

CHAPTER 2 - LITERATURE REVIEW

2.1. Business Transformation

The digital transformation of businesses is increasingly driven by the integration of Application Programming Interfaces (APIs), which facilitate the creation of digital ecosystems and enable new business models. APIs are critical in the transition to more flexible and dynamic systems, offering both opportunities and challenges for organizations across sectors. APIs are at the heart of companies' digital transformation efforts, serving as the backbone for creating a digital ecosystem that enables seamless integration of services and platforms. APIs enable businesses to innovate and offer new value propositions by connecting multiple systems and facilitating data exchange (Krasimir Todorov Shishmano et al., 2022; Naga Mallika Gunturu, 2022; Rahul C. Basole, 2016). In the banking sector, APIs are transforming traditional business models by enabling open banking, which increases customer engagement and competition (Anshu Premchand and Anurag Choudhry, 2019). As a result, businesses are undergoing significant transformation, especially in transaction processes that leverage APIs to connect customers with services. However, this transformation poses challenges in API-based transactions, which can create problems that require effective solutions.

2.1.1. Business Challenges in API Transactions

API transactions and performance present significant business challenges in the modern ecosystem. Transaction issues stem from the complexity of integrating multiple partners, inconsistent standards, and misaligned expectations, leading to failures and missed opportunities. Trust and security issues further complicate transactions, as insecure APIs can expose sensitive data, resulting in financial loss and reputational damage. On the performance side, scalability challenges during periods of high demand, such as slow response times or outages, frustrate customers and decrease revenue, while inadequate monitoring and testing delays detection of bottlenecks or

downtime, disrupting business continuity (Sebastien Andreo and Jan Bosch, 2019). Addressing these issues requires robust API design, effective performance optimization, and strategic ecosystem management. In fact, there are currently many APIs that focus on performance to provide API optimization that will later have a significant impact on the business.

2.1.2. Business Optimization in API-Driven Transactions

Optimizing API performance in transactional businesses involves a combination of resource allocation, service composition, and service scheduling (Tomasz Górski and Adrian P. WOźniak, 2021). Heuristic methods, especially genetic algorithms, are critical in improving the efficiency of business processes executed through web services. By focusing on these optimization strategies, businesses can improve the performance of their APIs, leading to faster and more reliable transactions. Results from another study analyzed over 120 companies that adopted APIs, correlating API adoption with performance improvements using a difference-in-differences approach (Seth G. Benzell et al., 2017). The results showed that API adoption significantly increased sales, market capitalization, intangible assets, and, in some cases, reduced operating costs. The impact was most pronounced in IT companies but extended to other industries as well. While internal APIs drove sales growth more effectively than B2C APIs, API usage intensity also correlated with financial results, although the causality remains unclear. Specifically, API adoption increased market capitalization by about 12.7%, with open developer portals and diverse API types increasing profits but not critical to success. The previous two studies provide valuable insights into performance, but ignore the important aspect of security. Security is just as important as performance, as a lack of security can lead to significant losses in the event of an API incident.

2.1.3. Common API Issue

Common API issues include weak authentication, excessive permissions, inadequate input validation, and poor error handling, leading to vulnerabilities such as injection

attacks, privilege escalation, and data exposure. Lack of rate limiting and poor encryption enforcement leave APIs vulnerable to denial of service, brute force, and data security risks (Wei Lu et al., 2022). Additionally, inconsistent logging and outdated or incomplete documentation hamper security tracking and developer understanding. Ambiguous naming, unclear relationships between API types, and excessive flexibility complicate usability, especially for less experienced developers (Marco Piccioni et al., 2013). Cognitive overhead, unpredictable behavior, and a steep learning curve further exacerbate these challenges. To address these challenges, APIs require robust security measures, clear documentation, intuitive naming, simplified design, and proactive monitoring.

2.1.4. Balancing Performance and Security

Many modern APIs prioritize speed, with gRPC being a popular choice due to its high performance and low latency (Kasun Indrasiri and Danesh Kuruppu, 2020). At my previous company, gRPC was implemented as the primary API due to these benefits. Despite its superior performance, a successful cyberattack that compromises personal financial information can result in significant losses to shareholder wealth, reduced sales growth, and negative impacts on stock prices in the affected industry (Shinichi Kamiya et al., 2018). Security testing identifies vulnerabilities in a system and assigns a risk score, helping to minimize potential losses caused by cyberattacks (Kyle Courtney et al., 2023). Therefore, despite its high performance and low latency, it is critical to balance these benefits with strong security measures, as security is as important to businesses as performance.

2.2. API Evolution

An Application Programming Interface (API) is a software application comprising a collection of commands and functions designed to communicate with external applications, executing standard operations. Renowned for its user-friendly nature, flexibility, and efficiency, an API acts as a seamless connector, facilitating smooth

interaction among modules, software, and developers. In the contemporary landscape, APIs are crucial in developing advanced mobile applications, Software as a Service (SaaS) platforms, and web applications (Ronghua Sun et al., 2022). APIs are now available in various types and delivery methods, reflecting evolving needs. New API methods with distinct characteristics, such as SOAP, REST, GraphQL, and gRPC, have emerged, with this research focusing on gRPC here.

2.2.1. gRPC

gRPC was created by Google in 2015 to address the need for a more efficient and scalable communication protocol for microservices (Kasun Indrasiri and Danesh Kuruppu, 2020). The “g” in gRPC originally stood for Google, but is now considered “generic,” reflecting its broader use. gRPC uses HTTP/2 and Protocol Buffers (protobuf) to ensure high performance, low latency, and efficient data serialization, making it ideal for real-time communication between services in distributed systems. Historically, gRPC evolved from Google’s internal project, “Stubby,” a highly efficient Remote Procedure Call (RPC) system used within Google. The goal was to create a framework that could handle large-scale, complex microservice architectures while improving upon traditional methods like REST, which often suffer from performance bottlenecks due to text-based formats like JSON and XML. By open sourcing gRPC, Google is making this robust solution available to external developers, enabling the creation of modern APIs that offer greater speed and security across platforms.

Google Remote Procedural Protocol (gRPC) is a RPC architecture-based technology that can be used as an API using HTTP/2 as the protocol with the message to be converted into a protocol buffer message (Gelei Deng et al., 2023). The gRPC API is different from APIs in general, where this API uses a code generation mechanism created on the client side called a stub is usually called a contract because it contains data that is used to communicate with the service (Anthony Giretti, 2022; Gelei Deng et al., 2023; Lukasz Kamiński et al., 2023). Later, this stub will be used in the application to make requests and get responses. The following is an illustration of how the gRPC API works.

