UNIVERSITÀ DEGLI STUDI DI VERONA

Department of Engineering for Innovation Medicine

Master's Degree in

Computer Engineering for Robotics and Smart Industry

Automated API Specification Inference from Network Traffic of Android Applications

Supervisor:
Prof. Mariano Ceccato

Candidate:
S. M. Reza Ahmadi
Matricola: VR510067

Co-supervisor:
Prof. Michele Pasqua

Collaborators:
Dr. Davide Corradini
Sofia Mari

Academic Year 2025/2026

Acknowledgments

Special thanks to Professor Mariano Ceccato, whose unwavering support and dedication made this research possible.

Contents

Chapter 1

Introduction

In the software development concept, the security [1] has multiple aspects. Some aspects are focused on the client side, such as released files (e.g., distributable packages such as APK, EXE, or DEB), and some others, after the installation part (e.g., reverse engineering and extracting non-obfuscated [2] source code). Another aspect is the security of the server-side (e.g., Distributed Denial-of-Service (DDoS) [3] attacks). But the third aspect of software security is API [4] security, because it is the communication part of client and server, and with any bottleneck, the security of both client and server will fall. This kind of security is even more important than the other two aspects due to the leaking of sensitive data, and sometimes, compensation for damages is almost impossible. One of the most critical approaches that attackers (or analyzers) are using to organize this kind of attack is called Man-in-the-Middle (MITM) [5]. In fact, MITM aims to observe transferred data between the client and server.

In this study, we discuss how to use and integrate existing tools to develop a system for MITM testing of Android [6] mobile applications. This study is not limited to implementing a simple script but takes a step further by developing an automated system to MITM-test and analyze a large dataset [7] of Android applications.

Running tests on a large-scale dataset of Android applications is time-consuming and resource-intensive, and for this reason, the implemented system is designed to be parameterized and controllable with respect to resource usage, and to run parallel test experiments to reduce testing time.

The main goal of this study is to analyze the network traffic of Android applications. To this end, a pre-programmed process is needed to eliminate unnecessary data and make the data noise-free [8], so that the gathered information can be effectively studied.

At the end of this research, an evaluation system assesses how closely the estimated results match the actual results and how well the implemented system discovers vital information.

1.1 Motivation

Analysis of Android mobile applications is vital from many aspects. By analyzing the extracted information, it is discoverable how an application interacts with the server, and what kind of data is being sent and received between the client and the server. It is possible to have many metrics to check the network usage of an application, and furthermore, to discover many vulnerabilities and security issues. For example, by analyzing the network traffic of an application, it is possible to discover if the application is sending sensitive information in clear text, or if it is using secure communication protocols, or if the service or application may contain downloadable purchase content without any safety measures [9]. Additionally, by analyzing the network traffic, it is possible to discover if the application is communicating with any malicious [10] servers or if it is sending data to third-party services without the user's consent. Overall, analyzing the network traffic of Android applications can provide valuable insights into the application's behavior and can help identify potential security risks and vulnerabilities. For instance, it is possible to discover an undocumented API, which is highly important for security analysis to prevent data leakage [11]. Protecting user data is highly important for every service and company. Nowadays, there are many conventions and regulatory organizations that ask to protect data users, and they control this procedure, for instance, GDPR [12]. Furthermore, leaking the news about the data leak of each service is highly impactful for User Trust, and each service and company should control the sensitive data and information of their users, and studying APIs by the MITM approach is a robust way to discover Zero-Day vulnerability [13] threats.

As a target operating system [14], Android is selected because of its popularity and wide usage. Android is the most widely used mobile operating system in the world [15], with a large user base and a vast ecosystem of applications.

1.2 Problem Statement

Manual MITM test and analysis of an Android mobile application is a time-consuming and error-prone process. To organize an MITM test, many tools

should be integrated together, and an operator should observe and handle all of the steps from testing part of an Android application and manipulation up to classifying results and output files. Considering this long and exhaustive process for just one application, testing large datasets is almost impossible. By default, all the tools are separate islands, and the lack of some automated system [16] makes the task even more impossible.

By automation processes, it is possible to run many tests in parallel. Furthermore, the parallel running is not the only available concern; another important aspect is managing and allocating resources by the system. Parallel running of test experiments reduces the whole duration of the test for a dataset significantly. It is totally visible that by manual testing, an operator can test a single application every time, but by the implemented automation system, there is a possibility to use all the capacity of the host machine resources.

Also, it is not just about running the tests; there are some post-processing steps that are really important for understanding the results. In most cases, after the capturing process of intercepted network API data, the results are noisy, and a manual process to make data noise-free is almost impossible, considering that in each experiment, the input is a dataset of applications and not a single application.

Another characteristic of an automation system is the repeatability [17] of the experiments. Because of the implemented architecture, there is a possibility to repeat every step of the pipeline, and it is drastically helpful for improving the filtering data steps.

1.3 Contributions

Within this study, several critical aspects of software security testing and implementation are introduced. The main contributions of this thesis are summarized as follows:

- **Automated Testing Pipeline:** Design and development of a fully automated Man-in-the-Middle (MITM) testing pipeline capable of processing large datasets of Android applications without manual intervention.
- **Scalable Architecture:** Implementation of a highly scalable system architecture that leverages dynamic resource allocation and parallel execution to reduce the overall testing time significantly.

- **Advanced Data Filtering:** Development of a multi-stage noise reduction mechanism that strictly filters raw captured network traffic, successfully transforming raw data into actionable information.
- **Experimental Repeatability:** Engineering a deterministic system infrastructure that guarantees the strict repeatability of the experiments at every stage of the pipeline.
- **Ground Truth [18] Filtering:** Introduction of a scoring system and approach to filter mobile-related data from ground truth to reduce the noise comparison.
- **Standardized Data Conversion:** Introduction of a systematic approach to convert the filtered information into standard formats (e.g., OpenAPI Specifications [19]) to match the exact structure of the target Ground Truth data.
- **Evaluation and Benchmarking:** Establishment of a robust evaluation framework to compare the inferred results against the Ground Truth, alongside a comprehensive stability comparison of the related conversion and evaluation tools.
- **Distinguishing results:** Introducing multiple types of differences between results and Ground Truth, such as missed, undocumented, and same-founding APIs.

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

- **Chapter 2:** Reviews theoretical background, Android security, and network interception concepts.
- **Chapter 3:** Discusses existing state-of-the-art tools and identifies current research gaps.
- **Chapter 4:** Details the system architecture, automation pipeline, and resource management.
- **Chapter 5:** Covers data processing, noise filtering, and automated specification generation.

- **Chapter 6:** Introduces conversion tools, comparison frameworks, and the evaluation metrics.
- **Chapter 7:** Presents dataset selection, ground truth definition.
- **Chapter 8:** Presents experimental results and evaluation.
- **Chapter 9:** Summarizes thesis contributions, core findings, and future research directions.

Chapter 2

Background

This chapter introduces the fundamental concepts of API analysis and network interception. Understanding these principles is essential before examining the architecture of the proposed project.

First, the dynamic analysis of Android applications is discussed. Then, the concepts of Man-in-the-Middle (MITM) network interception are explained. Finally, the chapter concludes by exploring the standard methods for API Specification Inference.

2.1 Dynamic Analysis of Android Applications

Exploring any Android application has many aspects and ways. To understand how an Android application works and how it communicates with its environment and operating system, there are 2 perspectives: static and dynamic. static approach [20] points to investigate within the installation file (APK [21]) and, dynamic analysis [22] (running an Android application and observing its behavior).

All Android applications have a similar, well-defined architecture for file types (the role of each file type) and share the same directory hierarchy [23]. Thanks to this standard, it is possible to examine the written code of all applications using a single approach, but there are some bottlenecks and limitations that make this approach impractical for most of the typical and standard applications. In the case of this study and the API analysis, many Android applications generate API URLs [24] dynamically and at runtime, and there is no final shape of the request and REST API [25] (consisting of URL, header, parameter, etc.) within the source code; furthermore, almost most of the applications in the release phase and build APK, use obfuscation

technique [2] to improve security and reduce size of application by some built-in tools of Android ecosystem like ProGuard [26] and R8 [27]. Some tools help to check the source code within an APK file to determine whether it is obfuscated. The most appropriate tool is APK analyzer [28], a built-in feature of Android Studio [29].

Due to limitations in API static analysis, dynamic analysis is more effective, but it requires an isolated system and an Android device or emulator [30]. The dynamic approach requires running the Android application on the victim machine [31] and allowing it to exhibit its behavior. The most critical requirement for dynamic analysis is an operator to interact with the application, as it can cause the application to run and harm the victim's machine.

Due to the analysis on a large-scale dataset of Android application human interaction, dynamic analysis in this project is totally impossible. There are some developed tools to interact with an Android application without human interaction needed, in fact, they are able to be programmed to do a specific task by the command prompt and scripting for instance, 5 random taps (click) on 5 random areas of screen or 1 min random tap on the random area of screen with 2 seconds gap between each tap. One of the most well-known tools for this purpose is Monkey [32], a built-in tool in the Android Software Development Kit (SDK) [33] which is used in this project.

2.2 Network Traffic Interception and MITM

In this generation of the internet, for the secure exchange of users' data, everything is encrypted using protocols such as HTTPS [34] and TLS [35]. In fact, HTTPS enables encrypted communication over HTTP [36] via a secure TLS connection, and TLS operates in the background, conducting a "handshake" to establish secure encryption. Without TLS, HTTP is unencrypted, allowing security analysts to view or steal information. By using HTTPS (which requires a TLS/SSL certificate [37]), data, such as usernames, passwords, and credit card numbers, is rendered unreadable to anyone except the sender and receiver. [38].

Nowadays, security analysts use MITM attacks to circumvent this security mechanism. Technically, they use their own SSL certificate on the victim device to observe the data exchanged between the client and the server.

The Mitmproxy [39] tool provides a comprehensive suite of features for implementing this interception architecture. It dynamically generates and signs SSL certificates [40] on the fly, enabling the proxy system to terminate the TLS

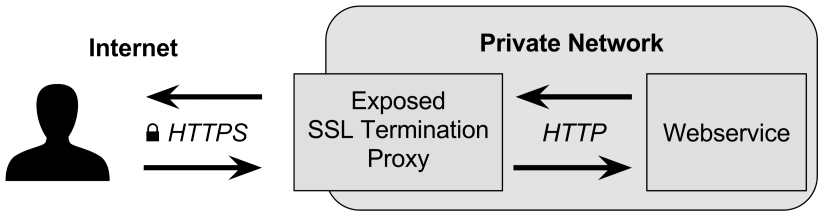


Figure 2.1: The architecture of an SSL/TLS termination proxy.

connection seamlessly. Consequently, any data transferred over this intercepted connection becomes fully observable and loggable in plaintext.

Figure 2.2: The simple architecture of a Man-in-the-Middle (MITM) interception using mitmproxy.

The procedure for network interception on an Android victim emulator is the same, and the SSL certificate for mitmproxy should be pushed to the emulator, since manipulating system-level storage requires granting Root access [41].

2.3 API Specification Inference Methods

The most important part of a data comparison is the data structure. In the previous section, the mechanism of MITM network interception was explained. It is necessary to save the results after each capturing operation. There is a file format used by Mitmproxy, called the HTTP Archive (HAR) format [42], which is a JSON-formatted [43] archive for logging a web browser or application's interaction with a site. The common extension for these files is **.har**.

The format is used, among other tools, by the HTTP Archive [44] project to collect and analyze web performance data.

Each HAR file contains all interactions between the server and the client; that is, all the called REST APIs include the URL, request headers, HTTP method, body, time, etc.

Listing 2.1: JSON schema representing the core structure of a HAR file.

```
{
  "log": {
    "entries": [
      {
        "request": {
          "method": "<String: HTTP method (e.g., GET,
             POST)>",
          "url": "<String: Absolute endpoint URL>",
          "headers": [ "..." ],
          "queryString": [ "..." ],
          "postData": {
            "text": "<String: Request payload>"
          }
        },
        "response": {
          "status": "<Number: HTTP status code>",
          "content": {
            "text": "<String: Response payload>"
          }
        }
      },
      "..."
    ]
  }
}
```

The raw HAR files captured are not suitable for study or for extracting understandable information, because they contain repeated APIs (the same URL and parameter), non-vital APIs, and eventually noise data that should be filtered and processed before any use.

With clean, structured information, it is possible to use a Black-box approach [45] to discover (or infer) the exact way to use the API then, as an Inference Process, it is possible to convert raw information into critical knowledge for reuse.

For instance, generate the final pattern for each API endpoint [46] using clustering [47]. For example by finding these two same endpoints `api.com/user/12` and `api.com/user/85` this represented pattern will be discovered: `api.com/user/{id}`

At the end, the purpose is to convert HAR files to a human-readable format like the OpenAPI Specification (OAS), a standardized, structured format known to Swagger [48]. This clean structure is designed for reuse and comparison. OAS is stored by **.json** [49] or **yaml** [50] and with JSON-Object structure.

OAS defines a standard, programming language-agnostic interface description for HTTP APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic [51].

When properly defined via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interface descriptions have done for lower-level programming, OAS removes guesswork in calling a service.

An OpenAPI document that conforms to the OAS is itself a JSON object, which may be represented either in JSON or YAML format. Examples in this specification will be shown in YAML for brevity.

All field names in the specification are case-sensitive. This includes all fields that are used as keys in a map, except where explicitly noted that keys are case-insensitive.

OAS Objects expose two types of fields: fixed fields, which have a declared name, and patterned fields, which have a declared pattern for the field name. Patterned fields MUST have unique names within the containing object.

OAS is versioned using a major.minor.patch versioning scheme. The major.minor portion of the version string (for example 3.1) SHALL designate the OAS feature set. .patch versions address errors in, or provide clarifications to, this document, not the feature set. Tooling which supports OAS 3.1 SHOULD be compatible with all OAS 3.1.* versions. The patch version SHOULD NOT be considered by tooling, making no distinction between 3.1.0 and 3.1.1 for example.

Certain fields or features may be marked Deprecated. These fields and features remain part of the specification and can be used like any other field or feature. However, OpenAPI Description authors should use newer fields and features documented to replace the deprecated ones whenever possible.

At this time, such elements are expected to remain part of the OAS until the next major version, although a future minor version of this specification may define a policy for later removal of deprecated elements.

Occasionally, non-backwards compatible changes may be made in minor versions of the OAS where impact is believed to be low relative to the benefit

provided.

Listing 2.2: JSON schema representing the core structure of an OAS file.

```
{
  "openapi": "3.0.3",
  "info": { "title": "<String: Extracted API Name>" },
  "paths": {
    "/<String: Discovered Endpoint>": {
      "<String: HTTP Method>": {
        "parameters": [
          { "name": "<String>", "in": "<String>", "schema
              ": { "type": "<Type>" } }
        ],
        "requestBody": {
          "content": { "<String: MIME type>": { "schema":
              "<Object: Payload>" } }
        },
        "responses": {
          "<String: Status Code>": {
            "content": { "<String: MIME type>": { "schema
                ": "<Object: Response>" } }
          }
        }
      }
    }
  }
}
```

Chapter 3

Related Work

This chapter focuses on the available tools for network API security analysis and the bottlenecks of each tool for having an automated Android MITM test system. To this end, the chapter is organized as follows: Section 3.1 reviews the state-of-the-art tools and existing solutions, categorizing them into network traffic capture utilities and API specification inference frameworks. Following this review, Section 3.2 summarizes the identified research gaps and limitations that directly motivate the proposed automated architecture presented in this thesis.

3.1 State of the Art: Existing Tools

The MITM test on Android requires several tools to work. In fact, there are no tools available to run an end-to-end, fully automated test of this approach, but some tools can technically help with part of it. This section introduces existing tools for capturing device traffic regardless of the victim's operating system, as well as Specification Inference tools. Furthermore, it is discussed why some tools are not useful for Android API security analysis and why an Automated Pipeline Test System (APTS) is required to use practical tools.

3.1.1 Traffic Capture Tools

The Idea of analyzing and intercepting a network can be implemented using several approaches and tool types. Essentially, the main goal is to gather information from two computer devices; in the case of an API Specification, the two devices are a client and a server. Since the emergence of cybersecurity, many approaches have been developed to gather leaked information exchanged

within computer networks.

