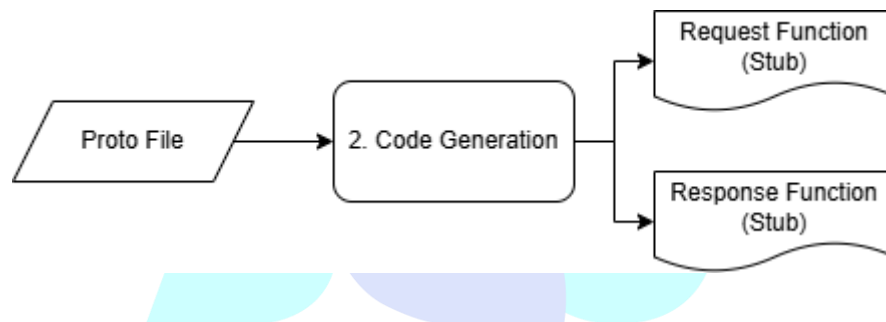


Figure 3.4 Phase 2 - Code generation

- **Input:** The input to this process is a proto file from the gRPC service.
- **Process:** In this process, a proto file will be generated to become a request and response function.
- **Output:** The main output of this process is a stub or request and response function.

3.2.3. Test Case Creation

The stub parameters are used to generate test data, including valid and invalid inputs, based on vulnerabilities in payloads. The payload from the JSON/YAML file is paired with the stub parameters to create test cases and serialized into a protocol buffer according to the Figure 3.5. The steps include:

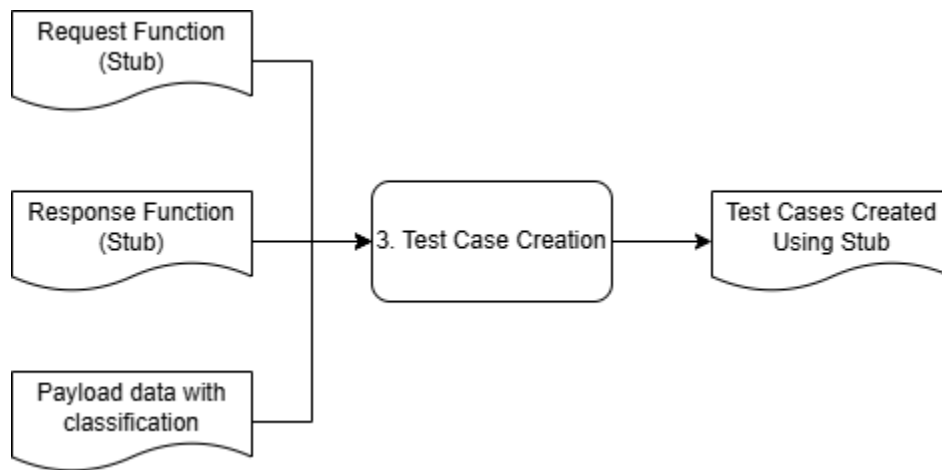


Figure 3.5 Phase 3 - Test case creation

- Input: The input to this process is the payload generated from Phase 1 and the stub generated from phase 2.
- Process: In this process, a payload will be combined with a stub or request and response function in gRPC based on the data type to become a test case and convert to protocol buffer.
- Output: an object in the form of a combined test case and payload that can be used for the next phase.

3.2.4. Execution

Test cases are executed by sending payloads to the gRPC API, and the responses are evaluated and validated to determine success & error rates, as well as pass and fail in the test cases by comparing the results according to the Figure 3.6. Then, the results are recorded for reporting. The process includes:

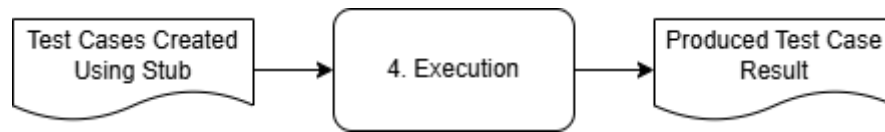


Figure 3.6 Phase 4 - Execution

- Input: The input for this process is the object that was created in phase 3 which will later be used in carrying out the execution.
- Process: Sending payloads via stub to gRPC services with multiple sub processes including:
 - Evaluation: Evaluate the **success** and **error rates** of recording to ensure the system handles invalid input correctly without crashing.
 - Validation: **Comparing responses** to expected results (e.g., responses data, status codes, errors from application or system).
 - Verification: **Manual verification** is performed to confirm the validation results by identifying false positives through direct execution and validation.
- Output: Recording results, highlighting vulnerabilities, vulnerability details & detail **execution time** with coverage details. To find out the efficiency of execution time between manual testing and GSTF testing, use the following formula:

$$((ETM - ETG) / ETM) * 100\% \quad (3.1)$$

ETM = Execution time for manual testing

ETG = Execution time for GSTF testing

3.3. Scope of Test

The PoC testing will be performed on a sample API developed using gRPC, utilizing all functions specified in the proto file. **The security testing execution** process will include several testing and comparisons, such as:

- Manual security testing
- GSTF security testing
- Comparison between manual and GSTF testing based on execution time
- Comparison of GSTF security testing results with other tools

For the tool built from the GSTF framework, the primary focus will be on evaluation, validation, and verification to accurately identify false positives. The goal of this process is to evaluate the execution of requests, mainly focusing on execution time, to achieve a minimum efficiency improvement of 60% as stated in the hypothesis when comparing between manual testing and GSTF testing.

3.4. Evaluation

The evaluation phase focuses on executing test cases by sending payloads to gRPC APIs using the GSTF framework. During this phase, the success and error rates of test executions are measured to assess the framework's ability to handle invalid inputs, special cases, and vulnerabilities without causing any damage. The feasibility, effectiveness, and adaptability of the framework to different gRPC implementations are also evaluated by creating mechanisms that can work with different types of gRPC. The results, including success/error rates and recorded data, are documented for reporting and analysis. In addition, this process aims to provide insight into the success and failure rates, as well as test coverage details.

3.5. Validation

Validation is conducted after executing requests by comparing the system's response with the expected outcomes in the test cases with coverage details. This ensures that the results align with predefined criteria, such as status codes and error messages. Validation is applied to both GSTF and manual testing to highlight their differences, particularly in terms of efficiency and accuracy. The process includes the following steps:

1. Response validation
 - Compare the received response (status codes, error messages) with the expected output.
 - Identify deviations or unexpected behaviors.
2. Manual verification
 - Identify and document false positives and false negatives.
 - Compare results to assess the reliability of GSTF in detecting unique vulnerabilities.
3. Result documentation
 - Present findings using tables, charts, and graphs for better visualization.