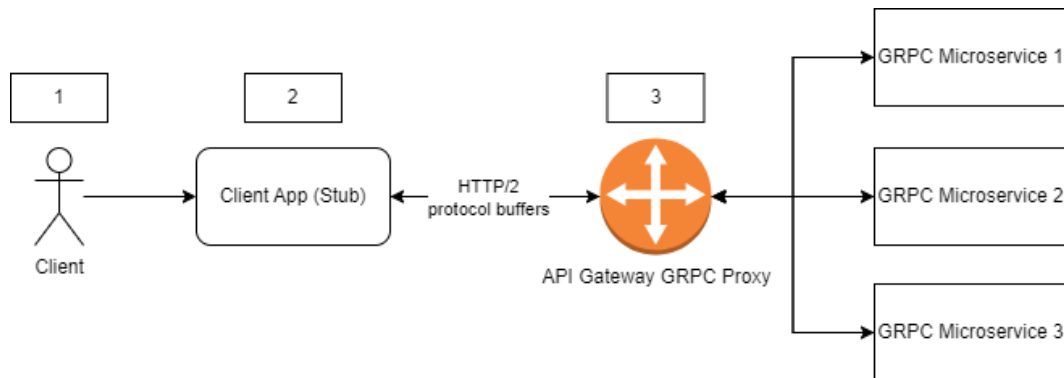


Figure 2.1 gRPC API Architecture (Anthony Giretti, 2022)

From the illustration Figure 2.1 it is divided into 3 main processes as follows:

1. The client (actor) initiates a request through the Client App, which can be either a web or mobile application seamlessly integrated with the gRPC API.
2. The Client App generates a stub function from proto file, and this stub, in turn, initiates a request. The request, facilitated by the request function, is then transmitted and subsequently received by the response function. During the transmission, the message undergoes conversion into a protocol buffer format and is sent via the HTTP/2 protocol to the API gateway gRPC proxy. The address used in this transmission corresponds to the function name employed during the initial request.

```
syntax = "proto3";

service CountryService {
  rpc GetById (CountrySearchRequest) returns (CountryReply) {}
}

message CountrySearchRequest {
  int32 CountryId = 1;
}

message CountryReply {
  int32 Id = 1;
  string Name = 2;
  string Description = 3;
}
```

Figure 2.2 Sample proto file (Anthony Giretti, 2022)

3. The API Gateway redirects this request to a microservice with a matching function name, which was invoked by the client. The microservice processes the request and generates a response, which is then accessible to the Client App through the response function for data extraction.

With the information above, it is found that before the data is sent, the data is converted into a protocol buffer and uses HTTP/2 as the protocol. Then the API address is not degenerated on the server side, but the function name is degenerated from the proto file. This is quite different from how APIs work in general, because gRPC is a client side generated function (Lukasz Kamiński et al., 2023). gRPC does not have an HTTP method and will send directly to the API side without any defined methods (Bohdan Marii and Ivan Zholubak, 2022). gRPC has a header called metadata, which can be used to store data such as tokens and others.

Currently gRPC is a technology that is used for various things, especially APIs and can be combined with various architectures (Collin T Blakley et al., 2024). Therefore, as gRPC is increasingly used in many architectures, it is necessary to carry out proper testing to reduce the risk of threats from cybersecurity.

2.2.1.1. Type of gRPC Service

gRPC offers various types of services that can support an application as a whole. These services include:

- Unary

The client initiates the process by sending an RPC (Remote Procedure Call) that includes the method name, metadata, and a message to the gRPC server. The server then responds by sending back metadata and a message to the gRPC client that can be seen in Figure 2.3.

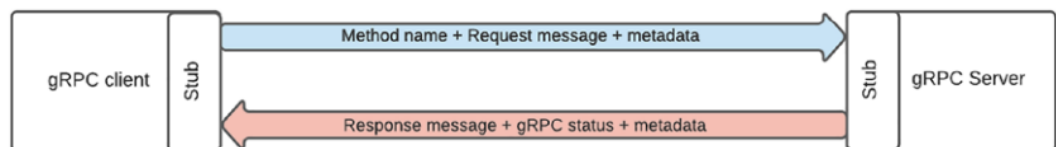


Figure 2.3 Unary type process (Anthony Giretti, 2022)

- Server-streaming

The client initiates the process by sending an RPC (Remote Procedure Call) that includes the method name, metadata, and a message to the gRPC server. The server will send a streaming response until all responses are completed, after which it will send a gRPC status and metadata to conclude the process and the process can be seen in Figure 2.4.

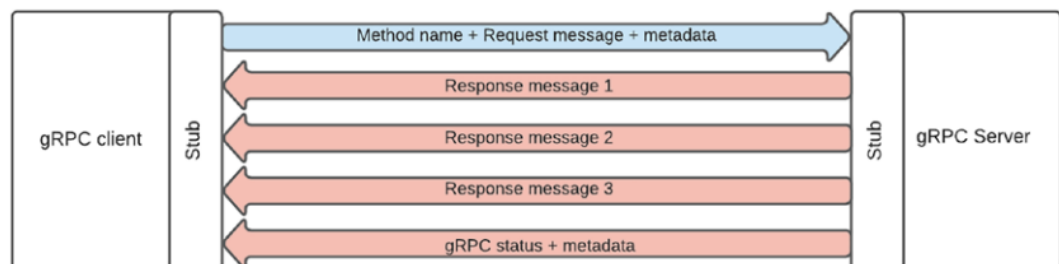


Figure 2.4 Server-streaming type process (Anthony Giretti, 2022)

- Client-streaming

The client initiates the process by sending an RPC (Remote Procedure Call) that includes the method name, metadata, and a streaming message to the gRPC

server. The server will eventually send a response to the gRPC client that includes a message, status, and metadata that can be seen in Figure 2.5.

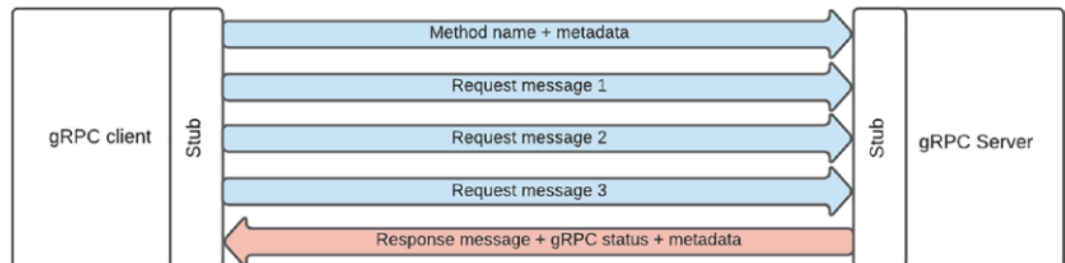


Figure 2.5 Client-streaming type process (Anthony Giretti, 2022)

- Bidirectional

In the bidirectional type, both the gRPC client and server can send messages in a streaming manner, with no specific order of transmission. With this type, the server will return a response message and, once the request is complete, will conclude with a gRPC status and metadata, for the sequence process that can be seen in Figure 2.6.

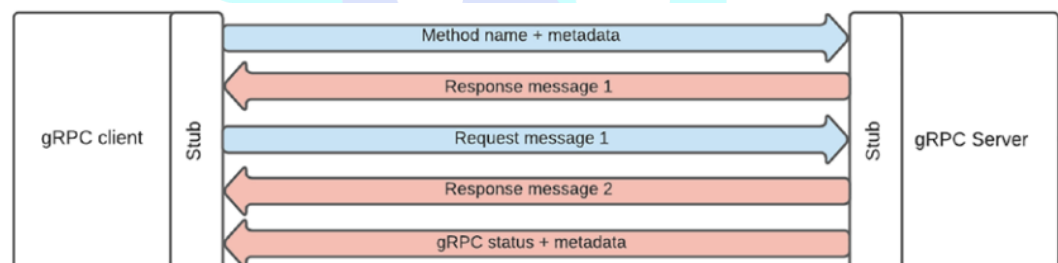


Figure 2.6 Bidirectional streaming type process (Anthony Giretti, 2022)

Unlike REST API, which only supports unary type calls (where the client sends a request and receives a response, and the server accepts the request and returns a response), gRPC can support all four types of calls.