One of the most important series of tools is Packet Sniffers or Packet analyzers [52], which are designed to intercept network traffic at the TCP/IP [53] layer. Typically, this kind of tool captures raw packets without decrypting their contents; this approach is a passive way to observe data passing through a network. Mostly, packet sniffers are used to analyze networks without private keys, which is almost impossible; instead, they remain completely invisible and undetectable within a network, and they do not require manipulating clients or setting up a proxy. Some robust tools based on this passive approach are Wireshark [54] and tcpdump [55].

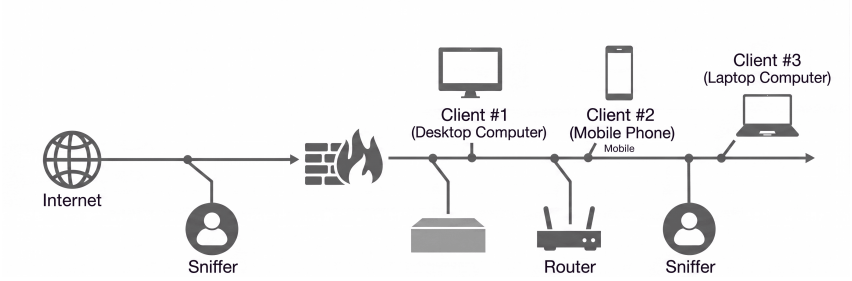


Figure 3.1: The Architecture diagram illustrating how a packet sniffer operates on a network.

On the other hand, MITM tools work on the same object but use a different approach. Actually, Man-In-The-Middle tools rely on installing a Root CA [56] and setting up a proxy on the victim client system, which gives them robust capabilities to decrypt traffic; meanwhile, these tools and approaches involve active interception, but they are easy to detect by Firewalls [57] and by certificate errors. The most famous tools that are created based on the MITM approach are Fiddler [58], Charles Proxy [59], Mitmproxy [39].

Since the automated environment in this project provides full control over the client device (allowing for proxy setup and certificate installation), MITM tools are the most practical choice for data analysis. They are particularly effective for reliably generating structured HAR files for target applications. Among the available options, Mitmproxy is selected as the core interception component due to its command-line interface, extensive Python API, and high programmability, which are essential for seamless integration into a fully APTS .

Feature	Packet Analyzers (Sniffers)	MITM Proxy Tools
Intervention Type	Passive (Observation and monitoring)	Active (Interception and intervention)
Data Modification	None (Reads raw data; traffic passes unaltered)	Yes (Can stop, edit, drop, or inject data on-the-fly)
HTTPS/TLS Decryption	Impossible (Unless the server's private key is pre-shared)	Possible (Relies on generating fake certificates and installing a custom Root CA)
Network Configuration	Not required (Captures traffic directly from the network interface)	Required (Client traffic must be explicitly routed through the proxy)
Detectability	Extremely low to invisible	Higher (Easily detected by firewalls, certificate pinning [60], or SSL errors)
Use Cases	Network troubleshooting, performance analysis (Network Admin)	Penetration testing, bug bounty, application hacking (Security Researcher)

Table 3.1: A Comparison between Packet Analyzers and Man-In-The-Middle (MITM) Tools.

3.1.2 Specification Inference Frameworks

Captured data (HTTP Archive (HAR)) is not suitable for a case study on its own. It contains repeated endpoints, URLs, and unwanted data. HAR is a recording of actual network traffic, for instance, the specific requests, responses, headers, and payloads sent during a session. OAS, on the other hand, is the contract or blueprint that defines how the API is supposed to work structurally. To perform this important procedure, many specialized tools have been developed to convert a HAR to an OAS.

Mitmproxy2swagger [61] is one of the most famous tools for converting Mitmproxy captures to OpenAPI 3.0 [51] specifications. This Python-based tool focuses on converting the exports of mitmproxy, which has significant compatibility with that tool. Supporting the path parameter regex allows for defining a custom regular expression to determine which segments of a URL path should be treated as dynamic variables rather than static paths. Furthermore, there are 2 flags that instruct the tool to extract the actual request/response payloads and HTTP headers from the captured traffic and embed them into the generated schema.

Another robust tool is har-to-openapi [62], which is a tool based on har2openapi [63]. This JavaScript-based tool supports both library and CLI usage [64]. It supports Automatic Path Parameterization, which attempts to identify and extract dynamic variables from URLs, as well as other features such as Domain Filtering & Multi-Spec Generation, Parameter & Header Inference, Output Pruning and Cleanup, etc., making it much more robust than other tools.

Another useful tool is RestTestGen [65], a framework for automated black-box testing of RESTful APIs. This Java-based program, developed by the University of Verona, is a tool and framework for automated black-box testing of RESTful APIs. It features multiple built-in testing strategies, including nominal and error testing, mass-assignment security checks, and deep reinforcement learning [66]. The tool intelligently maps out API workflows using an Operation Dependency Graph (ODG) to automatically handle complex data passing and dynamic authentication. Upon completion, it outputs comprehensive HTML reports alongside replayable JUnit and REST-assured test cases to help developers easily reproduce bugs. This tool was introduced in ICST 2020 [67] [68].

Furthermore, there are other tools for converting the HTTP Archive to OAS, but this thesis uses the tools mentioned above.

3.2 Summary of Research Gaps

Despite the availability of sufficient approaches and tools to capture network traffic and convert it to OAS, an automated test pipeline is lacking, especially in the Android environment.

All these frameworks require human interaction for the conversion process. When dealing with large-scale datasets, it is almost impossible to handle them manually. Furthermore, the raw captured data is not directly suitable for conversion to OAS, and some post-processing is required to make it noise-free and aligned with the base APIs. The gap in the Android ecosystem is even bigger due to the lack of an automated system designed to capture Android application traffic.

The core of this thesis focuses on the orchestration and organization of tools, approaches, and scripts from dataset inputs to the automated test pipeline, capturing data, and performing post-processing on the captured data to extract vital information from the large, unclear raw data.

Chapter 4

System Architecture and Methodology

The main pipeline of the APTS is introduced in this chapter. How to automate the entire data-capture process is the main topic of this chapter. First, the APTS is explained; then, the characteristics of the host machine and the outer layer of containers are discussed, along with how the entire test system is controlled by the host machine and balanced with available resources. Finally, to show how an Android application is tested within a container, the Intra-Container Lifecycle is discussed. For any automation system, a time machine is needed to check past events in the future for tracking the system's state and step-by-step events. In the last section of this chapter, the events log system is described to enhance the debugging of the pipeline system. All the required concepts to understand how the APTS works will be covered by the end of this chapter. APTS's output is raw data, ready for post-processing.

4.1 Overview of the Automated Pipeline Test System

The main objective of the APTS is to take an input APK dataset and run a series of actions in iterations for each APK in the dataset. To ensure easy flow control and parallel testing, it is designed to run tests within containerized environments [69]. This architecture is chosen for its ease of implementation and for safe, repeatable operation.

In fact, in the pipeline system, there is a dedicated test container for each Android application, and a programmed actor is required to control the

system's flow and run the containers according to specific rules and conditions. To achieve maximum compatibility and flexibility, scripting is the best approach for working as an actor within this architecture. Because of its compatibility with the host machine, Linux-based operating system, and containerization libraries, Python [70] is chosen.

The main script will run on the host machine and control the entire APTS . It is not just limited to running containers; furthermore, there are more demanded actions that need to be done by the actor script, such as:

- Getting the controlling parameters of the system from the input
- Building the container image for testing
- Load Input APKs from dataset
- Run containers in parallel
- Handling a demanded delay between running the container procedure
- Check availability of the resources before running a new container, and apply the demanded delay after any resource checking
- defining dedicated names and IDs for each container
- Passing required information to each container
- Storing the log of the entire system
- Aggregating the generated network data (HAR files) and performing cleanup operations after testing.
- Store the final results in a specific directory for post-processing and evaluation, and separate them by the package name or package ID [71] of the application for better organization.

Ultimately, the main script is the brain of the entire system, controlling the pipeline flow, managing resources, and ensuring all components work together seamlessly. Testing will be done within the container.

This strict isolation ensures that captured network traffic is generated solely by the target application, significantly reducing system-level noise.

Consequently, the resulting dataset provides a highly reliable foundation for the subsequent API specification inference phases.

4.2 Environment Configuration

In this section, the configuration of the host machine and the containerized environment is discussed. The host machine runs the main script, which controls the entire automated test pipeline. The most characteristic property of the host computer is having sufficient resources to run the experiment, since the procedure of any single Android application runs within a container, and the host must allocate sufficient resources to it. Under this hypothesis, it is understandable that, for running tests of single Applications in parallel, the host must be equipped with more powerful hardware resources. In general, the heaviest part of each container is the emulator [72] used to run the application.

Fundamentally, the host is programmed to build the base image for containerization and, within an iterative process, run an equal number of containers based on that image, dedicating one to each application derived from the dataset. For this procedure, some tools are needed; for instance, Python is chosen as the scripting language, and it is necessary to install and use some libraries within the automated system. Firstly, the most important requirement for the library is an SDK for containerized programs, because it is more practical, safe, and trustworthy to communicate with containerized programs via the SDK rather than running commands in a Python script. Secondly, other libraries are used as helpers or utilities, such as `psutil` [73] and Androguard [74].

Python System And Process Utilities, or in short, `psutil`, is a cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors) in Python. It is mainly useful for system monitoring, profiling, limiting process resources, and managing running processes. It is used within this project to check the free hardware resources before running any container and allocating resources to it.

Androguard is a Python-based framework designed to facilitate the thorough examination of Android application files. Its primary focus lies in the structural dissection of APK (Android Package) files, the standard packaging format for Android apps. This library is used to fetch important information from each APK before running the container, such as the package name, package ID, or application ID [71].

In conclusion, the environment should be prepared to run test containers with the necessary software and tools and sufficient hardware resources. More resources make it possible to run more parallel test containers, and more parallel containers shorten the overall testing duration.

4.2.1 Containerization with Docker and Podman

In this section, the containerization tools used in this project are explained in detail.

Docker [75] is among the most well-known containerization tools. It was developed to handle large-scale processes and offers many options and utilities to control the flow of the demanded automated test pipeline. The installation process on the host machine is not complicated. Docker Hub [76] offers many default images built from official and unofficial open-source projects. Another important tool is Podman [77] as an open-source tool. It totally supports Docker images and Dockerfiles [78], and it is a good alternative to Docker. The main reason for using Podman is that it does not require root privileges to run containers, which enhances security and simplifies deployment in certain environments. Both tools provide robust features for building, running, and managing containers, making them suitable choices for the automated testing pipeline. Podman is added just for comparing the results of both tools and checking the stability of the pipeline by using different containerization tools [79] [80].

4.2.2 Base Image Provisioning

Docker usage is primarily based on Dockerfiles. In fact, a Dockerfile is the base configuration for the APTS that prepares a base image based on a Linux operating system [81], such as Debian [82] or Ubuntu [83]. After configuring the base image from an operating system, it is important to install the required tools and packages in the base container, such as qemu-kvm [84], the Java JDK [85] (required for the Android SDK [86]), and Mitmproxy for network interception. Then it should create and configure the Android emulator and other tools. At the end, it copies the required script to run a test for an application within a container.

To improve connectivity with the Docker system, build images and run containers from the base image, it is better to use the Docker SDK [87]. It enables working with Docker properly within a Python script without command integration.

For Podman, the same Dockerfile can be used, with the only difference being the SDK used to build and run the container, since Podman provides its own SDK [88]. It is compatible with Dockerfiles and Docker images.

Listing 4.1: Semi-code representation of the base container configuration.

```
# 1. Initialize Base Operating System
FROM ubuntu:24.04


# 2. Install Core System Dependencies
INSTALL aapt, openjdk-21-jre, qemu-kvm, libvirt, bridge-utils


# 3. Setup Network Interception Tool
DOWNLOAD and INSTALL mitmproxy


# 4. Setup Android Environment
DOWNLOAD Android Command Line Tools
SET ENV ANDROID_SDK_ROOT = /root/Android/Sdk
UPDATE PATH variable


# 5. Install Android SDK Packages & Emulator
EXECUTE sdkmanager to install:
    - platform-tools
    - system-images;android-32;default;x86_64
    - emulator


# 6. Configure Android Virtual Device (AVD)
CREATE AVD named "mitm_android_test" (Device: Nexus 5X, API 32)
CLEANUP temporary cache and image files


# 7. Inject and Execute the Experiment Pipeline
COPY run_experiment_multiple.sh to container root
MAKE script executable
CMD ["sh", "/run_experiment_multiple.sh"]
```

4.2.3 Automated Pipeline Script

In the previous sections, it was discussed how to prepare the base image of containerization. In this section, it will be explained how a script as an actor controls the pipeline system. Fundamentally, this script is the most important part of the system and serves as the integration point, orchestrating and organizing all the provided tools to run the test machines.

This script starts the procedure by getting input configuration from CLI, such as:

- Minimum available resources (CPU, Memory, etc.) to run a new container
- Delay between running each container
- Input dataset directory
- The name of the containerization tool (Docker or Podman)
- The name of the experiment (for better organization of results)

Subsequently, it tries to run a container for an APK in parallel, using an iteration script. First, it checks the remaining resources before running the new container to ensure the system won't run out of resources. Then it tries to extract information from the target APK and pass it to the container for testing, and it delays before repeating the procedure. This procedure will continue through the last APK in the dataset.

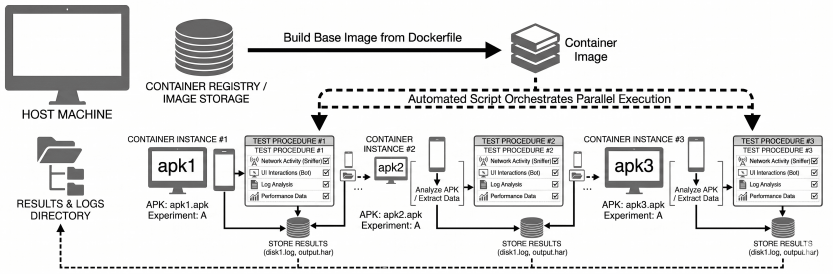


Figure 4.1: Workflow of the automated script orchestrating parallel container-based testing and data collection.

Listing 4.2: Semi-code representation of the main automated pipeline script.

```
INITIALIZE minimum_ram, minimum_cpu, wait_delays
CREATE directories for logs, results, and apks

PROMPT user FOR engine_type (Docker OR Podman)
PROMPT user FOR experiment_name AND duration

CONNECT to selected container client (engine_type)
START background thread for continuous resource logging

BUILD base container image from Dockerfile

LOAD apk_files_list FROM input directory

FOREACH apk IN apk_files_list:
    WHILE True:
        current_ram = GET available RAM
        current_cpu = GET available CPU

        IF current_ram > minimum_ram AND current_cpu >
           minimum_cpu:
            package_name = EXTRACT package name FROM apk
            CREATE result directories FOR package_name

            RUN container ASYNCHRONOUSLY WITH:
                image = base_image
                environment_variables = [package_name,
                   experiment_name, duration]
                volume_mounts = [apks_dir, results_dir]
                resource_limits = [cpu_quota, mem_limit]

            START background thread TO CAPTURE container logs

            WAIT wait_loop_after_run
            BREAK (proceed to next apk)
        ELSE:
            WAIT wait_loop_resource_check
```

4.3 Intra-Container Lifecycle

The main task of the entire procedure is testing, which should be done within the container. As discussed in the previous sections, the automated script attempts to run a container for each APK in the dataset. The container task is to organize tools (such as mitmproxy, the Android emulator, etc.) to handle network proxying and the emulator, install the application, and finally run random clicks.

Because all prompts run in a single terminal within the container, using tools like tmux [89] is vital for running parallel commands.

4.3.1 Emulator Setup and Virtual Device Management

The most important and resource-intensive part of the container test is the Android emulator. Fundamentally, the management of states and waits is crucial for the emulator. Additionally, the emulator is not ready for use in the MITM test on its own, and some parameter adjustments are required. Because there is no callback or event system in the Android SDK for the emulator, the required delay must be applied manually between time-consuming tasks, which makes implementation and testing more complex.