2.2.1.2. Object Type In gRPC

In a gRPC function or stub, as shown for sample in Figure 2.2, the data types are defined by a proto file, which is then used in the generated function (Protobuf, 2024a). These data types must match the data types used during the request. gRPC supports various data types, including:

- String is a character sequence encoded in UTF-8 or 7-bit ASCII, with a maximum length of 2^{32} characters.
- Double is a 64-bit precision floating-point number used for storing decimal values.
- Float is a 32-bit precision floating-point number for representing decimal values
- Integer types, including int32, int64, and their signed versions, are variable-length encoded numbers, with signed integers more efficient for negative values. Fixed integers use fixed-length encoding
- Unsigned integers, such as uint32 and uint64, are non-negative numbers that use variable-length encoding
- Boolean values, represented as true or false, indicate binary states
- Byte refers to a sequence of arbitrary bytes, with a length limit of 2^{32} bytes
- Fixed types are integers with fixed-length encoding, using 4 or 8 bytes, optimized for larger numbers.
- Signed integers (sint32, sint64) use variable-length encoding to more efficiently represent negative numbers

Based on the types of objects above, gRPC has very specific specifications for each variable and can be adjusted to application needs because it supports almost all types of objects.

2.2.1.3. Status Code In gRPC

Similar to other APIs such as REST APIs with 2xx status codes, gRPC has a unique set of RPC status codes. These codes are used to provide information about the request sent and are included in the response to indicate whether the request was successful or not (Anthony Giretti, 2022). Each status code comes with its own description, as detailed in Table 2.1.

Table 2.1 gRPC Status Code (Anthony Giretti, 2022)

Code	Number	Description
OK	0	Success.
CANCELLED	1	The operation was cancelled.
UNKNOWN	2	Unknown error.
INVALID_ARGUMENT	3	The client specified an invalid argument.
DEADLINE_EXCEEDED	4	The deadline expired before the operation could complete.
NOT_FOUND	5	The requested entity was not found.
ALREADY_EXISTS	6	The entity already exists.
PERMISSION_DENIED	7	The caller doesn't have the required permission.
RESOURCE_EXHAUSTED	8	The requested resource has been exhausted.
FAILED_PRECONDITION	9	A precondition has failed.
ABORTED	10	The operation was aborted.
OUT_OF_RANGE	11	An invalid argument causes that exception during its cast.

UNIMPLEMENTED	12	The operation is not implemented.
INTERNAL	13	Internal error.
UNAVAILABLE	14	The service is currently unavailable.
DATA_LOSS	15	Unrecoverable data loss or corruption.

2.2.2. REST

Proposed in the year 2000, Representational State Transfer (REST) is a software architecture that fundamentally governs the functioning of large-scale systems on the internet, notably the Web. Within the REST architecture, system behavior is constrained, particularly on the interface side, allowing users to interact with data through the CRUD (Create, Read, Update, and Delete) mechanism (Roy Thomas Fielding and Richard N. Taylor, 2000). Basically, the REST API makes requests directly from the client to server that can be seen on Figure 2.7 with a protocol can use HTTP/1 and HTTP/2 (Bohdan Marii and Ivan Zholubak, 2022).



Figure 2.7 REST API Architecture (Anthony Giretti, 2022; Bohdan Marii and Ivan Zholubak, 2022)

When a request is sent, the REST API will return resource status information when the request is made by the client, while the message formats used are generally JSON (Javascript Object Notation), HTML and plaintext format (Amid Golmohammadi et al., 2023). The REST API has a request method mechanism that is used to make requests, including GET, POST, PUT, PATCH and others (Amid Golmohammadi et al., 2023; Bohdan Marii and Ivan Zholubak, 2022) that makes requests can be defined with methods according to needs.

2.2.3. SOAP

SOAP, or Simple Object Access Protocol, is a messaging protocol for exchanging structured information in web service implementations (Enes Bajrami et al., 2024). The protocol primarily uses XML as its message format, making it platform-independent and highly interoperable. Designed to be transport and language-independent, SOAP is compatible with HTTP, SMTP, and other application-layer protocols. The protocol is commonly used in enterprise-level applications due to its robustness and support for advanced security, addressing, and transaction features through standards such as WS-Security.

2.2.4. GraphQL

GraphQL is a declarative query language and server-side runtime designed for APIs, allowing clients to specify exactly what data they need from the server (Aaron Lazar et al., 2021). This flexibility is particularly useful for applications that retrieve complex, nested data structures, as it reduces the number of requests required to retrieve information. Unlike REST, which often requires multiple endpoints, GraphQL uses a single endpoint to serve a variety of data requirements, leading to optimized performance in data retrieval and reduced data over- or under-retrieval. The data journey began at Facebook in 2011. At the time, Facebook was facing performance challenges on mobile devices, where data requirements were increasingly complex and traditional APIs were inefficient at handling hierarchical data structures. Engineers Lee Byron, Dan Schafer, and Nick Schrock developed GraphQL to improve the efficiency of the news feed feature in mobile applications. This internal solution was expanded across Facebook's infrastructure and, in 2015, was released as an open-source project. The open-source release spurred widespread adoption, with companies such as Netflix also developing similar efficient API solutions.

2.2.5. Comparison of gRPC and Other API

Observing information in point 2.2.1 above related to the gRPC API and point 2.2.2 above concerning the REST API, distinctions between the processes of gRPC and

REST API are evident, as illustrated in Figure 2.1 and Figure 2.7. These differences include:

Table 2.2 Comparison between gRPC and other API based on API Component

Component	gRPC	REST	SOAP	GraphQL
HTTP Method	No HTTP method	Use HTTP method like GET, POST, HEAD and others	POST	POST
Protocol	HTTP/2	HTTP/1 and HTTP/2	HTTP/1.1, SMTP	HTTP/1.1
Header	Metadata	HTTP header	SOAP-specific headers	Standard HTTP headers
Format Communication	Protocol buffer	JSON, HTML, text and others	XML	Text (JSON)
Supported Browser	gRPC-web and a proxy layer	Various browsers	Limited support	Support
Request Method	Code generation	Request directly	Message-based	Query-based with custom schema definitions
Performance	High performance, low latency	Moderate, can vary	Generally slower	Moderate, depends on complexity

gRPC (Bohdan Marii and Ivan Zholubak, 2022), REST (Amid Golmohammadi et al., 2023; Bohdan Marii and Ivan Zholubak, 2022), SOAP (Enes Bajrami et al., 2024), GraphQL (Aaron Lazar et al., 2021)

Responding to the Table 2.2, there are important points that need to be discussed in the process:

1. gRPC involves a client-side or code generation mechanism, while the other API can directly initiate requests to the server. Therefore, the framework must be able to perform **code generation** in one of its processes.
2. When transmitting data from the client to the API, gRPC utilizes a protocol buffer format, while other APIs typically employ JSON, text and XML for data representation. Therefore, the protocol buffer is a component that is very vulnerable to exploitation considering that this component is a component used in transmitting data or exchanging data (Nihala Basheer et al., 2024). From the previous information, it has indirectly answered the question in Q1 and is the

same as Hypothesis 1 regarding components that are easily manipulated. Oleh karena itu,

3. In terms of performance, gRPC has high performance and low latency, unlike other APIs which have lower performance.

From the above points, conclude based on the collected data that gRPC stands out, especially in terms of performance, by showing significant differences in other components compared to other APIs (Kasun Indrasiri and Danesh Kuruppu, 2020). This is interesting, as its high performance makes it worthy of further research, considering the possibility that gRPC could become a widely used API in the future and this research helps in identifying vulnerabilities with frameworks tailored to gRPC.