As discussed, the primary emulator needs to be modified to use the MITM test within a non-graphical container. The most important modification is to connect the emulator to Mitmproxy so that the emulator's network traffic passes through Mitmproxy's proxy. Due to support for setting the proxy port via the emulator CLI [90], the proxy-setting procedure can be applied at startup and does not require GUI manipulation or ROM customization of the emulator, but another part of the connection should be done by pushing the certificate file with a .pem [91] extension. Pushing the certificate is not applicable at emulator startup via the command line; it should be done in several steps. Furthermore, root access to the emulator and writable-system parameter is required to manipulate root files (certificates in the Android hierarchy are owned by the root user). Additionally, pushing the cert file depends on the Android Emulator version [92] and is split into API levels ¿ 28 and API levels ¡= 28. Pushing the certificate should be done when the emulator is running, but before that, the Root Access should be granted. Given the availability of Android Debug Bridge (ADB) [93] command line tools, the procedure will then continue by renaming the generated certificate file, pushing it, and rebooting the emulator.

Finally, the emulator is ready; the APK file will be installed on it, and it is ready to launch the test.

4.3.2 Automated UI Exercising via Monkey Testing

The most important part of gathering information from an Android application in the MITM procedure is the actor who manipulates the application's GUI to simulate normal user actions and cause the application to use its APIs. This procedure will also be done by another automation system.

Monkey [32] is a built-in Android SDK tool that is designed to do this exact task. The Monkey runs on the emulator or device and generates pseudo-random streams of user events, such as clicks, touches, and gestures, as well as system-level events. It is possible to use the Monkey to stress-test applications in a random yet repeatable manner.

Monkey supports multiple parameter adjustments [94]; meanwhile, the most important parameters for this project are the exercise duration, which accepts a duration in milliseconds, and the package name to set the application ID, so that Monkey knows which application to run and perform the exercise.

To increase the accuracy and quality of the exercise, additional parameters will be set to skip crashes and keep exercising continuously to the end of the duration.

4.3.3 Traffic Interception and HAR Generation

In the Emulator Setup section, it was discussed how to prepare the emulator and integrate it with Mitmproxy's proxy and certificate. Before reaching that point, it is necessary to run Mitmproxy to stabilize and generate the certificate file.

Mitmproxy is built from 3 modules: mitmproxy for an interactive command-line interface, mitmweb for a browser-based GUI, and mitmdump [95] for non-interactive terminal output. For this project, the best module to use is mitmdump, so within the container, mitmdump will be run to intercept the emulator's network.

Mitmproxy offers a comprehensive set of input parameters for modification [96], of which the most important one for this project is the custom proxy port. In fact, a predefined proxy port should be used for both the emulator and mitmdump.

Listing 4.3: Semi-code representation of the container's internal execution and testing script.

```
# 1. Initialization and Validation
LOAD environment variables (duration, current time, proxy port)
EXTRACT actual package_name FROM target APK
VERIFY AND OVERRIDE provided package_name IF necessary

# 2. Network Proxy Setup
RUN Mitmproxy IN BACKGROUND to capture network traffic
WAIT for proxy initialization

# 3. Emulator Boot Sequence
RUN Android Emulator IN BACKGROUND WITH proxy settings
WAIT for emulator boot to complete

# 4. System Certificate Injection
REQUEST root access via ADB
REMOUNT emulator file system
REBOOT emulator AND WAIT
REQUEST root access AND REMOUNT again
COMPUTE hash of Mitmproxy CA certificate
PUSH CA certificate TO Android system trust store
SET required permissions on certificate file
REBOOT emulator AND WAIT for restart

# 5. Application Deployment
INSTALL target APK via ADB
WAIT for installation completion

# 6. Automated Testing (UI Exerciser)
LAUNCH Monkey tool on target app FOR defined duration
WAIT for defined experiment duration
TERMINATE Monkey process

# 7. Finalization
SEND termination signal TO Mitmproxy
WAIT for network logs (HAR) to be written
EXIT
```

4.4 Dynamic Resource Orchestration and Monitoring

Developing, improving, and debugging an automated system that handles large-scale data and long-running tasks requires appropriate parameterization and a robust monitoring and logging system. Considering that it is not possible to implement the system in the first scripting/version without any bugs or problems, this highlights why the logging system is important: it allows you to use it like a time machine to check what the system was like at any past time.

Fundamentally, this approach separates into two aspects: how to use the host machine's resources most efficiently, and how to save logs at multiple levels.

4.4.1 Parallel Execution and Resource Thresholds

The duration of the entire experiment is determined by the number of APKs in the dataset, the duration of each test, and the number of parallel containers that can run, constrained by available resources. The more parallel containers, the shorter the overall testing duration.

$$T_{total} = \frac{N_{apk} \cdot (t_{test} + t_{delay})}{C_{parallel}} \tag{4.1}$$

where:

- T_{total}: The total estimated time to complete the entire testing pipeline.
- N_{apk}: The total number of APK files present in the input dataset.
- t_{test}: The duration of the test procedure executed inside each container.
- t_{delay}: The scheduled delay or wait time between launching consecutive containers.
- $C_{parallel}$: The maximum number of containers authorized to run simultaneously.

The required hardware resources for each container primarily determine this concurrency limit, with Random Access Memory (RAM) often the primary bottleneck. For instance, in this project, utilizing Android API 32 requires

substantial memory, necessitating an allocation of 8 GB per container. Consequently, the maximum number of parallel containers can be calculated based on the host system's memory capacity:

$$C_{parallel} = \left\lfloor \frac{M_{total} - M_{reserved}}{M_{container}} \right\rfloor \tag{4.2}$$

where:

- M_{total}: The total physical memory (RAM) available on the host machine.
- $M_{reserved}$: The memory safety margin reserved for the host operating system to ensure stability and prevent resource exhaustion (e.g., a 12 GB threshold).
- $M_{container}$: The dedicated memory allocated for a single container instance (e.g., 8 GB).
- $\lfloor \ldots \rfloor$: The floor function, ensuring the calculated capacity is rounded down to the nearest whole container.

4.4.2 Events Log System and Debugging

The logging system within this APTS has been implemented at different levels for debugging. In fact, the system generates and stores two types of logs: the APTS logs and container logs.

The APTS logs capture the system's runtime behavior and provide data on image building, application list iteration (which consequently runs containers), and resource monitoring. Furthermore, there is a separate tracking log for each container that records all events related to the emulator, monkey, and MitM.

To ensure traceability and facilitate troubleshooting, these records are systematically categorized and stored within a strict directory hierarchy on the host machine. The logging architecture is structured into the following components:

- **Image Build Logs:** Whenever a new base container image is constructed, the standard output and error streams of the build process are captured. These logs are essential for debugging dependency issues or build failures and are saved under the hierarchy:
  `logs/build/images/<TIMESTAMP>/image_build_log.txt`.

- **Resource Utilization Logs:** To monitor the health of the host machine and ensure the parallel execution threshold is not breached, a background thread continuously records CPU and RAM usage. This data is exported as a Comma-Separated Values (CSV) [97] file, allowing for post-experiment performance analysis. It is stored at: `logs/run/<EXPERIMENT_TYPE>/<TIMESTAMP>/resource_usage.csv`.
- **Container Execution Logs:** For every individual APK tested, the script attaches to the running container and streams its standard output and error logs. This captures the internal sequence of events, such as emulator boot status, certificate injection, and Monkey testing progress. To keep the results highly organized, these logs are grouped by the application's package name and execution time: `results/<EXPERIMENT_TYPE>/<PACKAGE_NAME>/<TIMESTAMP>/log.txt`.

By structuring the logs in this hierarchical manner, the system isolates errors effectively, allowing administrators to pinpoint whether a failure occurred at the infrastructure level (e.g., image building or resource exhaustion) or at the application testing level inside an isolated container.

Chapter 5

Data Processing Pipeline

In the previous chapter, it was discussed how an APTS works and how to implement and use it. In fact, by the end of the APTS procedure, it produces raw captured data in several HAR files.

Inherently, the APTS-exported data are malformed, noisy, and difficult to convert into meaningful information, and they require several processing steps to filter, denoise, and structure. The root cause of this ambiguity in the shape of data is related to the characteristics of testing. Due to capturing the entire network of an application, and in the bigger picture, an emulator, there are many repeated endpoints due to random and repeated clicking of the Monkey exerciser tool; furthermore, raw data may contain irrelevant endpoints that are not vital to the case study of the target application.

On the other hand, an HAR file may contain multiple endpoints (Base URLs) for a service (company, website, etc.), which, when studying and comparing exported information, should be separated into distinct comparisons.

In this chapter, the conversion and process are discussed for performing on raw data to produce vital information, while maintaining a comparable structure to ground-truth data to assess the accuracy of the output and the correctness of the APTS system.

5.1 Characterization of Network Noise

Any captured raw file in APTS represents the entire network interception of the emulator across the entire duration of the experiment. It contains any communication between the emulator and its connected network. Since the captured data is not just related to running the application (possibility of OS background work on other tasks), and there are several existing working

endpoints within the application's REST API, the captured data contains various types of noise.

The noise data is not limited to irrelevant endpoints; even APIs with the target Base URLs should be considered noise, as the information is not vital to reverse engineering, and especially for the study in this thesis.

5.1.1 Connectivity Check Endpoints

The network's connectivity check is the most frequently occurring noise in HAR files. These APIs are embedded in an Android device and are visible in every experiment and captured HAR file. The Operating System checks network connectivity using these APIs, and even when some applications are not interceptable by APTS system due to security measures, these kinds of links are present in captured data. The most repeated API URLs are:

- `http://www.google.com/gen_204`
  - A legacy Google connectivity check.
- `http://play.googleapis.com/generate_204`
  - Telemetry and network validation performed by Google Play Services.
- `http://connectivitycheck.gstatic.com/generate_204`
  - The primary endpoint used by modern Android versions to detect captive portals and internet access.

5.1.2 Static Assets and Media Resources

Some static URLs appear in every application and do not represent APIs. For example, developers may use online resources instead of local assets to reduce installation size and support dynamic content. To find this noise, look for URLs ending in media or file extensions, for example:

- **Multimedia Assets:**
  - *Images:* `.png`, `.jpeg`, `.jpg`, `.webp`, `.svg`
  - *Video:* `.mp4`, `.webm`, `.avi`, `.mkv`, `.gif`
  - *Audio:* `.mp3`, `.wav`, `.ogg`, `.aac`

- **Configuration and Constants:**
  - *Remote Configurations:* Endpoints fetching static `.json` or `.xml` payloads used strictly for feature flags or app settings rather than dynamic API interactions.
- **Typography and Styling Resources:**
  - *Web Fonts [98]:* `.woff`, `.woff2`, `.ttf`, `.otf`

5.1.3 Third-party Analytics and Telemetry SDKs

Modern Android applications integrate third-party Software Development Kits (SDKs) to improve the quality of their service. Consequently, captured network traffic often includes API calls to external domains that do not reflect the target application's core business logic. Because they are API services related to other endpoints, it is feasible to skip them and consider them as noise data. Most of these URLs are related to metrics and advertising services. Before any post-study analysis of the captured network traffic for each application, a priori prediction of comprehensive exclusion lists is infeasible due to the dynamic nature of third-party network traffic. After the capture step, it is not clear what the exact shape of the list is. For this reason, before scripting the filtering of the captured data, we need an additional step to investigate and identify the exact base URLs of services that are not vital to the analysis. In the next section, the Fast Peeking approach will be discussed to show how these noises are discovered and removed before analysis.

5.2 The Fast Peeking Approach

Fast Peeking is an approach to quickly view all captured URLs. In the System Architecture section, it is clear that after completing the experiment on an application dataset, a list of HAR files will be generated, each corresponding to an Android application in the dataset. In Fast Peeking, two or more automated scripts will be applied with almost identical missions and approaches on captured HAR files. At the first level, a script crawls all the HAR files and creates a large dataset of all the URLs accessed by all candidate applications in the experiment. It could be a JSON [49] array within a JSON object, containing multiple array items; each item represents a URL list for each application.

Listing 5.1: Generalized JSON schema representing the aggregated URL dataset generated during the Fast Peeking phase.

```
{
  "applications": [
    {
      "package": "<String: Target Application ID>",
      "urls": [
        "<String: Captured Endpoint URL>",
        "..."
      ]
    },
    {
      "package": "...",
      "urls": [
        "..."
      ]
    }
  ]
}
```

After this step, an initial review of the generated dataset reveals the number of HAR files collected and the URLs they contain. The first prepared dataset contains a large amount of noisy data, including the three types discussed in the previous part. Furthermore, in this step, hidden and unrecognizable noisy URLs are also discoverable, adding to the exclusion list for the next process. After the next process and filtering, the new file contains all the necessary information. With respect to the captured data, it is possible to see some of the applications removed within the second process and filter, which means all the captured URLs of that experiment of the application were classified as noise, which is due to some situations, such as: the candidate application was not installed on the emulator due to some problems like mismatching Android API versions; in some cases, the application crashed on the emulator or, for security reasons, the application did not permit to intercept its network.

5.2.1 Aggregated URL Dataset Generation

For implementing the Fast Peeking approach, several important points should be considered. Due to the large-scale captured data, it is crucial to skip the first filtering step. Fundamentally, the first step and the first script are

designed to gather all endpoints and URLs within a HAR file. An important step could be checking only for the Content-Type header for JSON [99] and non-empty body requests. Under this condition, all static and media files will be excluded, making the process faster and providing noiseless data at the end of the gathering endpoints phase.

Listing 5.2: Semi-code representation of the HAR parsing and URL dataset aggregation process.

```
INITIALIZE extracted_data AS {applications: [], count: 0}
SET root_directory TO experiment results path

FOREACH package_name IN root_directory:
    INITIALIZE empty SET unique_urls
    LOCATE "output.har" FOR package_name

    IF "output.har" EXISTS:
        har_data = PARSE JSON FROM "output.har"

        FOREACH entry IN har_data.log.entries:
            url = EXTRACT request.url
            body_size = EXTRACT response.bodySize
            content_type = EXTRACT "Content-Type" FROM response
               .headers

            // Filter non-API and heavy media traffic
            IF (content_type CONTAINS "application/json" OR
               body_size == 0):
                IF url STARTS WITH "http://" OR "https://":
                    ADD url TO unique_urls  // Set structure
                       ensures deduplication

    IF unique_urls IS NOT EMPTY:
        APPEND {package: package_name, urls: unique_urls} TO
           extracted_data.applications

extracted_data.count = LENGTH OF extracted_data.applications
SAVE extracted_data AS JSON
```

5.2.2 Heuristic Discovery of Noise Patterns

At this step, by reviewing all captured URLs, it is possible to produce a list of excluded base URLs for the second phase. Then, in the second script, all excluded URLs from the exported dataset of the first phase will be removed, and a new, clean dataset will be created. The most important part of the Fast Peeking approach is the repeatability of this step, because it is vital to be able to check the exported dataset from the first phase and apply filtering and exclusion across multiple runs of the second phase.

Listing 5.3: Semi-code representation of the heuristic-based URL filtration and noise elimination process.

```
DEFINE exclude_domains AS LIST OF known noise domains (e.g., "
   crashlytics.com", "adjust.com")

LOAD json_data FROM THE FIRST PHASE OF FAST PEEKING
INITIALIZE filtered_apps AS EMPTY LIST

FOREACH app IN json_data.applications:
    INITIALIZE filtered_urls AS EMPTY LIST

    FOREACH url IN app.urls:
        hostname = PARSE HOSTNAME FROM url (e.g., using
           urlparse)
        is_noise = FALSE

        // Check if the hostname belongs to any excluded domain
        FOREACH domain IN exclude_domains:
            IF domain IS IN hostname:
                is_noise = TRUE
                BREAK

        IF NOT is_noise:
            APPEND url TO filtered_urls

    // Drop the application entirely if all URLs were noise
    IF filtered_urls IS NOT EMPTY:
        APPEND {package: app.package, urls: filtered_urls} TO
           filtered_apps

INITIALIZE final_data AS {applications: filtered_apps, count:
   LENGTH OF filtered_apps}
SAVE final_data AS JSON TO a NEW FILE
```

5.3 Noise Reduction and HAR Filtering

To ensure the integrity of subsequent analysis, captured HAR files must undergo rigorous noise reduction. Leveraging the explicit exclusion lists generated during the Fast Peeking stage, a dedicated automated script systematically parses the aggregated dataset. For every candidate application, the script identifies and excises non-functional endpoints and irrelevant telemetry data. Crucially, to preserve experimental repeatability [17], this process does not mutate the original network logs. Instead, it generates a distinct, noise-free HAR file for each application, ensuring the raw captured data remains immutable and accessible for future auditing and review.

Listing 5.4: Semi-code representation of the automated noise reduction and HAR file filtration script.

```
DEFINE excluded_extensions AS list of media and static file
   suffixes
DEFINE excluded_keywords AS list of telemetry and tracking
   domains

PROMPT user FOR experiment_name
SET input_directory AND output_directory
CREATE output_directory IF NOT EXISTS

INITIALIZE audit_log (eliminated.csv) WITH headers ["
   package_name", "url"]

FOREACH file IN input_directory:
    IF file IS a ".har" file:
        package_name = EXTRACT name FROM file
        har_data = PARSE JSON FROM file
        filtered_entries = EMPTY LIST

        FOREACH entry IN har_data.log.entries:
            url = CONVERT entry.request.url TO lowercase

            IF url CONTAINS ANY keyword IN excluded_keywords:
                APPEND [package_name, url] TO audit_log
                CONTINUE

            IF url ENDS WITH ANY extension IN
               excluded_extensions:
                APPEND [package_name, url] TO audit_log
                CONTINUE
```

```
            APPEND entry TO filtered_entries

        IF filtered_entries IS NOT EMPTY:
            har_data.log.entries = filtered_entries
            SAVE har_data AS JSON TO output_directory/file
```

5.4 HAR Segmentation by Base API

The study captured HAR files that demonstrate that each application may contain multiple base URLs, even after removing noise. In fact, it shows that the API services for those applications are split across several base URL APIs, such as `example.com`, `api.example.com`, and `example.it`, all of which are present in the captured HAR file, and the philosophy of HAR Segmentation by base URL API is defined accordingly. Also, discovering the base URL APIs would be impactful for two other reasons in the evaluation system. The first is that for many conversion tools, from HAR to OAS, the HAR file's base URL is required; the second is that before the conversion step, when matching two OAS files, a smaller OAS file makes the matching process faster and lighter. An automated script can lead this process. The script would crawl all the HAR files iteratively. Within each iteration, it finds all distinct base URLs and creates a HAR file for each. For better management of the HAR files and easier identification of their relatives, the created HAR files would use the same name as the original HAR file, with a number for separation; for instance, example.har with two base URLs would be example_1.har and example_2.har

Listing 5.5: Semi-code for HAR segmentation using base URL bucketing.

```
SET input_dir AND output_dir
CREATE output_dir IF NOT EXISTS

FOREACH file IN input_dir:
    IF file IS NOT ".har" THEN CONTINUE

    har_data = PARSE JSON FROM file
    buckets = EMPTY MAP

    // Group all entries by their Base URL in a single pass
    FOREACH entry IN har_data.log.entries:
        base_url = EXTRACT SCHEME_AND_DOMAIN FROM entry.request
            .url
```

```
        APPEND entry TO buckets[base_url]

    // Export each grouped bucket as a distinct HAR file
    index = 1
    FOREACH base_url, entries IN buckets:
        new_har = COPY OF har_data
        new_har.log.pages = EMPTY LIST
        new_har.log.entries = entries

        new_filename = EXTRACT NAME FROM file + "_" + index + "
            .har"
        SAVE new_har AS JSON TO output_dir / new_filename

        index = index + 1
```

5.5 Automated OAS Generation

By the end of all procedures for filtering and splitting HAR files based on the base API, the output is ready for conversion to OAS. Because many tools that convert HAR to OAS require a base URL, before running any automation script on a single HAR file, it is necessary to identify the base API URL in that file. Fundamentally, the automation script tries to load all of the HAR files sequentially in an iteration, then within the loop it extracts the base URL and runs the command for each tool, and stores the exported OAS in the output directory with the same name as the input HAR but in a different extension (.json or .yaml, depending on the conversion tool)

Listing 5.6: Semi-code representation of the automated OAS generation process

```
SET input_directory TO path of filtered and split HAR files
SET output_directory TO path for generated OAS files
CREATE output_directory IF NOT EXISTS

DEFINE conversion_tools AS LIST OF tools (e.g., ToolA, ToolB)

// Outer Loop: Iterate through all HAR files sequentially
FOREACH har_file IN input_directory:
    IF har_file ENDS WITH ".har":

        // Step 1: Load HAR and identify the Base URL
        har_data = PARSE JSON FROM har_file
        base_url = EXTRACT BASE_URL FROM har_data.log.entries
```

```
        base_filename = EXTRACT NAME FROM har_file (removing ".
            har")

        // Inner Loop: Run the command for each conversion tool
        FOREACH tool IN conversion_tools:

            // Step 2: Prepare output path using the tool's
                native extension
            output_file_path = output_directory + "/" +
                base_filename +
                                "_" + tool.name + tool.
                                    output_extension

            // Step 3: Construct and execute the tool command
            conversion_command = CONSTRUCT COMMAND(
                executable = tool.executable_path,
                target_url = base_url,
                input_file = har_file,
                output_file = output_file_path
            )

            EXECUTE SYSTEM CALL (conversion_command)

            IF execution IS SUCCESSFUL:
                LOG "Success: Generated OAS with " + tool.name
                    + " for " + base_url
            ELSE:
                LOG "Error: Tool " + tool.name + " failed on "
                    + base_url
```

Chapter 6

Experimental Settings

This chapter focuses on the tools used for converting and comparing OAS files. The main objective of this chapter is to define the tools and compare them, and then to define the scoring and metrics system for evaluating the APTS system. Finally, the threats to validity of the experiment are discussed. Initially, the tools for conversion and their differences are defined; then, the tools for comparison and their differences are described. For each tool, its main features and characteristics are outlined.

6.1 OAS Inference Tools

OAS inference tools are designed to automatically generate OAS documents from various sources, such as HAR files, API traffic, or even code annotations. These tools use various algorithms and heuristics to analyze input data and produce a structured OAS document describing the API's endpoints, methods, parameters, and responses. In this section, several popular OAS inference tools used in this project are discussed, along with their features and capabilities.

6.1.1 Har-to-OpenAPI

HAR to OpenAPI [62] is a tool based on **har2openapi** [63] that is developed in JavaScript. Although this tool is based on JavaScript, it supports robust Command-Line Interface (CLI) Integration and is highly practical for use with APTS, which is developed in Python. It is also easy to run the tool's command-line prompts from APTS . This tool supports Automated Path Parameterization, which is one of the most critical features for specification inference. It allows the tool to dynamically analyze URLs

to detect identifiers (like UUIDs or specific numeric lengths via a MinimumLengthForNumericPath parameter) and abstracts them into variables. For instance, converting `/api/users/123e4567-e89b...` into `/api/users/uuid`. This prevents the generation of thousands of duplicate endpoints in the final OAS. This tool also supports Native Noise Reduction and Filtering. It features built-in cleaning mechanisms like `filterStandardHeaders` and `dropPathsWithoutSuccessfulResponse`. It is discussed how the data processing pipeline is applied to reduce noise, but this tool's internal noise reduction can further enhance the quality of the generated OAS by automatically excluding irrelevant or non-functional endpoints. Furthermore, this tool supports Type Inference to analyze the query, path to mathematically infer their scalar types, and Authentication Header Discovery, which enables the tool to attempt to automatically identify and extract authorization mechanisms from the HAR file.

6.1.2 mitmproxy2swagger

mitmproxy2swagger [61] is the most compatible tool with **mitmproxy** [39] in theory, and it is developed and designed to do specialized conversion from HAR files generated by mitmproxy to OAS files (Mitmproxy tool is the primary tool that is used within APTS containers). This Python-based tool has a straightforward CLI and is designed to be easily integrated into the APTS pipeline. All the HAR files will be converted to OAS version 3.0, the most recent OAS version. It supports Sensitive Data Warnings, which means that by using the `--examples` or `--headers` command-line flags, it will inject the actual captured payloads. This can embed sensitive data (like authentication tokens or passwords) directly into the final OpenAPI document. Meanwhile, there is a bottleneck in this tool: unlike fully autonomous tools, this tool requires the user to run the command, manually open the generated file to remove `ignore:` prefixes from the desired paths, and then run the command a second time. This could be a problem for large-scale experiments, as it is not practical to do this manually for each generated OAS file.

6.1.3 RestTestGen

RestTestGen-HAR2OpenAPI [65] is a Java-based tool that was developed at the University of Verona. This tool is a specific branch of RTG and designed for automated black-box testing [45] of RESTful web APIs. As a testing tool,

it incorporates various strategies that are valuable for identifying bugs and vulnerabilities in the API under test. It interacts with the API solely through HTTP, without requiring access to the source code. The only requirement is an OpenAPI specification of the API being tested. In addition to its role as a testing tool, RestTestGen also serves as a framework, offering multiple components and features to assist researchers, practitioners, and developers in implementing custom testing strategies. This framework empowers users to tailor their testing approach to specific requirements and explore innovative methods to enhance their API testing endeavors. This tool supports various other features, such as:

- Custom parser for OpenAPI (v3) specifications (JSON and YAML).
- Operation Dependency Graph (ODG) to model data dependencies among operations.
- Dictionaries to store values observed at testing time for future use
- Smart parameter management using hierarchical data structures.
- Additional framework components (e.g., mutation operators, test oracles, and coverage measurements).

6.1.4 Comparative Summary of Conversion Features

Fundamentally, any Conversion tool has its own features and they are designed for specific usage.

Table 6.1: Comparative analysis of conversion features across OAS inference tools.

Feature	Har-to-OpenAPI	mitmproxy2- swagger	RestTestGen
Core Language	JS / TS	Python	Java
Primary Output	v3.0 / v3.1	v3.0	v3.0
CLI Integration	High (1-pass)	Med (2-pass)	High (Docker)
Auto-Param.	UUID / Regex	Manual	ODG-Based
Noise Reduction	Built-in	Manual Prefix	Heuristics
Type Inference	Scalar Detect	Via Flags	Dictionaries
Auth Discovery	Heuristic	Header Flags	JSON Cmds
State Modeling	No	No	Yes (ODG)
Payload Support	JSON Schema	Optional	Hierarchical
Target Use Case	General	Reverse Eng.	API Testing

* **ODG (Operation Dependency Graph):** A structural model that maps data dependencies between API operations to handle stateful request sequences.

6.2 OAS Comparison Tools

In this section, several popular OAS comparison tools used in this project are discussed, along with their features and capabilities. These tools are designed to compare two OAS documents and identify differences in endpoints, parameters, responses, and other components. The main tools discussed are **openapi-diff** [100], **oasdiff** [101], **api-schema-diff** [102], and **RestTestGen** [65]. These tools are integrated outside the APTS pipeline, as they are used for post-processing and analysis of the generated OAS files. They provide insights into the accuracy of the inferred specifications by comparing them against ground-truth OAS documents, which are manually created based on the known API structure of the target applications.

6.2.1 openapi-diff

openapi-diff [100] is a Java-based tool that provides a comprehensive comparison of two OAS documents. Due to Java based implementation, it can be utilized for post processing after the APTS job, meanwhile its docker image is available on Docker Hub [103] to make usage easier. This tool supports multiple features, such as:

- **Fail-on-changed and Fail-on-incompatible:** These features allow users to control the flow if failing the comparison based on the type of differences.
- **Config-file and Config-prop:** Config file to override default behavior and Config property to override default behavior.
- **Set custom header** This feature allows users to set custom headers for authentication when comparing OAS documents that require authorization.
- **Multiple export type supporting:** CLI Output, Markdown, JSON, HTML and Asciidoc.
- **Multi layer logging system:** TRACE, DEBUG, INFO, WARN, ERROR, OFF

6.2.2 oasdiff

oasdiff [101] is a modern specification tool written in the Go programming language [104]. This tool is designed for comfortable usage, and it supports multiple features, such as:

- **Different types of output:** It can export the difference between two OAS in different formats, such as HTML, JSON, Markdown, markup, text, yaml, etc.
- **Compare APIs split across multiple files**
- **Filter endpoints:** Narrowing the diff to a subset of endpoints
- **Normalization:** Merging schemas, merging common parameters, Path prefix modification, Case-insensitive header comparison, etc.
- **API lifecycle:** Deprecate APIs and parameters and API stability levels (draft / alpha / beta / stable)

- **Filtering changes:** Exclude certain kinds of changes, Track OpenAPI extensions, Exclude specific extension names

Furthermore, the creators offer some paid online services with specific features like Rich PR comment and Commit status check blocks that are vital for software developers.

6.2.3 api-schema-diff

api-schema-diff [102] is a fully open-source tool that is written in Python. This modern tool that is published early, supports multiple features and a robust CLI support. There are multiple comprehensive features within the tool, such as:

- **Multiple input type supports**
- **Breaking changes detectors**
- **Non-breaking changes detectors**
- **CI/CD Integrations**
- **Multiple Schema Directories**
- **Exclusive Docker Image**

6.2.4 RestTestGen

RestTestGen [65], is another branch of the RestTestGen framework that is designed for testing RESTful APIs. In addition to its OAS inference capabilities, RestTestGen also includes a powerful OAS comparison feature. This tool can compare two OAS documents and identify differences in endpoints, parameters, responses, and other components. It supports various features, such as:

- **Different type of logging:** DEBUG, INFO, WARN, ERROR
- **Gathering information about authentication**
- **Supporting configuration files:** to control the behavior of the comparison process.

6.2.5 Comparative Summary of Comparison Features

By the end of the discussion of the comparison tools, it is clear that each tool has its own features and capabilities, and they are designed for different use cases. The following table provides a comparative analysis of the features across the discussed OAS comparison tools.

Table 6.2: Comparative analysis of features across OAS comparison tools.

Feature	**openapi-diff**	**oasdiff**	**api-schema-diff**	**RestTestGen**
Core Language	Java	Go	Python	Java
Execution Method	Docker / CLI	CLI / Binary	Docker / CLI	Framework / CLI
Breaking Change Det.	Excellent (Categorized)	Supported (Flags)	Highly Focused	Via Security Oracles
Output Formats	MD, JSON, HTML, Asciidoc	YAML, HTML, Text, JSON	Markdown, JSON	JSON, Console
Schema Normalization	Minimal	Advanced (Prefix/Case)	Standard	Dictionary-based
Lifecycle Support	Standard	Advanced (Draft to Stable)	Standard	No
Auth Comparison	Custom Headers	No	Supported	Extracted Commands
CI/CD Readiness	High	High (PR/Status)	High (Integration)	Medium (Testing-focused)
Logging Granularity	6-Layer System	Standard	Robust	4-Layer System
Primary Use Case	Backward Compatibility	Performance & Normalization	Modern Python Ecosystem	Test Verification

6.3 Threats to Validity

In this section, potential threats to the validity of the experimental results are discussed, categorized into external, internal, construct, and conclusion validity.

6.3.1 External Validity

The primary threat to external validity is the selection of target APIs. While the experiment includes a variety of RESTful services, the results may not generalize to all API architectures, such as GraphQL [105] or gRPC [106]. Furthermore, the reliance on captured HAR files means that the quality of the inferred specification is directly bounded by the coverage of the initial network traffic.

6.3.2 Internal Validity

Internal validity may be affected by the manual creation of the ground-truth OpenAPI documents. Although these files were developed based on official documentation, human error in the manual specification process could lead to false positives or negatives during the comparison phase. Additionally, the configuration parameters for the inference tools (e.g., path parameterization thresholds) were kept at default values, which may not be optimal for every target API .

6.3.3 Construct Validity

Construct validity concerns whether the comparison metrics accurately reflect the system's performance. The use of automated diffing tools like **openapi-diff** and **oasdiff** provides a standardized measure of structural differences, but these tools may differ in their classification of b̈reakingv̈ersus n̈on-breakingc̈hanges, potentially affecting the final evaluation scores.

Chapter 7

Evaluation Baseline and Dataset

In this chapter it will be discussed how to prepare required datasets to run and evaluate APTS . Fundamentally APTS requires a dataset of Android applications and after ending the pipeline of the test it is required to compare generated OAS with the ground truth. This chapter demonstrates approaches to obviate demanded inputs for the system.