2.2.5.1. gRPC API Criteria

Based on the comparison results of each API reviewed in 2.2.5 above, the requirements were established to ensure that the proposed framework meets all necessary criteria. These requirements include:

1. Since gRPC uses request method **code generation** in the form of stubs, the framework should be able to support this functionality.
2. gRPC requests utilize protocol buffers, so the framework should be capable of **executing** requests by converting them into protocol buffer format.
3. During testing, the framework should effectively **manage payloads** and **combine them** with the generated stubs from the code generation process.

2.3. API Security Testing

API security testing helps detect privacy disclosure and functional weaknesses in communication applications, thereby reducing security risks and improving system reliability (Shanling Li et al., 2023). By identifying vulnerabilities such as improper authentication, data leakage, or weak encryption mechanisms, security testing ensures that APIs meet stringent security requirements, especially for gRPC APIs, which are the primary focus of this research. This proactive approach protects sensitive user data, prevents unauthorized access, and reduces the potential for exploitation by attackers. In securing such exploitation, you can use the automated API method. Automated API testing and scanning involves using tools and techniques to automatically create and execute test cases for APIs, ensuring their functionality, reliability, and security despite the potential for false positives (Andrea Arcuri, 2019; Jayshish M. Popat et al., 2021). However, integrating manual and automated API security testing into the API development cycle improves regulatory compliance and strengthens trust in the communications platform, ultimately supporting business continuity and user satisfaction. On average, organizations often report that automation can handle 60% until 80% of testing (kms-sulition, 2024), leaving the remaining portion for manual testing where human insight is necessary. As a result, testing will become more efficient because it has been integrated into the lifecycle and is carried out through automation and manual testing. Furthermore, leveraging errors found during testing is critical to improving APIs, as analyzing these errors helps identify the root cause of failures and improves error handling mechanisms (Yuan Kang et al., 2016). This analysis directly contributes to optimizing success and error rates, allowing APIs to perform more reliably under a variety of conditions. As a result, testing becomes more efficient and resilient, as it integrates lifecycle automation, manual insights, and iterative fixes based on real-world errors.

2.3.1. API Vulnerabilities

gRPC is a technology that is currently used for various things, especially APIs and can be combined with various architectures (Collin T Blakley et al., 2024D). The purpose of this research is to determine security vulnerabilities that are widespread according to the components owned by gRPC itself. Quoted from the API testing paper, there are

security vulnerabilities that often exist in a REST API, which will later be used as a **security focus** for testing on the framework to be created (Amid Golmohammadi et al., 2023; Gelei Deng et al., 2023) such as:

- JSON hijacking
- Insecure Direct Object Reference (IDOR)
- Broken Object Level Authorization (BOLA)
- Common Sense Security Policy (CSSP)
- Regex based denial of service (ReDos)
- Nominal test cases (Input validation test)
- Test HTTP methods
- Test Network & Database Connection
- Error Handling
- Directory Traversal
- SQL Injection
- XSS
- Other Injections

With the details of the vulnerabilities above, mapping will be carried out using gRPC according to its capabilities. Analyzing the comparative findings from 2.2.5 above and Table 2.2, our findings suggest that vulnerabilities identified in the REST API can be assessed in gRPC based on its components. For a more in-depth exploration of specific vulnerabilities, refer to the details below:

- HTTP Method

The HTTP Method section, where gRPC does not have an HTTP method like the REST API, this will also underlie the absence of testing or security gaps in HTTP methods in gRPC (Bohdan Marii and Ivan Zholubak, 2022). Therefore, there is no testing for vulnerabilities in the HTTP method component in gRPC.

- Protocol

The protocol used by gRPC only uses HTTP/2, where the benefits of using HTTP/2 are streaming and bidirectional support which can receive multiple requests from client and produce multiple responses (Bohdan Marii and Ivan Zholubak, 2022). Therefore, the vulnerabilities that can be tested on the gRPC protocol are network tests.

- Header

The header in gRPC is different from the header in the REST API, where this header is called metadata or the header used by gRPC (Bohdan Marii and Ivan Zholubak, 2022). Therefore, the vulnerabilities that can be tested on the gRPC header are injection and JSON hijacking.

- Format Communication

The communication format used by gRPC is a protocol buffer, this format is data that is converted into binary and can reduce the size of the message (Bohdan Marii and Ivan Zholubak, 2022). Therefore, the vulnerabilities that can be tested on the gRPC Format Communication are injection, IDOR, BOLA, nominal test cased (input validation test), error handling and directory traversal.

- Supported Browser

Supported browsers in gRPC are gRPC-Web and proxy layer, where browsers generally cannot be used to access the gRPC API before using gRPC Web or proxy layer (Bohdan Marii and Ivan Zholubak, 2022). This component is only used as support for intermediary requests to gRPC.

- Request Method

The request method carried out by gRPC is by carrying out code generation on the client side. This is what differentiates it from the way the REST API works, where only using a collection can be used directly using tools such as Postman or request directly. This component is only used as support for making requests to gRPC.

2.3.2. Black-box

Testing during the black-box stage involves employing techniques in which information about how the application functions is unfamiliar (Omer Bin Tauqueer et al., 2021). In such a scenario, testing is conducted using proto files, and requests are generated without any prior knowledge about the application.

2.3.3. Gray-box

Gray-box testing is conducted with partial knowledge about the application/system (Omer Bin Tauqueer et al., 2021). In this approach, the information utilized for testing in a gray-box scenario includes metadata on gRPC, such as token values, device IDs, and other parameters essential for authentication during the transmission of requests.

2.3.4. CVSS

The Common Vulnerability Scoring System (CVSS) is a tool used to calculate possible assessments of software vulnerabilities (Maciej Nowak et al., 2021). In this research, all vulnerabilities found will be given a score to determine the severity level of the vulnerabilities found. The assessment will be divided into several parts:

- Access Vector (AV)

This matrix reflects the position of the attacker, where when the attacker is further away, the score will be greater. In this assessment, the score is divided

into 4, which is Network (N), Adjacent (A), Local (L) and Physical (P) (Maciej Nowak et al., 2021).

- Access Complexity (AC)

The matrix is related to the difficulty of the exploit carried out by the attacker, the more difficult it is, the higher the assessment will be. The assessment on this matrix is divided into 2, which is Low (L) and High (H) (Maciej Nowak et al., 2021).

- Privileges Required (PR)

This matrix is related to the privileges used by the attacker in carrying out the exploitation. If there are no privileges at all then the attack will be easier. This assessment is divided into 3, which is None (N), Low (L) and High (H) (Maciej Nowak et al., 2021).