7.1 Ground Truth Definition

Evaluation of the APTS's outputs demands preparing specific information to compare with. Basically, for evaluation of the system the shape of output OAS such as base URL API, endpoints, headers, parameters, etc. should be controlled. Every difference is meaningful within this comparison. Generally comparison of two OAS will find three types of APIs:

- **Missed:** demonstrate APIs that APTS did not detect them
- **New:** API drifts [107] from the ground truth
- **Present:** Common APIs in Captured OAS and Ground Truth

7.2 Public OAS Repositories

This section focuses on how to prepare demanded OAS for preparing by APTS captured OAS . Generally it is expected to run APTS first then try to investigate for corresponding OAS files but after many efforts in this project the experiment demonstrates that instinct or random selection of candidate applications does not guarantee availability of published OAS files corresponding

to each application that is intercepted. For this reason, before running the APTS it is necessary to investigate target candidate applications. There are many official and unofficial sources that could be sources of OAS ground truth files. Basically some of the public services publish their official OAS corresponding to their services but also there are open-source community that tries to collect OAS files from official sources or manual capturing.

7.2.1 API Guru

The most comprehensive unofficial source is **API Guru** [108]. This service contains official and unofficial OAS files that are available to the public. API Guru offers practical features for investigating OAS files on their website also the raw OAS files are available on their public GitHub repository [109] which is used in this project.

7.2.2 Isolating Mobile-Targeted Specifications

The most critical part of Ground Truth selection is related to choosing mobile only OAS files. Fundamentally within the open-source OAS repositories there is not any possibility to filter just only OAS files that are related to mobile applications. For this reason, it is necessary to investigate each OAS file and try to find out if it is related to a mobile application or not. Checking every single OAS file and investigating it by their references and owner services is almost impossible due to the large number of OAS files, for this reason, an automated script is designed to check each OAS file and try to find out if it is related to a mobile application or not. The script is designed to check the OAS file for some specific keywords. There are multiple keywords in different types, that are signs to be related to mobile applications, then the script multiplies every sign keyword with its respective weight that are predefined in the script, and then it calculates a score for each OAS file. If the score is higher than a specific threshold, the OAS file is considered as related to mobile applications and added to the list of candidate OAS files for the experiment. The heuristic keywords and conditions are defined such as:

- **android keywords (weight 2):** Detects words such as android, androidId, apk, play store, packageName. Indicates direct reference to the Android platform or packaging.

- **push stack (weight 3):** Detects fcm, gcm, Firebase [110], push, notifications, device token. Strong indicator of mobile push notification backends.
- **device registration (weight 4):** Detects endpoints such as registerDevice, /device/register, unregisterDevice. Explicit device registration — very strong signal for mobile backends.
- **device identifiers (weight 3):** Detects fields such as deviceId, installationId, manufacturer, model, osVersion, appVersion. Typical identifiers sent by mobile apps.
- **platform headers (weight 3):** Detects headers like X-Platform: android, or user-agent strings mentioning Android. Explicit platform declaration.
- **mobile features (weight 2):** Detects terms such as deeplink, app links, intent, android-app://, in-app billing. Features strongly tied to mobile ecosystems.
- **version checks (weight 3):** Detects endpoints like /app/version, /update/check, minSupportedVersion, forceUpdate. Used by apps to enforce client updates.
- **telemetry (weight 2):** Detects paths such as /crash, /analytics, /events, sessionStart, deviceInfo. Suggests mobile crash and telemetry reporting.
- **mobile hosts (weight 2):** Detects URLs or hostnames containing mobile, android, or app. Indicates platform-specific subdomains.
- **docs metadata (weight 2):** Detects mentions of android or mobile in OAS info.title, info.description, or tags. Explicit documentation reference.
- **mobile auth (weight 2):** Detects OAuth2 [111] flows with PKCE [112], device code, or explicit mention of "mobile app" in authentication context. PKCE and device-code flows are designed for native/mobile clients.
- **push topics (weight 1):** Detects topic-based subscription endpoints (/subscribe, /unsubscribe, topics). Indicates Firebase-style push topic management.

Listing 7.1: Semi-code for isolating mobile-targeted OAS files based on heuristic scoring.

```
SET THRESHOLD = 15
SET SCORING_RULES = MAP OF (category -> {weight, terms_list})

FUNCTION calculate_mobile_score(file_content):
    total_score = 0
    content_lower = CONVERT file_content TO LOWERCASE

    FOREACH rule IN SCORING_RULES:
        weight = rule.weight

        FOREACH term IN rule.terms_list:
            occurrences = COUNT term IN content_lower
            IF occurrences > 0 THEN
                total_score = total_score + (occurrences *
                    weight)

    RETURN total_score

FUNCTION filter_mobile_oas(folder_path):
    mobile_candidates = EMPTY LIST
    all_oas_files = SCAN DIRECTORY folder_path

    FOREACH file IN all_oas_files:
        content = READ file
        score = calculate_mobile_score(content)

        IF score >= THRESHOLD THEN
            APPEND {file_name, score} TO mobile_candidates

    RETURN mobile_candidates
```

7.3 APK Dataset Selection and Sourcing

The selection of Android applications for the experiment is a critical step. In the previous sections, it is discussed how to prepare the ground truth OAS files, and how to filter them to be related to mobile applications, by the end of that process, a list of candidate OAS files is prepared. For each OAS file, it is required to find the corresponding Android application that uses the API described in that OAS file. This step is faced by several challenges, the first one is preparing APK files that are impossible to be downloaded from the

Google Play Store and manually downloading, and the second one is finding the corresponding APK file for each OAS file within the dataset. Because of several available OAS files, it is not possible to find the corresponding APK file for each OAS file by manual searching, for this reason, an automated script is designed to search for the corresponding APK file for each OAS file. Due to non presence the exact name of the application in the OAS file, each OAS file is searched by domain name, and if the domain name is found in the OAS file, it is used as a keyword to search for the corresponding APK file. This procedure is not perfect, but it is the most practical way to find the corresponding APK file for each OAS file within the dataset. At the end of this process, a dataset of APK files is prepared, each OAS files is corresponding to zero, one or more APK files, and each APK file is corresponding to one OAS file.

Listing 7.2: Semi-code for matching candidate OAS files to corresponding Android APKs using domain name extraction.

```
SET candidate_oas_list = LOAD previously_filtered_oas
SET oas_to_apk_mapping = EMPTY MAP

FOREACH oas_file IN candidate_oas_list:
    content = PARSE oas_file

    // Extract domain name to use as search keyword
    domain_name = EXTRACT DOMAIN FROM content.servers.url

    IF domain_name IS NULL THEN CONTINUE

    // Search APK repository (e.g., AndroZoo) using domain name
    found_apks = SEARCH_APK_REPOSITORY(domain_name)

    // One OAS file can correspond to zero, one, or multiple
       APKs
    IF LENGTH(found_apks) > 0 THEN
        oas_to_apk_mapping[oas_file] = found_apks

RETURN oas_to_apk_mapping
```

7.3.1 Androzoo

Androzoo [113] is a comprehensive repository of Android applications collected from various sources, including the Google Play Store [114], third-party app stores, and direct submissions that is prepared by the University of Luxem-

bourg. [18] Androzoo offers a dedicated API to access to their private database. Within this project Androzoo is used as the primary source for sourcing APK files corresponding to the candidate OAS files.

Chapter 8

Experimental Results and Evaluation

This chapter presents the results obtained from executing the APTS pipeline. The evaluation focuses on the stability of the inference tools, the accuracy of the generated OAS compared to the ground truth, and the performance metrics of the system. The results are analyzed using various comparative tools—specifically **api-schema-diff**, **oasdiff**, **openapi-diff**, and **RestTestGen**—to provide a comprehensive and multi-faceted assessment.

8.1 Discovered Ground Truth

In the previous chapter the approach of finding APK datasets for APTS was explained. The result of running scoring system on API guru repository was **259** target OAS files and **10,958** responded APK files that **9674** were present in AndroZoo dataset. The downloaded APK list occupied about **160 GB** on the host machine storage. Firstly **5 minutes** test by APTS has run on all target applications to understand which applications are interceptable by APTS and **2073** were intercepted at the end. Then the primary experiment on the interceptable applications was done, dedicating 1 hour for each application. By running **12** APTS containers on the host machine simultaneously the entire process took almost **8 days**. By the end of running 1 hour test experiment on each candidate application and processing output HAR files and converting to OAS by different OAS conversion tools and comparing ground truth and output of each, these results were found:

Table 8.1: Total matched OAS files (including multiple occurrences per API) between inference outputs and ground truth.

Conversion Tool	Matches Found
RestTestGen (RTG)	30
har-to-openapi	40
mitmproxy2swagger	40

The above table shows performance, crash-free and compatibility of conversion tools for converting HAR to OAS .

8.2 Execution Overview and Tool Stability

This section provides an overview of the execution of comparison tools and their stability in processing the generated OAS files. Fundamentally, the stability of the inference tools means that the percentage of successfully generated OAS files in their outputs, in fact how many of the HAR files are able to be converted to OAS files by the tool without any error or crash.

It should be emphasized that the discrepancy between the figures in Table 8.1 and Table 8.2 arises from the difference between the total number of processed files and unique domains. While the former reports all successful matches (40 and 30), the latter focuses on unique Base URLs to avoid redundancy caused by multiple HAR captures for the same API (e.g., GitHub). Consequently, when filtering for distinct APIs, the coverage reflects 28 unique domains for the Python-based tools and 21 for RTG .

Table 8.2: Comparison success across tools for matched OAS files.

Base URL	openapi-diff	oasdiff	api-schema-diff	RTG
Converter: RestTestGen (RTG)				
`api.avaza.com (1)`	✓	–	✓	✓
`api.avaza.com (2)`	–	✓	–	–
`api.flickr.com (1)`	✓	–	✓	✓
`api.flickr.com (2)`	–	✓	–	–
`api.github.com (1)`	✓	–	✓	–
`api.github.com (2)`	✓	–	✓	–
`api.github.com (3)`	✓	–	✓	–
`api.github.com (4)`	✓	–	✓	–

Continued on next page

Table 8.2 – *Continued from previous page*

Base URL	openapi-diff	oasdiff	api-schema-diff	RTG
api.github.com (5)	–	✓	–	–
api.github.com (6)	–	✓	–	–
api.github.com (7)	–	✓	–	–
api.github.com (8)	–	✓	–	–
api.hubapi.com (1)	✓	–	✓	–
api.hubapi.com (2)	✓	–	✓	–
api.hubapi.com (3)	–	✓	–	–
api.hubapi.com (4)	–	✓	–	–
api.reverb.com (1)	✓	–	✓	✓
api.reverb.com (2)	–	✓	–	–
api.stripe.com (1)	✓	–	✓	✓
api.stripe.com (2)	–	✓	–	–
api.vimeo.com (1)	✓	–	✓	✓
api.vimeo.com (2)	–	✓	–	–
api.vtex.com (1)	✓	–	✓	✓
api.vtex.com (2)	–	✓	–	–
api.zenoti.com (1)	✓	–	✓	–
api.zenoti.com (2)	–	✓	–	–
api2.bigoven.com (1)	✓	–	✓	✓
api2.bigoven.com (2)	–	✓	–	–
app.asana.com (1)	✓	–	✓	✓
app.asana.com (2)	–	✓	–	–
app.launchdarkly.com (1)	✓	–	✓	–
app.launchdarkly.com (2)	–	✓	–	–
github.com (1)	✓	–	✓	–
github.com (10)	–	✓	–	–
github.com (2)	✓	–	✓	–
github.com (3)	✓	–	✓	–
github.com (4)	✓	–	✓	–
github.com (5)	✓	–	✓	–
github.com (6)	–	✓	–	–
github.com (7)	–	✓	–	–
github.com (8)	–	✓	–	–
github.com (9)	–	✓	–	–
rms.api.bbc.co.uk (1)	✓	–	✓	✓
rms.api.bbc.co.uk (2)	–	✓	–	–
s3.amazonaws.com (1)	✓	–	✓	✓
s3.amazonaws.com (2)	–	✓	–	–
sandbox.gerermesaffaires.com (1)	✓	–	✓	✓
sandbox.gerermesaffaires.com (2)	–	✓	–	–
slack.com (1)	✓	–	✓	✓
slack.com (2)	–	✓	–	–

Continued on next page

Table 8.2 – *Continued from previous page*

Base URL	openapi-diff	oasdiff	api-schema-diff	RTG
trashnothing.com (1)	✓	–	✓	✓
trashnothing.com (2)	–	✓	–	–
www.docusign.net	✓	–	✓	–
www.theblueallliance.com (1)	✓	–	✓	✓
www.theblueallliance.com (2)	–	✓	–	–
www.ticketmaster.com (1)	✓	–	✓	–
www.ticketmaster.com (2)	–	✓	–	–
Converter: Mitmproxy2swagger				
api.avaza.com (1)	✓	✓	✓	✓
api.avaza.com (2)	–	✓	–	–
api.flickr.com (1)	✓	✓	✓	✓
api.flickr.com (2)	–	✓	–	–
api.github.com (1)	✓	✓	✓	–
api.github.com (2)	✓	✓	✓	–
api.github.com (3)	✓	✓	✓	–
api.github.com (4)	✓	✓	✓	–
api.github.com (5)	–	✓	–	–
api.github.com (6)	–	✓	–	–
api.github.com (7)	–	✓	–	–
api.github.com (8)	–	✓	–	–
api.hubapi.com (1)	✓	✓	✓	–
api.hubapi.com (2)	✓	✓	✓	–
api.hubapi.com (3)	–	✓	–	–
api.hubapi.com (4)	–	✓	–	–
api.lyft.com (1)	–	✓	–	–
api.lyft.com (2)	–	✓	–	–
api.nytimes.com (1)	–	✓	–	–
api.nytimes.com (2)	–	✓	–	–
api.reverb.com (1)	–	✓	–	–
api.reverb.com (2)	–	✓	–	–
api.stripe.com (1)	✓	✓	✓	✓
api.stripe.com (2)	–	✓	–	–
api.twitter.com (1)	–	✓	–	–
api.twitter.com (2)	–	✓	–	–
api.vimeo.com (1)	✓	✓	✓	✓
api.vimeo.com (2)	–	✓	–	–
api.vtex.com (1)	✓	✓	✓	✓
api.vtex.com (2)	–	✓	–	–
api.zenoti.com (1)	✓	✓	✓	–
api.zenoti.com (2)	–	✓	–	–
api2.bigoven.com (1)	✓	✓	✓	✓
api2.bigoven.com (2)	–	✓	–	–

Continued on next page

Table 8.2 – *Continued from previous page*

Base URL	openapi-diff	oasdiff	api-schema-diff	RTG
app.asana.com (1)	✓	✓	✓	✓
app.asana.com (2)	–	✓	–	–
app.launchdarkly.com (1)	–	–	–	–
app.launchdarkly.com (2)	–	✓	–	–
github.com (1)	✓	✓	✓	–
github.com (10)	–	✓	–	–
github.com (2)	✓	✓	✓	–
github.com (3)	✓	✓	✓	–
github.com (4)	✓	✓	✓	–
github.com (5)	✓	✓	✓	–
github.com (6)	–	✓	–	–
github.com (7)	–	✓	–	–
github.com (8)	–	✓	–	–
github.com (9)	–	✓	–	–
gitlab.com (1)	–	✓	–	–
gitlab.com (2)	–	✓	–	–
platform.clearblade.com (1)	✓	✓	✓	✓
platform.clearblade.com (2)	–	✓	–	–
quotes.rest (1)	–	✓	–	–
quotes.rest (2)	–	✓	–	–
rms.api.bbc.co.uk (1)	✓	✓	✓	✓
rms.api.bbc.co.uk (2)	–	✓	–	–
s3.amazonaws.com (1)	✓	✓	✓	✓
s3.amazonaws.com (2)	–	✓	–	–
sandbox.gerermesaffaires.com (1)	✓	✓	✓	✓
sandbox.gerermesaffaires.com (2)	–	✓	–	–
slack.com (1)	–	✓	–	–
slack.com (2)	–	✓	–	–
trashnothing.com (1)	–	✓	–	–
trashnothing.com (2)	–	✓	–	–
worldtimeapi.org (1)	✓	✓	✓	✓
worldtimeapi.org (2)	–	✓	–	–
www.docusign.net	✓	✓	✓	–
www.theblue alliance.com (1)	–	✓	–	–
www.theblue alliance.com (2)	–	✓	–	–
www.ticketmaster.com (1)	–	✓	–	–
www.ticketmaster.com (2)	–	✓	–	–
Converter: HAR to OpenAPI				
api.avaza.com (1)	✓	–	✓	✓
api.avaza.com (2)	–	✓	–	–
api.flickr.com (1)	✓	–	✓	✓
api.flickr.com (2)	–	✓	–	–