- User Interface (UI)

This matrix discusses the user interface when an attack is carried out by an attacker, where the assessment will be carried out by attacks without or using the user interface during exploitation. The assessment in this matrix is divided into 2, which is None (N) and Required (R) (Maciej Nowak et al., 2021).

- Scope (S)

This matrix discusses the time after an attack occurs, where the impact results in changes to the scope or not. This assessment is divided into 2, which is Unchanged (U) and Changed (C) (Maciej Nowak et al., 2021).

- Confidentiality (C)

This matrix discusses the confidentiality of data obtained during exploitation. The greater the confidentiality of the data, the higher the assessment will be.

This assessment is divided into 3, which is None (N), Low (L) and High (H) (Maciej Nowak et al., 2021).

- Integrity (I)

This matrix discusses the integrity of data that has been exploited and can be modified. In terms of assessment, the more data can be modified during exploitation, the higher the assessment will be. This assessment is divided into 3, which is None (N), Low (L) and High (H) (Maciej Nowak et al., 2021).

- Availability (A)

This matrix discusses the availability of resources after exploitation. If the performance or availability of the application is disrupted, the assessment will be higher. This assessment is divided into 3, which is None (N), Low (L) and High (H) (Maciej Nowak et al., 2021).

2.3.5. DevSecOps

DevSecOps represents an evolution of DevOps, integrating development, operations, and security (Zaheeruddin Ahmed and Shoba. C. Francis, 2019). In my current workplace, there is an application security check, primarily focusing on the source code. Unfortunately, this approach has led to overlooking certain cases that were only identified during penetration testing. This research aims to improve the focus of gRPC API security by utilizing DevSecOps in the security lifecycle by utilizing the framework proposed in this research. The security tests developed will be seamlessly integrated into the workflow for automated testing during initial deployment, with the remaining assessments performed manually.

2.3.6. Tools

This study will be supported by various tools for conducting evaluation and support the framework process. These tools prove beneficial, particularly in cases where checks are conducted manually. Therefore, there are several tools available for making requests and intercepting data in gRPC, including:

- Python

Python serves as a programming language suitable for functioning as a client side in initiating requests to gRPC (Anthony Giretti, 2022). During security evaluation, this technology can be employed to automate the process of sending requests to the gRPC server and get the response using generated function, but before that we must convert proto file using python module, namely `protoc` (Protobuf, 2024b). The `'protoc'` module is used to generate functions or stubs from a proto file that can be seen in Figure 2.2. The generated output consists of Python functions that are aligned with the definitions provided in the proto file that can be seen in Figure 2.8. Furthermore, Python will play a crucial role in developing the framework proposed in this thesis.

```
protoc --proto_path=src --python_out=build/gen src/foo.proto src/bar/baz.proto
```

Figure 2.8 Protoc command in python

- gRPCurl

gRPCurl is an application that is similar to Postman, where this application can be used to make gRPC API requests (Anthony Giretti, 2022) that can be seen in Figure 2.9. By using this tool, it can be used to test the features of the API by manipulating data and used for security testing. However, this tool can only handle requests manually by directly adding the payload in the command, making it challenging to use for developing a framework aimed at automating requests. However, this tool only focuses on requests made manually without being able to perform automation security testing.

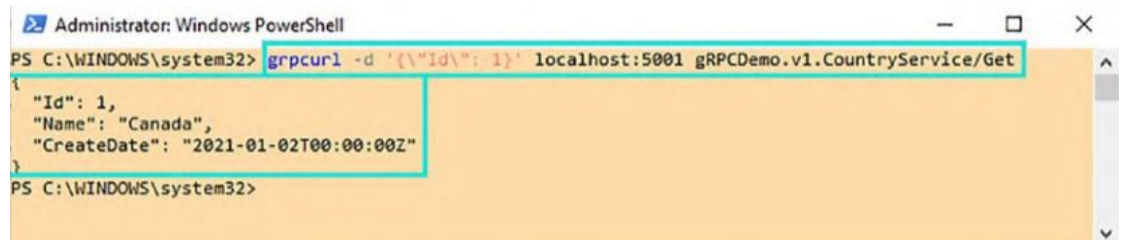


Figure 2.9 gRPCurl create a request (Anthony Giretti, 2022)

- Postman

Postman is a GUI application that can be used to generate HTTP requests to carry out tests for API (Anthony Giretti, 2022). Postman is also a tool that can be used to make requests to gRPC and can be used as a tool to carry out evaluation testing. However, this tool only focuses on requests made manually without being able to perform automation security testing.

- MiTM Proxy

MiTM proxy is an HTTP proxy that can be used to intercept requests and responses (Ayush Goel et al., 2024). Using this tool can help intercept and manipulate requests when executing requests to gRPC. This tool will be used to inspect requests made by the client side to the server or API. However, this tool only focuses on intercepting and changing requests without being able to be used in automation security testing.

- grpc-tester

The gRPC Tester is a tool designed to test gRPC APIs, offering developers a simple and efficient way to verify API functionality through a command-line interface (Shiva Reddy, 2023). However, its limitation lies in using static payloads for each request, making it overly complex to modify for use as a security automation testing tool.

- grpc-pentest-suite

This is a Burp Suite extension designed to intercept packets sent from the client to the API server. It is open-source and accessible online via GitHub for immediate use (Amin Nasiri and volodya, 2023). However, its limitation is the

inability to integrate with Burp Suite's core features, functioning solely as a decoder for proxy intercepts.

- karate-grpc

The Karate framework is a tool for API automation testing, focusing on scenarios related to functionality and performance. While it supports gRPC, manual adjustments are required (thinkerou et al., 2022). However, the framework has limitations in security testing due to the complexity of scenarios and the extensive customizations needed, making it less efficient in this area.

2.4. Related Work and Gap

Several approaches have been proposed for automated API testing, each contributing different insights and tools to address the complexities involved in securing modern APIs.

2.4.1. Previous Study Related Framework

Starting with the Karate DSL framework used as a testing tool designed to simplify API testing with its unique Gherkin-based syntax, making it accessible even to those with little programming experience (Benjamin Bischof et al., 2023). In 2017, Karate initially aimed to improve readability and ease of use in API testing. It leverages Java and JavaScript, allowing users to interact directly with APIs without complex code. Karate supports REST, SOAP, and GraphQL APIs, allowing for easy testing of modern API architectures. It can run data-driven tests, automate parallel testing, and even integrate with performance testing tools like Gatling. With built-in support for a variety of data formats and environments, Karate minimizes setup complexity while providing robust, readable test cases that work seamlessly across a variety of test types.

Next, there is the RestAssured Framework which is used as a popular tool designed to test REST APIs in Java (Jagdeep Jain, 2022). Created to simplify the testing process by BDD-Style (Behavior-Driven Development), this tool allows testers to focus on validating API functionality rather than the complexity of HTTP requests and

responses. RestAssured uses a fluent syntax and a simple structure following the given-when-then pattern, making it intuitive to set up prerequisites, define actions, and check results. By integrating with Java and supporting languages such as Kotlin and Scala, this tool handles complex JSON and XML responses, providing an effective and efficient approach to asserting responses directly in test scripts.