Continued on next page

Table 8.2 – *Continued from previous page*

Base URL	openapi-diff	oasdiff	api-schema-diff	RTG
api.github.com (1)	✓	–	✓	–
api.github.com (2)	✓	–	✓	–
api.github.com (3)	✓	–	✓	–
api.github.com (4)	✓	–	✓	–
api.github.com (5)	–	✓	–	–
api.github.com (6)	–	✓	–	–
api.github.com (7)	–	✓	–	–
api.github.com (8)	–	✓	–	–
api.hubapi.com (1)	✓	–	✓	–
api.hubapi.com (2)	✓	–	✓	–
api.hubapi.com (3)	–	✓	–	–
api.hubapi.com (4)	–	✓	–	–
api.lyft.com (1)	✓	–	✓	✓
api.lyft.com (2)	–	✓	–	–
api.nytimes.com (1)	✓	–	✓	✓
api.nytimes.com (2)	–	✓	–	–
api.reverb.com (1)	✓	–	✓	✓
api.reverb.com (2)	–	✓	–	–
api.stripe.com (1)	✓	–	✓	✓
api.stripe.com (2)	–	✓	–	–
api.twitter.com (1)	✓	–	✓	✓
api.twitter.com (2)	–	✓	–	–
api.vimeo.com (1)	✓	–	✓	✓
api.vimeo.com (2)	–	✓	–	–
api.vtex.com (1)	✓	–	✓	✓
api.vtex.com (2)	–	✓	–	–
api.zenoti.com (1)	✓	–	✓	–
api.zenoti.com (2)	–	✓	–	–
api2.bigoven.com (1)	✓	–	✓	✓
api2.bigoven.com (2)	–	✓	–	–
app.asana.com (1)	✓	–	✓	✓
app.asana.com (2)	–	✓	–	–
app.launchdarkly.com (1)	✓	–	✓	–
app.launchdarkly.com (2)	–	✓	–	–
github.com (1)	✓	–	✓	–
github.com (10)	–	✓	–	–
github.com (2)	✓	–	✓	–
github.com (3)	✓	–	✓	–
github.com (4)	✓	–	✓	–
github.com (5)	✓	–	✓	–
github.com (6)	–	✓	–	–
github.com (7)	–	✓	–	–

Continued on next page

Table 8.2 – *Continued from previous page*

Base URL	openapi-diff	oasdiff	api-schema-diff	RTG
github.com (8)	–	✓	–	–
github.com (9)	–	✓	–	–
gitlab.com (1)	✓	–	✓	✓
gitlab.com (2)	–	✓	–	–
platform.clearblade.com (1)	✓	–	✓	✓
platform.clearblade.com (2)	–	✓	–	–
quotes.rest (1)	✓	–	✓	✓
quotes.rest (2)	–	✓	–	–
rms.api.bbc.co.uk (1)	✓	–	✓	✓
rms.api.bbc.co.uk (2)	–	✓	–	–
s3.amazonaws.com (1)	✓	–	✓	✓
s3.amazonaws.com (2)	–	✓	–	–
sandbox.gerermesaffaires.com (1)	✓	–	✓	✓
sandbox.gerermesaffaires.com (2)	–	✓	–	–
slack.com (1)	✓	–	✓	✓
slack.com (2)	–	✓	–	–
trashnothing.com (1)	✓	–	✓	✓
trashnothing.com (2)	–	✓	–	–
worldtimeapi.org (1)	✓	–	✓	✓
worldtimeapi.org (2)	–	✓	–	–
www.docusign.net	✓	–	✓	–
www.thebluealliance.com (1)	✓	–	✓	✓
www.thebluealliance.com (2)	–	✓	–	–
www.ticketmaster.com (1)	✓	–	✓	–
www.ticketmaster.com (2)	–	✓	–	–

Note: Numbered duplicates indicate the presence of multiple distinct HAR files for the same base URL.

8.3 Accuracy Evaluation: Inferred vs. Ground Truth

In this section, the accuracy of the inferred OAS files is evaluated against the ground truth specifications. Fundamentally, in the previous section the big picture of the stability and performance of the inference tools is discussed, in this section the accuracy of the generated OAS files is evaluated by comparing them to the ground truth OAS. To understand how much captured quantify the utility it is necessary to compare the generated OAS files to the ground truth OAS files in details.

8.3.1 Endpoint Coverage Analysis

The Table 8.2 demonstrates which comparison tools were able to successfully process the generated OAS files and report differences compared to the ground truth. By analyzing in details the results of comparison tools there are several vital information points. Generally **148,833** records about entire comparison process are recorded by all comparison tools.

Table 8.3: Distribution of generated comparison records per comparison tool.

Comparator Tool	**Total Records Processed**
RestTestGen (RTG)	113,004
oasdiff	35,534
openapi-diff	149
api-schema-diff	146
Total	**148,833**

These results show that the majority of the comparison records are generated by RestTestGen (RTG) and oasdiff, while openapi-diff and api-schema-diff contribute a significantly smaller number of records. The cause of the gap between the number of records generated by RTG and oasdiff compared to the other two tools is that RTG and oasdiff are able to process a larger number of OAS files successfully. Furthermore, the study of the records generated by each comparison tool reveals that each comparison tool has different behavior with respect to different conversion tools outputs. The sparsity of records generated by different conversion tools is different.

Table 8.4: Sparsity by Converter Tool)

Tool	Common	New	Missed
RTG	107	71	37,547
har-to-openapi	142	149	55,293
mitmproxy2swagger	142	89	55,293

Table 8.5: Sparsity by Comparator Tool

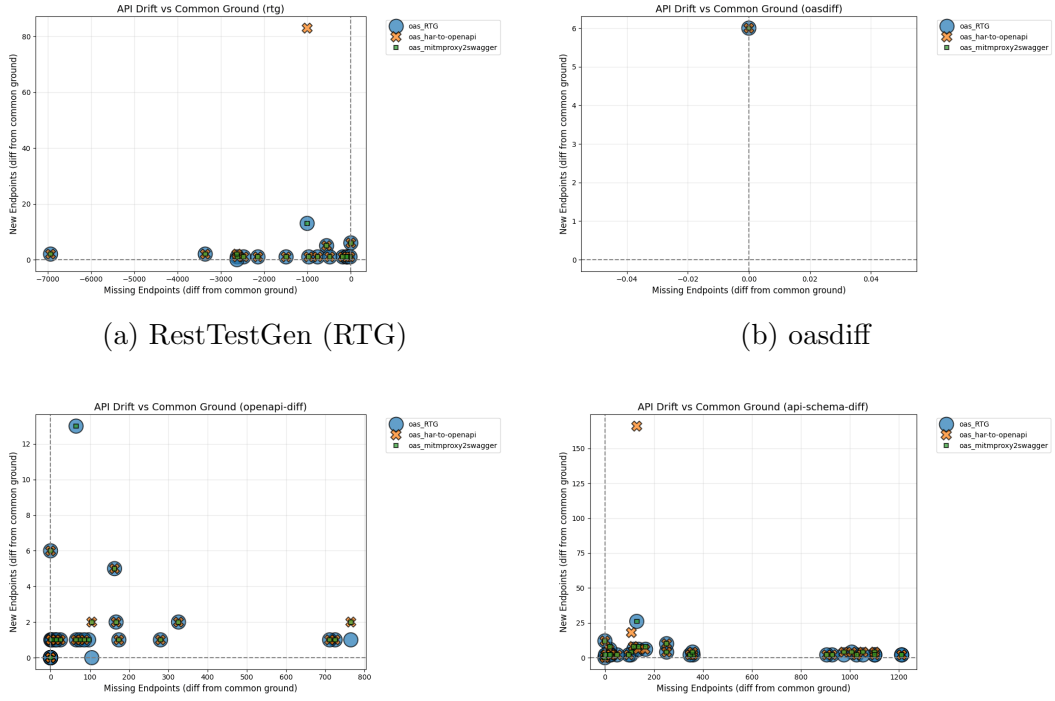
Tool	Common	New	Missed
api-schema-diff	146	0	0
oasdiff	3	262	35,269
openapi-diff	149	0	0
rtg	93	47	112,864

Table 8.6: By Converter Tool × Comparator Tool (combined)

Converter Tool	Diff Tool	Common	New	Missed
RTG	api-schema-diff	40	0	0
	oasdiff	1	52	10,195
	openapi-diff	41	0	0
	rtg	25	19	27,352
har-to-openapi	api-schema-diff	53	0	0
	oasdiff	1	140	12,537
	openapi-diff	54	0	0
	rtg	34	9	42,756
mitmproxy2swagger	api-schema-diff	53	0	0
	oasdiff	1	70	12,537
	openapi-diff	54	0	0
	rtg	34	19	42,756

8.3.2 Common Ground and Structural Overlap of Conversion tools

For a better understanding of the common results between converter tools by each comparing tool, the charts below are vital to study.



(a) RestTestGen (RTG)

(b) oasdiff

(c) openapi-diff

(d) api-schema-diff

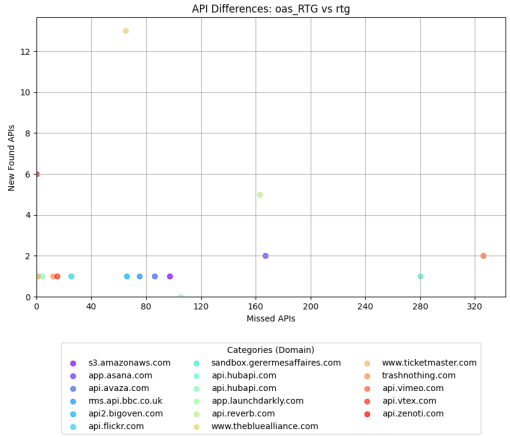
Figure 8.1: API differences (Missed vs. New Found APIs) evaluated by four distinct comparator tools.

8.3.3 Inference Tool Comparative Analysis

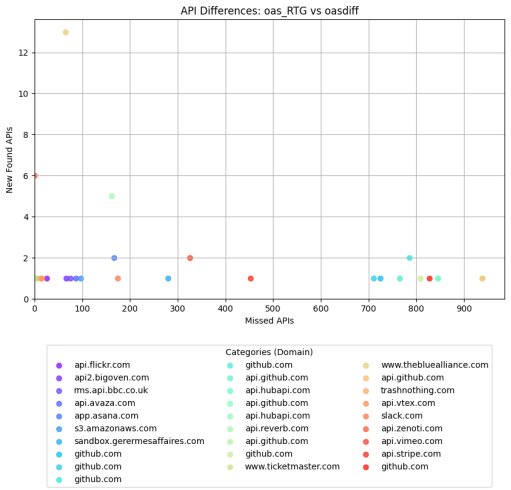
These results demonstrate that **Mitmproxy2swagger** is the most robust tool to convert captured HAR files to OAS files. It has found about 500 unique endpoints in a crashless process. Then **har-to-openapi** with 415 has a very similar behavior with the first tool. At the end **RestTestGen (RTG)** has the most fragile operation in comparing to the other tools.

8.4 OAS Comparator Behavior Analysis

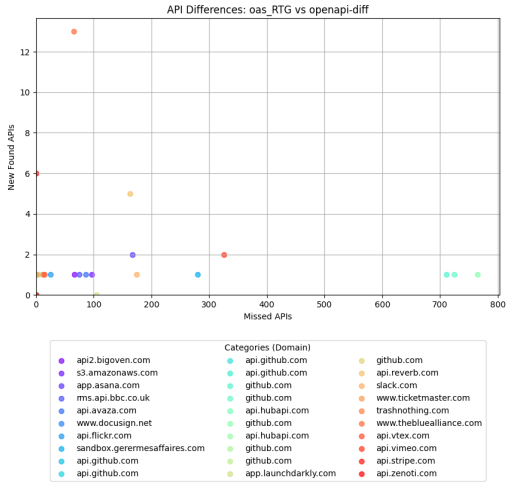
But in the comparison tools the results are significantly different. **RestTestGen (RTG)** is the most robust tool for generating more records from comparison operation with a large gap with respect to other tools. The second comparison tool was **oasdiff** with an acceptable result. **openapi-diff** and **api-schema-diff** have had just overall and superficial results.



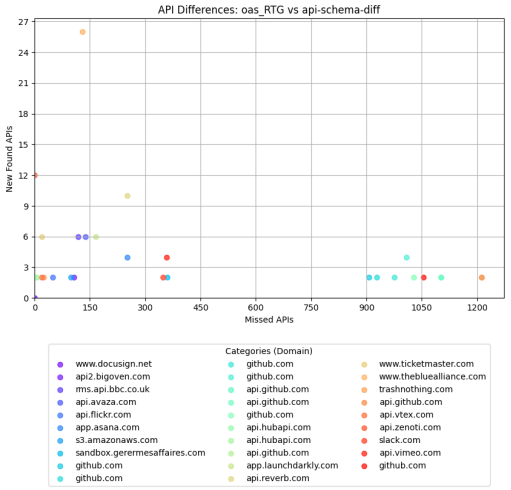
(a) RTG vs. RTG Comparator



(b) RTG vs. oasdiff

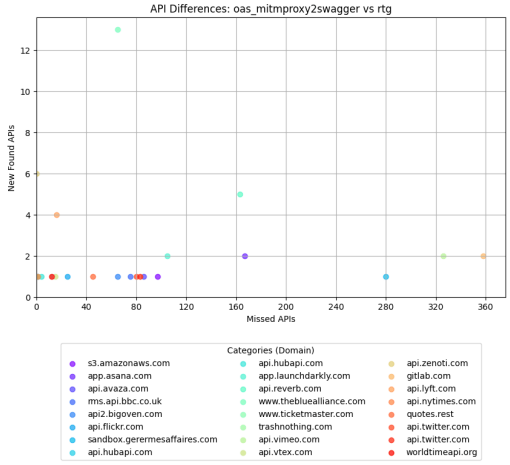


(c) RTG vs. openapi-diff

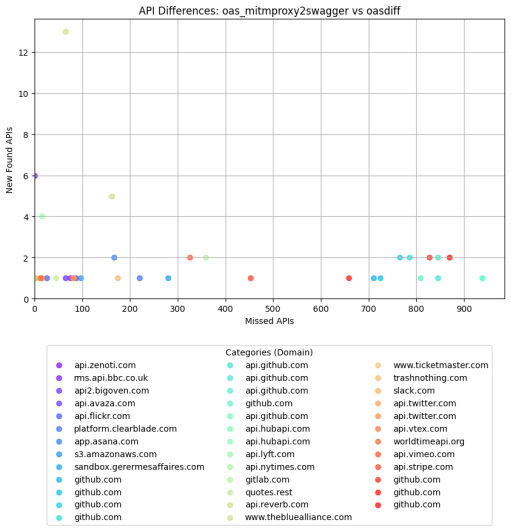


(d) RTG vs. api-schema-diff

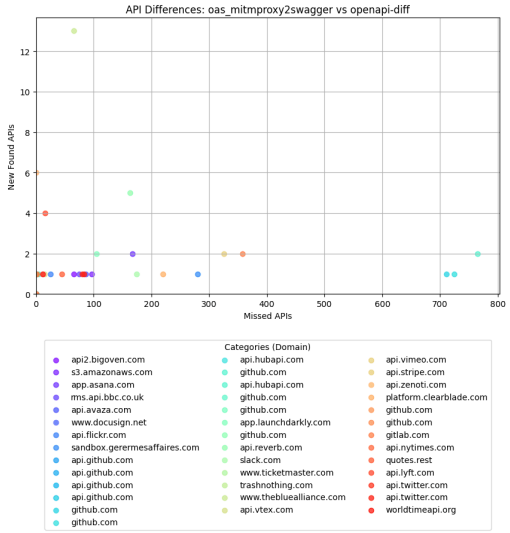
Figure 8.2: Scatter plots illustrating the correlation between Missed and New APIs generated by the RTG inference tool, evaluated across four different comparators.