In response to other frameworks that can be used in API security testing. The RJ-Plugin framework, designed for testing RESTful APIs with Testing and Test Control Notation Version 3 (TTCN-3), was created to support testing of web services in microservice architectures by introducing a structured and lightweight testing approach (Theofanis Vassiliou-Gioles, 2020). The framework enables testers to work with APIs at various levels of REST maturity, from basic HTTP communication to sophisticated hypermedia-driven designs. Built on the foundation of TTCN-3, RJ-Plugin supports a variety of API descriptions and data formats, including JSON and OpenAPI, and incorporates essential REST elements such as HTTP verbs, headers, and response codes. With RJ-Plugin, testers can validate API functionality, performance, and compliance with RESTful standards, leveraging TTCN-3's advanced parallel execution and modular testing capabilities to ensure thorough and efficient testing across distributed systems.

Next, API-Driven Testing Frameworks act as powerful tools in validating application behavior directly through APIs, rather than traditional user interfaces, which are constantly changing in modern agile development environments (Asha KR and Shwetha D J, 2015). Imagine a team focused on building a stable and automated testing process: instead of testing through buttons or menus, they interact with APIs specific on REST or SOAP, using pre-defined scripts to send requests and analyze responses to ensure everything works as expected. These frameworks help testers simulate realistic interactions between software components, automate repetitive tasks, and find problems early. Testers define inputs, invoke API calls, and validate outputs with minimal manual intervention, increasing test coverage and early defect detection. This setup saves time, improves product quality, and provides an efficient path to integrating testing into complex, interconnected systems.

More recent work like RESTler and RestTestGen framework, as seen in (Danar Gumilang Putera and Ruki Harwahu, 2024; Myeongsoo Kim and Qi Xin, 2022), provide automated fuzzing solutions for RESTful APIs. RESTler compiles OpenAPI specifications to generate fuzzing scripts, while RestTestGen enhances testing strategies by randomly generating and modifying input parameters. However, these tools, though powerful, remain limited in their scope, focusing only on REST APIs and lacking compatibility with gRPC's distinct architecture. RESTTESTGEN is a framework implemented into an automated black-box testing tool designed for REST APIs, which generates and executes test cases using only API interface definitions, such as Swagger (OpenAPI) files (Emanuele Viglianisi et al., 2020). The tool builds an Operation Dependency Graph to model the relationships between API operations, enabling strategic testing for both standard (nominal) and error scenarios. The tool first tests regular usage patterns through the Nominal Tester and then intentionally introduces errors through the Error Tester to evaluate API error handling and robustness. With a built-in oracle to validate correct responses and flag inconsistencies, RESTTESTGEN has demonstrated its effectiveness on a variety of real-world APIs, uncovering many defects and improving the reliability of API-based systems.

There are other frameworks with the name "API Message-Driven Regression Testing Framework" or MDRT Framework (Emine Dumlu Demircio ğlu and Oya Kalipsiz, 2022), they offer a framework focused on validating test cases against API messages using predefined JSON test scripts that can be seen in Figure 2.10. This approach provides an automated structure for validating API functionalities on REST & SOAP, yet it notably lacks a focus on gRPC APIs, limiting its direct applicability in environments where gRPC is prevalent. Their approach does, however, set a foundation for automating test case creation and validation.

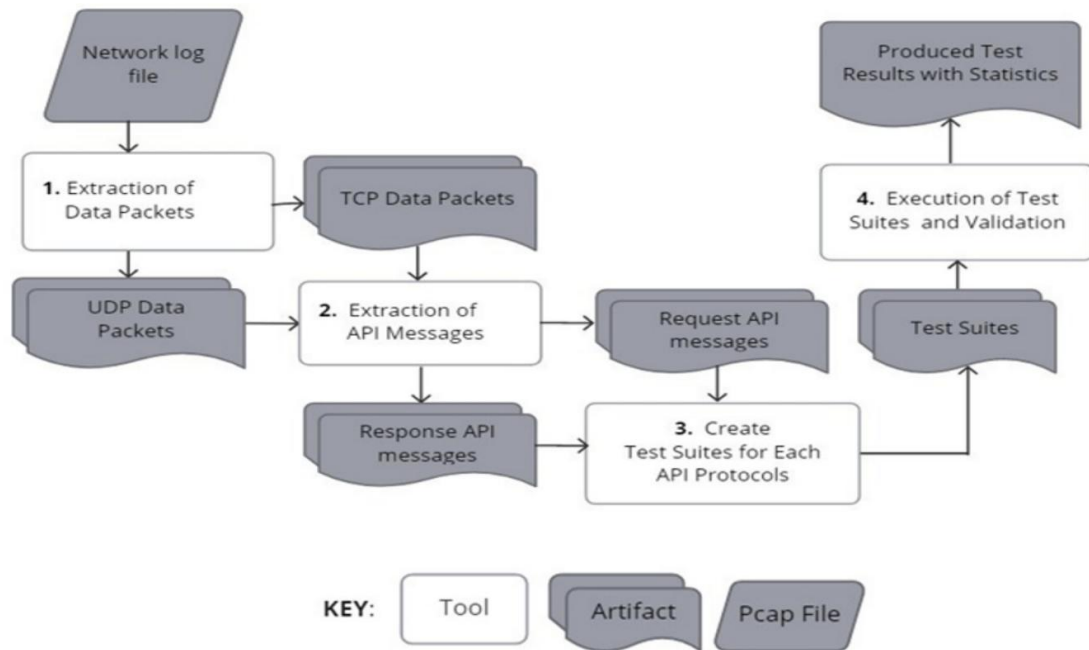


Figure 2.10 API Message-Driven Regression Testing (MDRT) Framework Process (Emine Dumlu Demircio ğlu and Oya Kalipsiz, 2022)

Collectively, these studies highlight a clear trend: while the field of API security testing is advancing rapidly with innovative tools and techniques, most efforts remain centered on RESTful APIs, primarily on existing frameworks. The ongoing development of automated testing frameworks is making headway in easing the complexities of testing modern API systems, but as the industry moves toward gRPC, there is a noticeable gap in tools designed explicitly for this protocol. Several points from previous studies do not address the security testing challenges for gRPC in this research, including:

1. Most studies focus on REST APIs, including the frameworks and tools used.
2. The few studies that discuss gRPC only cover fuzzing, without addressing data manipulation or other aspects.

In creating the framework, a reference framework are needed to conduct security testing on gRPC based on related work, addressing the primary issue of testing efficiency discussed in this thesis. The proposed framework, gRPC Security Testing Framework (GSTF), combines several advances from previous studies tailored to the criteria required by gRPC. MDRT Framework that can be seen in Figure 2.10 (Emine Dumlu

Demircio Şıglu and Oya Kalipsiz, 2022) will serve as the foundational reference for the framework. Based on the criteria in Table 2.3 where the main focus of the GSTF method is "Payload-stub-based test cases" with a security focus in "Vulnerability-specific testing validation" The criteria in the framework reference have been mapped to the process, which has been adjusted to meet the gRPC requirements discussed in section 2.2.5.1 above:

1. **Extraction of payload** used to perform **payload** extraction from file.
2. **Code generation** used for generate **stub** or request and response function in gRPC.
3. **Test case creation** used in performing **test case** generation by combining payload and stub.
4. **Execution** used as an execution to conduct security testing, the results of which will be evaluated, validated and verified.