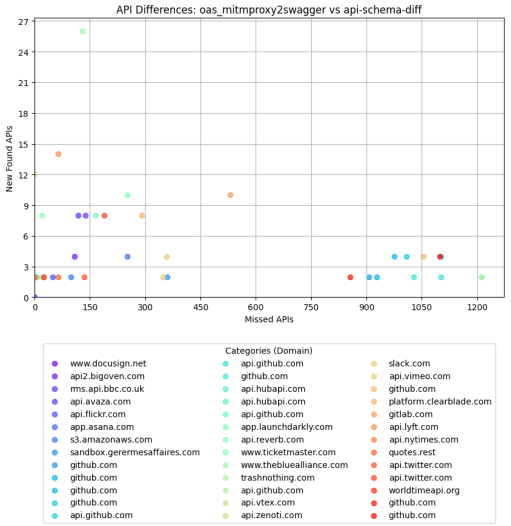
(a) mitmproxy2swagger vs. RTG Comparator



(b) mitmproxy2swagger vs. oasdiff

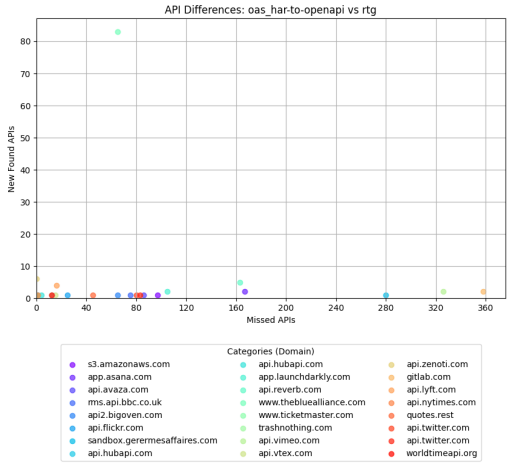


(c) mitmproxy2swagger vs. openapi-diff

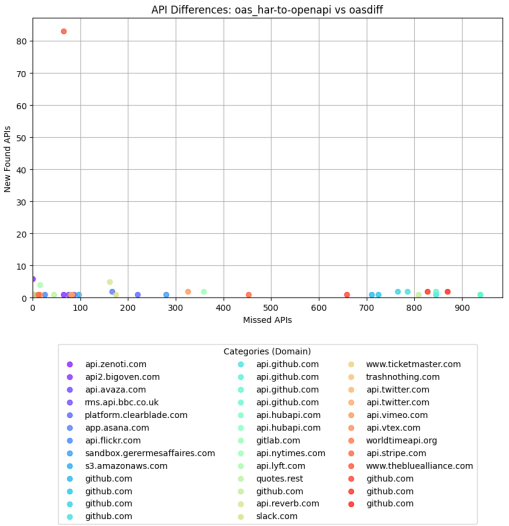


(d) mitmproxy2swagger vs. api-schema-diff

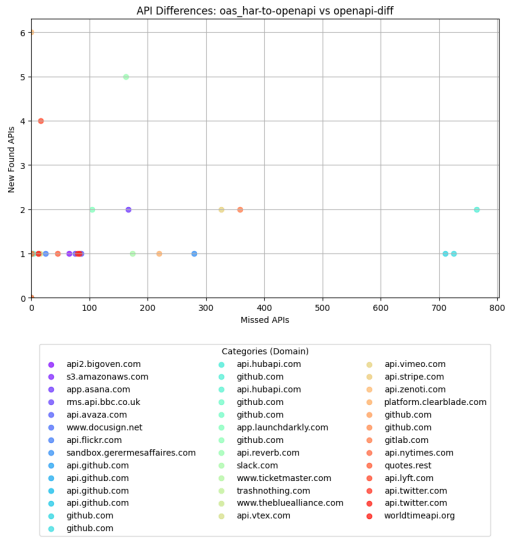
Figure 8.3: Scatter plots illustrating the correlation between Missed and New APIs generated by the mitmproxy2swagger inference tool, evaluated across four different comparators.



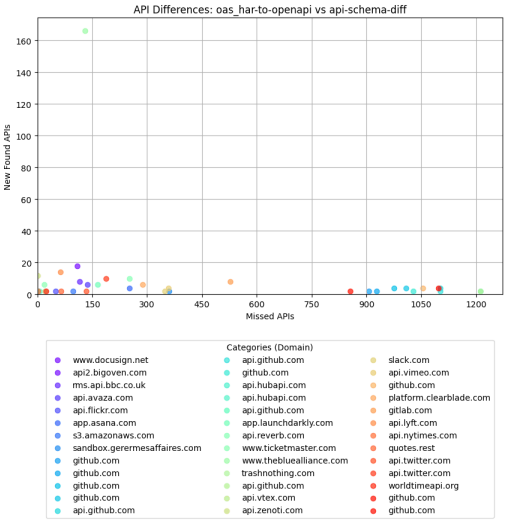
(a) har-to-openapi vs. RTG Comparator



(b) har-to-openapi vs. oasdiff



(c) har-to-openapi vs. openapi-diff



(d) har-to-openapi vs. api-schema-diff

Figure 8.4: Scatter plots illustrating the correlation between Missed and New APIs generated by the har-to-openapi inference tool, evaluated across four different comparators.

8.5 Performance and Execution Efficiency

The performance of comparison tools is related to their time-complexity and crashless working. The results of the experiment demonstrate the most unstable comparator is **RTG** due to missed output and results because of crashing in the middle of the converting process. On the other hand **oasdiff** has almost 100 percentage stability to finish comparison process.

Table 8.7: Ranking of Comparator Tools based on Execution Stability.

Rank	Comparator Tool	Total Executions	Success	Failed	Success Rate
1	openapi-diff	110	110	0	100.0%
2	api-schema-diff	110	107	3	97.3%
3	oasdiff	110	104	6	94.5%
4	RestTestGen (RTG)	110	65	45	59.1%

Furthermore, study this information from crashless aspects demonstrates which Conversion and Comparison tools have the most harmony with each other.

Table 8.8: Ranking of Pipeline Combinations by Execution Stability.

Rank	Inference Tool	Comparator	Matches	Success	Failed	Rate (%)
1	har-to-openapi	openapi-diff	40	40	0	100.0%
2	mitmproxy2swagger	openapi-diff	40	40	0	100.0%
3	RestTestGen (RTG)	openapi-diff	30	30	0	100.0%
4	har-to-openapi	api-schema-diff	40	39	1	97.5%
5	mitmproxy2swagger	api-schema-diff	40	39	1	97.5%
6	RestTestGen (RTG)	api-schema-diff	30	29	1	96.7%
7	har-to-openapi	oasdiff	40	38	2	95.0%
8	mitmproxy2swagger	oasdiff	40	38	2	95.0%
9	RestTestGen (RTG)	oasdiff	30	28	2	93.3%
10	har-to-openapi	RestTestGen (RTG)	40	24	16	60.0%
11	mitmproxy2swagger	RestTestGen (RTG)	40	24	16	60.0%
12	RestTestGen (RTG)	RestTestGen (RTG)	30	17	13	56.7%

Another side of the performance is time-consuming. Due to different algorithms of each tool, they finish the process within different duration. The

results shows RTG takes significantly more time to processing with respect to other tools.

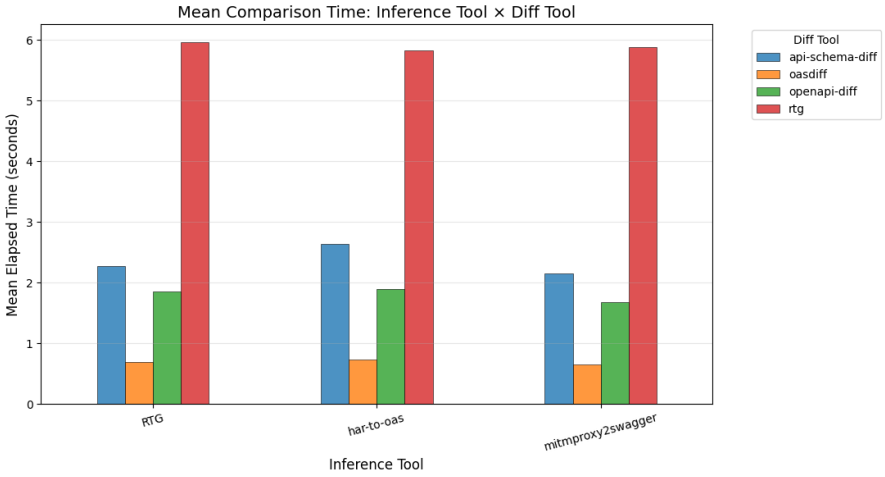


Figure 8.5: Mean comparison time categorized by Inference Tool and Diff Tool.

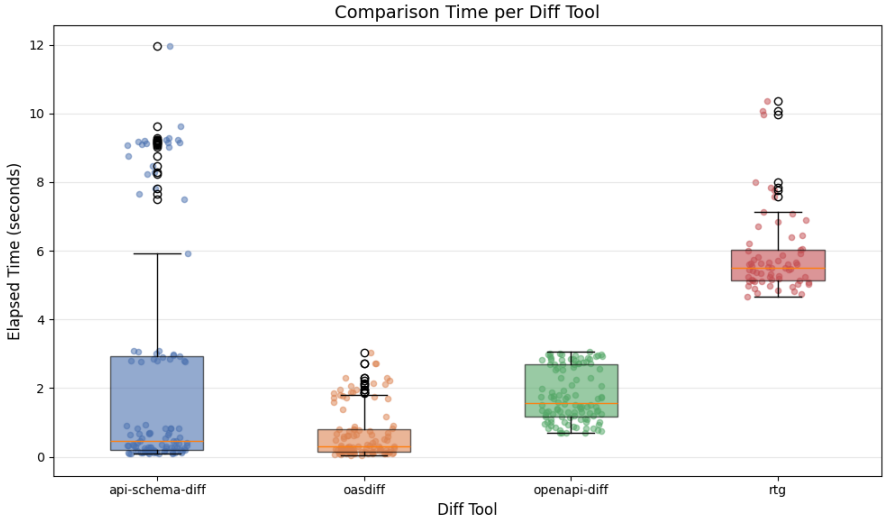


Figure 8.6: Distribution of comparison time per diff tool.

8.6 Discussion and Key Findings

The results of this study demonstrate that inference tools have different behavior in converting captured HAR files to OAS files. The most robust tool for this task is **Mitmproxy2swagger** with a significant gap with respect to other tools. On the other hand, for comparing generated OAS files with the ground truth OAS files, **RestTestGen (RTG)** is the most robust tool to generating more records from comparison operation with a large gap with respect to other tools. Furthermore, this study shows that the analysis of the generated OAS files is possible to do with several aspects to understand different dimensions of the generated OAS files.

Chapter 9

Conclusion and Future Work

This chapter is dedicated to summarizing the key findings of the study and the entirety of this project and also discussing potential future directions for research and development in the area of API inference and testing. In particular, there are some aspects which are not covered in this project and they can be considered as future directions for research.

9.1 Summary of Findings

At the end of this project, there are several key findings that can be highlighted. Some of the most important findings are related to APTS implementation. The results of this project demonstrate that the implemented APTS for Android applications is a practical and effective approach for capturing API interactions and generating OAS files. By considering many important aspects of the implementation, such as containerization, automation, and scalability, the APTS is able to handle a wide range of applications and capture their API interactions in a reliable manner.

Another important finding is related to the robustness and performance of different inference and comparison tools. The results of the study show that there are significant differences in the behavior of different tools, both in terms of their ability to successfully process OAS files and in terms of the number of records they generate during comparison.

Furthermore, a crucial observation pertains to the OAS inference and documentation. The results of the study demonstrate that the published and official OAS files for APIs are not always complete and accurate, and they will be obsolete after a while (API Drift). In fact, developers are adding new APIs and features to their services continuously without updating the official OAS

files.

In conclusion, this project provides a comprehensive and practical solution for capturing API interactions and generating OAS files for Android applications, and also provides a detailed analysis of the robustness and performance of different inference and comparison tools.

9.2 Future Directions

In this section, there are some potential future work directions that can be considered for further research and development in the area of API inference and testing. Fundamentally, this project has provided a comprehensive approach to have an effective APTS for Android applications. However, there are some aspects that can be further explored and improved in future work.

9.2.1 Support for Non-HTTP Protocols

This project has focused on capturing API interactions that use the HTTP protocol. Despite the fact that HTTP is the most widely used protocol for APIs, and the discovered information and studies about HTTP APIs are more than other protocols, there are still many APIs that use other protocols, such as gRPC [106], WebSockets [115], and MQTT [116]. For future work, it would be interesting to extend the APTS to support capturing API interactions that use these non-HTTP protocols, and also to generate appropriate specifications for these protocols. The future work can explore among lower layers of the network stack to capture API interactions, and using some related tools such as pcap [117] or tools based on eBPF [118] to capture API interactions at the network level, and then using some protocol-specific parsers to extract the relevant information and generate specifications.

9.2.2 Advanced AI-driven UI Exploration

In this project, the UI exploration component of the APTS is based on a simple random exploration strategy by Monkey tool that was a built-in tool in Android SDK. However, there are many advanced techniques for UI exploration that can be used to improve the coverage and effectiveness of the APTS. For future work, it would be interesting to explore the use of AI-driven techniques for UI exploration, such as reinforcement learning [119] or evolutionary algorithms [120]. These techniques can potentially learn more effective exploration

strategies over time, and can adapt to the specific characteristics of different applications. In recent years, many AI-driven tools have been proposed for UI exploration, such as DroidBot, AutoDroid and Humanoid which are practical for gathering more apis from applications due to AI driven exploration and passing from logging screens and other screens which are not possible to pass by simple random exploration.

9.2.3 Integration with Static Analysis Tools

In this project, the APTS is primarily focused on dynamic analysis of API interactions. At the end of the APTS results, it is observable that there are some applications that APTS is not able to capture any API interactions from, and it cannot cross their security barriers. For future work, it would be interesting to explore the integration of the APTS with static analysis [20] tools to break the security barriers of applications, and especially **SSL certificate pinning [60] bypass** by tools such as Frida [121] to disable application security scripts. However, it is important to consider the ethical implications of such an approach, and to ensure that it is used in a responsible and legal manner. Furthermore, due to the obfuscation of the source code of Android applications, especially in release versions, it could be a challenging task.

Bibliography

[1] Wikipedia contributors. "Computer security software — Wikipedia, the free encyclopedia." Accessed: Apr. 19, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Computer_security_software.

[2] Wikipedia contributors. "Obfuscation (software) — Wikipedia, the free encyclopedia." Accessed: Apr. 19, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Obfuscation_(software).

[3] Wikipedia contributors. "Denial-of-service attack — Wikipedia, the free encyclopedia." Accessed: May 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Denial-of-service_attack.

[4] Wikipedia contributors. "Api — Wikipedia, the free encyclopedia." Accessed: Mar. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/API.

[5] Wikipedia contributors. "Man-in-the-middle attack — Wikipedia, the free encyclopedia." Accessed: Mar. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Man-in-the-middle_attack.

[6] Google LLC. "Android - the platform pushing what's possible." Accessed: Mar. 22, 2026. (2026), [Online]. Available: https://www.android.com/.

[7] Wikipedia contributors. "Data set — Wikipedia, the free encyclopedia." Accessed: Apr. 18, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Data_set.

[8] Wikipedia contributors. "Noise — Wikipedia, the free encyclopedia." Accessed: Apr. 18, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Noise.

[9] Android Developers. "Insecure api usage." Accessed: Apr. 19, 2026. (2024), [Online]. Available: https://developer.android.com/privacy-and-security/risks/insecure-api-usage.

[10] Wikipedia contributors. "Malware — Wikipedia, the free encyclopedia." Accessed: Apr. 18, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Malware.

[11] Y. He *et al.*, "A systematic study of android non-sdk (hidden) service api security," *arXiv preprint arXiv:2203.09374*, 2022.

[12] European Union. "General data protection regulation (gdpr) – official legal text." Accessed: Apr. 20, 2026. (2026), [Online]. Available: https://gdpr.eu/.

[13] Wikipedia contributors. "Zero-day vulnerability — Wikipedia, the free encyclopedia." Accessed: Apr. 20, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Zero-day_vulnerability.

[14] Wikipedia contributors. "Operating system — Wikipedia, the free encyclopedia." Accessed: Mar. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Operating_system.