This study presents a comparison of various approaches based on different studies on API security testing frameworks. The aim is to identify the strengths and weaknesses of each approach, particularly in the developed framework, which has been summarized and is presented in Table 2.3.

Table 2.3 GSTF Compared with Other Approach

Criteria	Karate DSL	RestAssured	RJ-Plugin	API-Driven Testing F	RESTTESTGEN	MDRT	GSTF
Testing Methods	BDD-style & mocking	BDD-style	Modular testing	Request- response scripts validation	Fuzzing & error tests	Regression tests	Payload-stub- based test cases
API Focus	REST, SOAP, GraphQL	REST	REST	REST & SOAP	REST	REST	gRPC

Strength	Gherkin-style (part of BDD-style) and easy to use, multi-API support	Easy for BDD REST testing	Modular functionality	Agile-friendly API testing	High fuzz coverage	Automated regression	Full gRPC coverage, proto adaptability
Limitation	No gRPC Support	No gRPC Support	No gRPC Support	No gRPC Support	No gRPC Support	No gRPC Support	Complex setup
Security Focus	Basic validation	Basic validation	Basic validation only	Functional only	Error fuzzing	Functional validation	Vulnerability-specific testing validation
Tool	Karate	Junit/TestNG	N/A	N/A	N/A	N/A	GSTF Tool

Karate DSL (Benjamin Bischof et al., 2023), RestAssured (Jagdeep Jain, 2022), RJ-Plugin (Theofanis Vassiliou-Gioles, 2020), API-Driven Testing F (Asha KR and Shwetha D J, 2015), RESTTESTGEN (Emanuele Viglianisi et al., 2020), MDRT (Emine Dumlu Demircio ~glu and Oya Kalipsiz, 2022)

Based on related work, comparison and gaps in each approach above, the GSTF Framework is not just an incremental improvement, but a necessary innovation to address the unique challenges of gRPC security testing and it has also answered the question in Q3 and in accordance with Hypothesis 3. By adapting and refining the methodology from previous studies, GSTF provides a comprehensive, efficient and effective solution to ensure the resilience of gRPC-based systems. The tailored approach addresses gaps in existing frameworks, ensuring that gRPC's advanced capabilities are rigorously tested while maintaining efficiency and scalability.

2.4.2. Previous Study Related Method

In a study of various methods as discussed in “RESTful API Testing: A Survey” (Amid Golmohammadi et al., 2023) propose a more extensive set of techniques tailored for RPC-based APIs, particularly focusing on fuzzing as the primary testing method. Their white-box testing approach, using tools like EvoMaster, provides a robust methodology for discovering vulnerabilities in gRPC, REST, and GraphQL APIs with random generated payload. Unlike the previous work, which focuses on validating API messages, Zhang et al. concentrate more on manipulating requests to detect security weaknesses, particularly emphasizing fuzzing. However, their approach is somewhat narrow as it centers primarily on fuzzing, leaving out other key aspects of API security testing, such as performance or protocol-specific challenges.

In the ever-evolving software testing landscape (Myeongsoo Kim and Qi Xin, 2022), RESTest has emerged as a formidable ally for developers looking to ensure the reliability of their RESTful APIs. This innovative tool harnesses the power of model-based testing, using the OpenAPI specification as its roadmap, detailing every API endpoint and parameter. What sets RESTest apart is its keen ability to understand and manage dependencies between parameters, recognizing that the validity of one input often depends on the existence of another. By transforming these dependencies into actionable constraints, RESTest have a **method** that can intelligently generate a diverse test payload through a combination of constraint resolution techniques and random input generation. As it runs its tests, RESTest carefully monitors the API's responses, verifying that they align with expected outcomes and adhere to defined schemas. Through this rigorous and systematic approach, RESTest not only uncovers hidden bugs but also strengthens the resilience of APIs, paving the way for a more reliable digital ecosystem.

Addressing these wider concerns, “Automated API Testing” (Isha et al., 2018) propose a comprehensive automated API testing tool that goes beyond fuzzing to handle challenges like unpredictable JSON responses and test-case sequencing. While their approach automates various parts of the API testing lifecycle, such as comparing pass/fail results, it similarly focuses more on REST APIs rather than gRPC,

highlighting a recurring gap in the literature. This leaves room for more specialized frameworks addressing gRPC's unique architecture. Furthermore, the combination of response validation can be combined with a study from the DynER framework that offers a new approach to validating API responses with a focus on improving API success and error rates (Juxing Chen et al., 2024). With this combination, data related to vulnerabilities will be obtained from responses that pass or error with the category of pass results according to the desired response. Although response validation by comparing actual responses to expected results can still produce false positives from security tools, a manual verification method is needed to confirm the accuracy of the scan results.

According to a study, a manual verification method called Proof-Based Scanning (Jayshish M. Popat et al., 2021) to identifying and verifying security vulnerabilities in web applications. This approach not only detects vulnerabilities but also automatically exploits them in a safe, read-only manner to generate concrete proof of their existence. The scanning framework uses Python-based automation tools such as Metasploit (pymetasploit3) and web-scraping techniques (BeautifulSoup) to identify weak points like SQL injections, command injections, and cross-site scripting (XSS). Once a vulnerability is detected, an exploit is executed, and proof—such as database names or system information—is retrieved to confirm its legitimacy. This eliminates **false positives**, reducing the need for manual verification by security professionals. However, while proof-based scanning improves accuracy and efficiency, its reliance on automated exploitation raises concerns regarding ethical use and potential system disruption. Moreover, security professionals must still possess deep vulnerability analysis skills to interpret results and prioritize mitigation effectively. This method represents a shift towards more proactive security testing but requires careful handling to balance automation with expert oversight.

In contrast, (Hamza Ed-douibi et al., 2018) present an approach reliant on OpenAPI specifications to generate TestSuite models, which then feed into the security testing process. This work contributes significantly to REST API testing by leveraging automation through well-defined specifications but, once again, leaves out

considerations specific to gRPC, which doesn't always follow OpenAPI standards. Nevertheless, their method serves as a precursor to future work in testing specification-driven models for gRPC.

To tackle gRPC's security concerns specifically, EvoMaster, that can be seen in (Man Zhang et al., 2023), stands out. It leverages evolutionary algorithms and white-box techniques to test not only REST but also gRPC APIs, providing a broader range of automated testing capabilities that cover modern API environments. The use of AI-driven fuzzing offers a significant step forward in testing RPC-based APIs.

Lastly, the study from the book "Practical Security Automation and Testing" (Tony Hsiang-Chih Hsu, 2019) introduces a method that can potentially be applied to this research, even though it doesn't explicitly state that the method is for gRPC. This method involves a testing data generator, where the test data is created based on the data type of the payload being used.

Collectively, these studies only a few approaches, such as those involving EvoMaster, extend to gRPC, indicating the need for more focused work on gRPC-specific vulnerabilities and testing methods. However, based on the essential criteria for gRPC outlined in section 2.2.5.1 above, several methods are integrated into an approach called **"Data Type-Based Generation of Payload in a Model Test Case & Response Validation"** Within the framework, this is implemented as the **"Payload-Stub-Based Test Cases"** method that can be seen in Table 2.3, which is derived from various sources, including:

1. A proposed **method** for testing is based on generating test cases. The test case referred to at this stage is in the form of vulnerability test data which will later be carried out to adjust the stub from gRPC (Hamza Ed-douibi et al., 2018). This method is very useful because the test cases created can be used in every object of the gRPC proto file.
2. A proposed **method** using a data generator, where payloads are customized according to their data types to create test cases (Tony Hsiang-Chih Hsu, 2019).

This method is very important because gRPC has a specific data type for each parameter.

3. A proposed **method** for evaluate success and error rates and this method will be used for **the evaluation phase** (Juxing Chen et al., 2024). This method will be very important to measure the rate of requests made.
4. A proposed **method** that compares response outcomes between passing and failing test cases and this method will be used for **the validation phase** (Isha et al., 2018). This method is important in validating responses, especially to determine whether the response is vulnerable by comparing it with the expected result or a similar pattern that is vulnerable to each vulnerability.
5. A proposed method for manual verification is implemented to measure false positives that have been previously validated and this method will be used for **the verification phase** (Jayshish M. Popat et al., 2021). This method is very important to re-verify the validation results carried out by GSTF manually to reduce false positives from the scan results.

However, at this stage a comparison is made between the methods that have been combined previously based on the criteria and other methods that have been discussed previously, which will later be used in the framework proposed in this study, as illustrated in Table 2.4. The "**Data type-based generation payload in a model test case & response validation**" method offers advantages beyond efficiency—it classifies payloads based on data types, making it a more suitable approach compared to other methods that rely on random payload generation.

Table 2.4 Data type-based generation payload in a model test case & response validation compared with another method

Method	Testing Approach	Complexity	Strength	Limitation
Data type-based generation payload in a model test case & response validation	Grey-box	$O(n \times t)$ n = number of data type t = number of test cases	More efficient for payloads that have already classified their data type; response validation	Limited to basic data types; may miss complex cases.
Model based generated payload (RestTest)	Black-box	$O(n \times m)$ n = number of input parameters r = random payload	Can generate payload with randomly with constraint resolution techniques; status and response validation	All parameters will be tested which affect the testing time
Randomly Generated payload (RestTestGen)	Black-box	$O(n \times r)$ n = number of input parameters r = random payload	Can generate payload with randomly; status and response validation	Not security test focus and all parameters will be tested which affect the testing time

Data type-based generation payload in a model test case & response validation (Hamza Ed-douibi et al., 2018; Isha et al., 2018; Tony Hsiang-Chih Hsu, 2019), Model based generated payload (RestTest) (Myeongsoo Kim and Qi Xin, 2022), Randomly Generated payload (RestTestGen) (Myeongsoo Kim and Qi Xin, 2022)

2.4.3. Proposed Framework

By integrating the GSTF framework with gRPC specific criteria, the process of combining GSTF with structured methods to be applied at each stage of the framework. Essentially, this framework consists of four main processes: **Extraction of Payload, Code Generation, Test Case Creation, and Execution**, as outlined in the framework structure based on section 2.4.3.

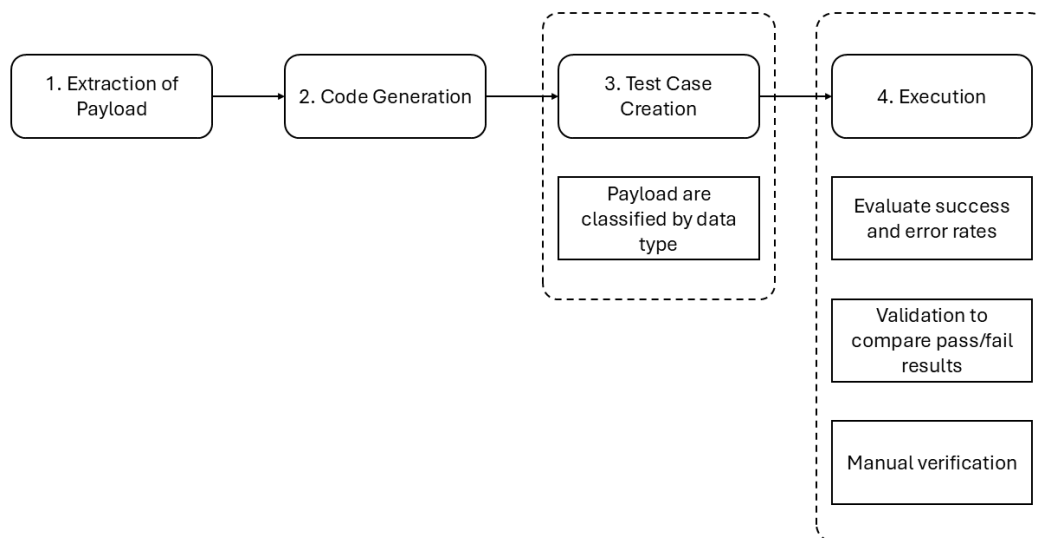


Figure 2.11 GSTF proposed framework

Based on the process in Figure 2.11, a combination of the framework and methods is applied. This combination aligns each method's function with the corresponding process stages in the framework. The detailed process is as follows:

1. Extraction of Payload

In Phase 1, payloads are extracted from data sources, such as payload files, based on the data type used in the testing. The extracted payloads will be utilized in the next stage to construct test cases that comply with the gRPC protocol (Emine Dumlu Demircio Ÿglu and Oya Kalipsiz, 2022).

2. Code Generation

In Phase 2, this step is responsible for generating stubs from the available proto files. These stubs are essential for sending requests to the target gRPC API in

the correct format. At this stage, the framework ensures that the generated code is compatible with the previously extracted payloads (Anthony Giretti, 2022).

3. Test Case Creation

In Phase 3, test cases are created by combining the outputs from Phase 1 and Phase 2 through the following sub-method:

- a. Extracted payloads are classified based on their data type to systematically structure the test cases (Tony Hsiang-Chih Hsu, 2019). The classification method based on data type is placed in Phase 3 because it aligns with combining the outputs from Phase 1, which consists of payloads categorized by data type, and Phase 2, which includes stubs containing object functions with specific data types. Therefore, combination payloads and stubs based on data type fits naturally within this process.

4. Execution

In Phase 4, test scenario execution in gRPC is performed. This phase consists of several sub-method, including:

- a. Evaluating the success rate of the test, including error rates and the effectiveness of payload exploitation against the tested API (Juxing Chen et al., 2024). This method is essential in Phase 4 to ensure that evaluation is conducted after the request is executed, allowing for data analysis afterward.
- b. Validating test results by comparing them with expected outcomes, identifying whether they fall into false positive or false negative cases (Isha et al., 2018). This method is crucial in Phase 4, as validation is performed using data after the request execution. The response data can then be validated or compared with the expected result.

- c. Conducting manual verification to ensure the accuracy of test results, particularly in detecting false positives in security reports (Jayshish M. Popat et al., 2021). This method is essential in Phase 4 after validation is completed, as the output from GSTF's validation needs to be verified to ensure there are no false positives.

Through this process, by combining the framework and the method, the limitations of other methods can be addressed, especially some methods that are done manually are already covered by these methods except manual verification, therefore, it can answer the question in Q2 and in accordance with Hypothesis 2.