[15] Wikipedia contributors. "Usage share of operating systems — Wikipedia, the free encyclopedia." Accessed: Mar. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Usage_share_of_operating_systems.

[16] B. Nuseibeh and D. Rombach, "Automated software engineering: An introduction," *ERCIM News*, vol. 58, 2004. [Online]. Available: https://www.ercim.eu/publication/Ercim_News/enw58/intro.html.

[17] C. Bates. "Why repeatability matters in testing." Accessed: Apr. 20, 2026. (2026), [Online]. Available: https://breakingac.com/news/2026/jan/16/why-repeatability-matters-in-testing/.

[18] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16, ACM, 2016, pp. 468–471. DOI: 10.1145/2901739.2903508.

[19] Wikipedia contributors. "Openapi specification — Wikipedia, the free encyclopedia." Accessed: Apr. 20, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/OpenAPI_Specification.

[20] Wikipedia contributors. "Static program analysis — Wikipedia, the free encyclopedia." Accessed: May 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Static_program_analysis.

[21] Wikipedia contributors. “Apk (file format) — Wikipedia, the free encyclopedia.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Apk_(file_format).

[22] Wikipedia contributors. “Dynamic program analysis — Wikipedia, the free encyclopedia.” Accessed: May 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Dynamic_program_analysis.

[23] Android Developers. “Projects overview – project view.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://developer.android.com/studio/projects#ProjectView.

[24] Wikipedia contributors. “Url — Wikipedia, the free encyclopedia.” Accessed: May 1, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/URL.

[25] Wikipedia contributors. “Rest — Wikipedia, the free encyclopedia.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/REST.

[26] Wikipedia contributors. “Proguard — Wikipedia, the free encyclopedia.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/ProGuard.

[27] Android Developers. “Enable app optimization.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://developer.android.com/topic/performance/app-optimization/enable-app-optimization.

[28] Android Developers. “Analyze your build with apk analyzer.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://developer.android.com/studio/debug/apk-analyzer.

[29] Android Developers. “Android studio.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://developer.android.com/studio.

[30] Google LLC. “Run apps on the Android Emulator.” Accessed: Mar. 22, 2026. (2026), [Online]. Available: https://developer.android.com/studio/run/emulator.

[31] ScienceDirect. “Victim computer - an overview.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://www.sciencedirect.com/topics/computer-science/victim-computer.

[32] Android Developers. “Ui/application exerciser monkey.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://developer.android.com/studio/test/other-testing-tools/monkey.

[33] Wikipedia contributors. “Android sdk — Wikipedia, the free encyclopedia.” Accessed: Apr. 21, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Android_SDK.

[34] Wikipedia contributors. “Https — Wikipedia, the free encyclopedia.” Accessed: Apr. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/HTTPS.

[35] Wikipedia contributors. “Transport layer security — Wikipedia, the free encyclopedia.” Accessed: Apr. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security.

[36] Wikipedia contributors. “Http — Wikipedia, the free encyclopedia.” Accessed: Apr. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/HTTP.

[37] Wikipedia contributors. “Transport layer security (ssl 1.0, 2.0, and 3.0) — Wikipedia, the free encyclopedia.” Accessed: Apr. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security#SSL_1.0,_2.0,_and_3.0.

[38] Wikipedia contributors. “Tls termination proxy — Wikipedia, the free encyclopedia.” Accessed: Apr. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/TLS_termination_proxy.

[39] The mitmproxy project. “Mitmproxy - an interactive HTTPS proxy.” Accessed: Mar. 22, 2026. (2026), [Online]. Available: https://www.mitmproxy.org/.

[40] mitmproxy contributors. “Certificates - mitmproxy documentation.” Accessed: Apr. 22, 2026. (2026), [Online]. Available: https://docs.mitmproxy.org/stable/concepts/certificates.

[41] Wikipedia contributors. “Rooting (android) — Wikipedia, the free encyclopedia.” Accessed: Apr. 22, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Rooting_(Android).

[42] Wikipedia contributors. “Har (file format) — Wikipedia, the free encyclopedia.” Accessed: Apr. 23, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/HAR_(file_format).

[43] JSON.org. “Introducing json.” Accessed: Apr. 23, 2026. (2026), [Online]. Available: https://www.json.org/json-en.html.

[44] HTTP Archive. “Http archive: Tracking how the web is built.” Accessed: Apr. 23, 2026. (2026), [Online]. Available: https://httparchive.org/.

[45] Wikipedia contributors. “Black-box testing — Wikipedia, the free encyclopedia.” Accessed: May 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Black-box_testing.

[46] Postman Team. “What is an api endpoint?” Accessed: May 1, 2026. (2026), [Online]. Available: https://blog.postman.com/what-is-an-api-endpoint/.

[47] Wikipedia contributors. “Cluster analysis — Wikipedia, the free encyclopedia.” Accessed: Apr. 23, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Cluster_analysis.

[48] SmartBear Software. “About swagger specification.” Accessed: Apr. 23, 2026. (2026), [Online]. Available: https://swagger.io/docs/specification/about/.

[49] Wikipedia contributors. “Json — Wikipedia, the free encyclopedia.” Accessed: Apr. 23, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/JSON.

[50] Wikipedia contributors. “Yaml — Wikipedia, the free encyclopedia.” Accessed: Apr. 23, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/YAML.

[51] OpenAPI Initiative. “Openapi specification.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://swagger.io/specification/.

[52] Wikipedia contributors. “Packet analyzer — Wikipedia, the free encyclopedia.” Accessed: Apr. 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Packet_analyzer.

[53] Wikipedia contributors. “Internet protocol suite — Wikipedia, the free encyclopedia.” Accessed: Apr. 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Internet_protocol_suite.

[54] Wireshark Foundation. “Wireshark.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://www.wireshark.org/.

[55] The TCPdump Group. “Tcpdump and libpcap.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://www.tcpdump.org/.

[56] Wikipedia contributors. “Root certificate — Wikipedia, the free encyclopedia.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Root_certificate.

[57] Wikipedia contributors. “Firewall (computing) — Wikipedia, the free encyclopedia.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Firewall_(computing).

[58] Progress Telerik. “Fiddler web debugging proxy.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://www.telerik.com/fiddler.

[59] XK72. “Charles web debugging proxy.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://www.charlesproxy.com/.

[60] Wikipedia contributors. “Http public key pinning — Wikipedia, the free encyclopedia.” Accessed: May 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Certificate_pinning.

[61] alufers. “Mitmproxy2swagger: A tool for automatically converting mitmproxy captures to openapi 3.0 specifications.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://github.com/alufers/mitmproxy2swagger.

[62] jonluca. “Har-to-openapi: Generate openapi specification from har files.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://github.com/jonluca/har-to-openapi.

[63] dcarr178. “Har2openapi: Convert har files to openapi specifications.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://github.com/dcarr178/har2openapi.

[64] Wikipedia contributors. “Command-line interface — Wikipedia, the free encyclopedia.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Command-line_interface.

[65] Software Engineering Lab - University of Verona (SeUniVr). “Resttestgen: Automated black-box testing of restful apis.” Accessed: Apr. 26, 2026. (2026), [Online]. Available: https://github.com/SeUniVr/RestTestGen.

[66] Wikipedia contributors. “Deep reinforcement learning — Wikipedia, the free encyclopedia.” Accessed: May 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Deep_reinforcement_learning.

[67] ICST 2020 Organization. “13th IEEE international conference on software testing, validation and verification (ICST 2020).” Accessed: Apr. 26, 2026. (2020), [Online]. Available: https://conf.researchr.org/home/icst-2020.

[68] E. Viglianisi, M. Dall'Ora, and M. Ceccato, "RESTTESTGEN: Automated black-box testing of RESTful apis," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, IEEE, 2020, pp. 142–152. [Online]. Available: https://ieeexplore.ieee.org/document/9159077/.

[69] GitHub. "Containerization." Accessed: Apr. 27, 2026. (2026), [Online]. Available: https://github.com/resources/articles/containerization.

[70] Python Software Foundation. "Welcome to python.org." Accessed: Apr. 27, 2026. (2026), [Online]. Available: https://www.python.org/.

[71] Android Developers. "Configure the app module: Set the namespace." Accessed: Apr. 27, 2026. (2026), [Online]. Available: https://developer.android.com/build/configure-app-module#set-namespace.

[72] Wikipedia contributors. "Emulator — Wikipedia, the free encyclopedia." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Emulator.

[73] G. Rodolà. "Psutil (python system and process utilities)." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://github.com/giampaolo/psutil.

[74] Androguard contributors. "Androguard: Reverse engineering, malware and goodware analysis of android applications." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://github.com/androguard/androguard.

[75] Docker Inc. "Docker official website." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://www.docker.com/.

[76] Docker Inc. "Docker hub container image library." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://hub.docker.com/.

[77] Podman contributors. "Podman: A tool for managing OCI containers and pods." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://podman.io/.

[78] Docker Inc. "Dockerfile reference." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://docs.docker.com/reference/dockerfile/.

[79] H. Husain, K. Marzuki, C. M. Lauw, and L. Z. A. Mardedi, "Analysis and implementation of comparison between podman and docker in container management," *International Journal of Electronics and Communications Systems*, vol. 3, no. 2, pp. 57–67, Dec. 2023. DOI: 10.24042/ijecs.v3i2.19860.

[80] S. Kanthed, "Docker vs. podman: Architecture differences and their impact on modern workflows," *International Journal of Multidisciplinary Research and Growth Evaluation*, vol. 4, no. 4, pp. 1139–1146, Jan. 2023. DOI: 10.54660/.IJMRGE.2023.4.4.1139-1146.

[81] Wikipedia contributors. "Linux — Wikipedia, the free encyclopedia." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Linux.

[82] The Debian Project. "Debian – the universal operating system." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://www.debian.org/.

[83] Canonical Ltd. "Enterprise open source and linux — ubuntu." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://ubuntu.com/.

[84] QEMU Project. "Qemu." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://www.qemu.org/.

[85] Wikipedia contributors. "Java development kit — Wikipedia, the free encyclopedia." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Java_Development_Kit.

[86] Android Developers. "Android studio and sdk tools." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://developer.android.com/tools.

[87] Docker Inc. and contributors. "Docker sdk for python documentation." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://docker-py.readthedocs.io/en/stable/.

[88] Podman contributors. "Podman sdk for python documentation." Accessed: Apr. 29, 2026. (2026), [Online]. Available: https://podman-py.readthedocs.io/en/latest/.

[89] tmux contributors. "Tmux wiki." Accessed: Apr. 30, 2026. (2026), [Online]. Available: https://github.com/tmux/tmux/wiki.

[90] Android Developers. “Start the emulator from the command line.” Accessed: Apr. 30, 2026. (2026), [Online]. Available: https://developer.android.com/studio/run/emulator-commandline#startup-options.

[91] Wikipedia contributors. “Privacy-enhanced mail — Wikipedia, the free encyclopedia.” Accessed: Apr. 30, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Privacy-Enhanced_Mail.

[92] mitmproxyf contributors. “Install system trusted ca on android.” Accessed: Apr. 30, 2026. (2026), [Online]. Available: https://docs.mitmproxy.org/stable/howto/install-system-trusted-ca-android/#3-insert-certificate-into-system-certificate-store.

[93] Android Developers. “Android debug bridge (adb).” Accessed: May 25, 2026. (2026), [Online]. Available: https://developer.android.com/tools/adb.

[94] Android Developers. “Ui/application exerciser monkey - command options reference.” Accessed: Apr. 30, 2026. (2026), [Online]. Available: https://developer.android.com/studio/test/other-testing-tools/monkey#command-options-reference.

[95] Mitmproxy Project. “Mitmproxy Documentation: mitmdump.” Accessed: Apr. 10, 2026. (2026), [Online]. Available: https://docs.mitmproxy.org/stable/#mitmdump.

[96] mitmproxy contributors. “Options - mitmproxy documentation.” Accessed: Apr. 30, 2026. (2026), [Online]. Available: https://docs.mitmproxy.org/stable/concepts/options/#available-options.

[97] Wikipedia contributors. “Comma-separated values — Wikipedia, the free encyclopedia.” Accessed: May 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Comma-separated_values.

[98] Wikipedia contributors. “Web typography — Wikipedia, the free encyclopedia.” Accessed: Apr. 13, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/Web_typography.

[99] ReqBin. “What is the correct json content type?” Accessed: May 1, 2026. (2026), [Online]. Available: https://reqbin.com/req/abghm4zf/json-content-type.

[100] OpenAPITools Contributors, *Openapi-diff: A utility for comparing two openapi specifications*, version 2.0, Accessed: May 6, 2026, 2026. [Online]. Available: https://github.com/OpenAPITools/openapi-diff.

[101] Oasdiff Contributors, *Oasdiff: A specialized tool for diffing openapi specifications*, Accessed: May 6, 2026, 2026. [Online]. Available: https://github.com/oasdiff/oasdiff.

[102] T. Lazer, *Schema-diff: A tool for comparing and identifying differences between json schemas*, Accessed: May 7, 2026, 2026. [Online]. Available: https://github.com/teolzr/schema-diff.

[103] OpenAPITools Contributors. "Openapi-diff Docker Image." Official Docker Hub repository. Accessed: May 6, 2026. (2026), [Online]. Available: https://hub.docker.com/r/openapitools/openapi-diff/.

[104] The Go Authors. "The go programming language." Accessed: May 25, 2026. (2026), [Online]. Available: https://go.dev/.

[105] The GraphQL Foundation. "Graphql: A query language for apis." Accessed: May 9, 2026. (2026), [Online]. Available: https://graphql.org/.

[106] The gRPC Authors. "Grpc: A high-performance, open source universal rpc framework." Accessed: May 9, 2026. (2026), [Online]. Available: https://grpc.io/.

[107] K. Sandoval. "What is API Drift, and What Can You Do About It?" Accessed: May 13, 2026, Nordic APIs. (2024), [Online]. Available: https://nordicapis.com/what-is-api-drift-and-what-can-you-do-about-it/.

[108] APIs.guru Contributors. "Apis.guru - wikipedia for web apis." Accessed: May 13, 2026. (2026), [Online]. Available: https://apis.guru/.

[109] APIs.guru Contributors, *Apis.guru GitHub Organization: A machine-readable wikipedia for web apis*, Accessed: May 13, 2026, 2026. [Online]. Available: https://github.com/APIs-guru.

[110] Google LLC. "Firebase." Accessed: May 25, 2026. (2026), [Online]. Available: https://firebase.google.com/.

[111] Wikipedia contributors. "Oauth — Wikipedia, the free encyclopedia." Accessed: May 25, 2026. (2026), [Online]. Available: https://en.wikipedia.org/wiki/OAuth.

[112] N. Sakimura, J. Bradley, and N. Agarwal. "RFC 7636: Proof key for code exchange by OAuth public clients." Accessed: May 25, 2026. (2015), [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7636.

[113] University of Luxembourg. "Androzoo: Growing collection of android applications." Accessed: May 14, 2026. (2026), [Online]. Available: https://androzoo.uni.lu/.

[114] Google LLC. "Android apps on google play." Accessed: May 25, 2026. (2026), [Online]. Available: https://play.google.com/store/apps.

[115] Wikipedia contributors. "Websocket — Wikipedia, the free encyclopedia." (2026), [Online]. Available: https://en.wikipedia.org/wiki/WebSocket (visited on 05/16/2026).

[116] MQTT.org. "Mqtt - the standard for iot messaging." (2026), [Online]. Available: https://mqtt.org/ (visited on 05/16/2026).

[117] Wikipedia contributors. "Pcap — Wikipedia, the free encyclopedia." (2026), [Online]. Available: https://en.wikipedia.org/wiki/Pcap (visited on 05/16/2026).

[118] eBPF Foundation. "Ebpf - introduction, tutorials & community resources." (2026), [Online]. Available: https://ebpf.io/ (visited on 05/16/2026).

[119] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html (visited on 05/16/2026).

[120] Wikipedia contributors. "Evolutionary algorithm — Wikipedia, the free encyclopedia." (2026), [Online]. Available: https://en.wikipedia.org/wiki/Evolutionary_algorithm (visited on 05/16/2026).

[121] Frida Developers. "Frida - android documentation." (2026), [Online]. Available: https://frida.re/docs/android/ (visited on 05/16/2026).