**Requirements Analysis Document (RAD)**

**Prepared for**

*Wopa - Intelligent Chat Safeguarder*

**Prepared by**

**Team 2**

**Yongcheng Liu, Shucheng Fang**

***12/12/2024***

# 1 Introduction

## 1.1 Purpose of the system

Wopa, the *Intelligent Chat Safeguarder*, is a mobile security solution designed to provide real-time protection against the growing risks posed by mobile messaging platforms. Users often share links, files, and other content without fully verifying their safety, which can lead to privacy breaches, phishing, or malware attacks. Traditional antivirus solutions and firewalls offer static analysis but often fail to catch dynamic threats.

Wopa aims to bridge this gap by running in the background on mobile devices, continuously monitoring messages for suspicious content. By analyzing the legitimacy of files and links through an AI-powered sandbox and behavior simulations, Wopa helps users prevent potential security risks in real time, all while ensuring seamless integration into the mobile user experience.

## 1.2 References

## 

| **Date** | **Type** | **Title** |
| --- | --- | --- |
| 09/05 | Document | Project Proposal |
| 09/09 | Meeting | Core Functionality Discussion Meeting |

## 

## 1.3 Scope of the system

Wopa is designed to protect mobile users from security threats associated with real-time communication. It runs in the background, similar to antivirus software or firewalls, constantly monitoring for potential risks in the messages users send and receive. The system checks suspicious content such as links and files, running them in a secure sandbox environment to assess their legitimacy.

Wopa is user-friendly and integrated into the mobile platform, providing real-time security feedback without disrupting the user experience. Users will receive automatic notifications if a threat is detected, but the tool works largely without requiring active input from the user.

## 1.4 Core System Functionalities

Wopa’s main functionalities include:

* **Secure Sandbox:** A controlled environment where suspicious files and links are safely executed and analyzed to prevent harm to the user’s device.
* **AI-Powered Log Analysis:** The system leverages AI to perform real-time analysis of logs, such as system calls, API interactions, and network traffic, to detect both static and dynamic threats.
* **Visual-Based Simulation:** Wopa simulates user interactions with apps to assess potential privacy violations or behavior-based threats that may not be caught by static analysis.
* **Continuous Monitoring:** The system continuously monitors activities on the device, ensuring that messages, links, and files are analyzed for risks like phishing, malware, or privacy invasions.

## 1.5 Objectives and Success Criteria of the Project

The success of Wopa will be measured by its ability to provide reliable, real-time threat detection and prevent potential attacks in mobile messaging environments. Key success criteria include:

* The system’s ability to detect dynamic and behavior-based threats with a high degree of accuracy.
* User satisfaction with the tool’s ease of use, performance, and integration into everyday mobile app usage.
* A 30-second or less response time for initial assessments, balancing speed and reliability.
* The tool’s ability to identify and report privacy violations in at least 85% of tested Android apps.

# 

# 2 Current System

## 2.1 Existing System

Currently, mobile security solutions primarily rely on **static analysis techniques** to detect malicious files or links. These methods focus on examining the binary code of suspicious programs and identifying patterns indicative of malicious behavior. While static analysis has been foundational in malware detection, it often fails to detect **dynamic threats** that require analysis during execution. Despite being resource-intensive, static methods remain effective in specific scenarios and are frequently combined with dynamic techniques to enhance accuracy.

The introduction of sandbox technology has shown significant promise. Costa et al. [[1]](#rdmdpfb77q92) demonstrated that sandbox environments allow security tools to detect significantly more malware in tested datasets. Sandboxes create **isolated environments** where suspicious programs or files are safely executed and monitored. Logs of system calls, API interactions, and network traffic are generated during execution, providing critical insights into potentially malicious behavior. Many modern sandboxes utilize **machine learning models** or pattern-based approaches for threat detection. Examples such as **DroidXP**, **CamoDroid**, and **DroidHook** enhance detection accuracy by offering features like **user simulation**, **device cloaking**, and **API hooking** to evade anti-sandbox techniques used by sophisticated malware.

However, these systems remain limited. They often require substantial computational resources and may fail to detect **real-time evolving threats**, especially when encountering new or unseen threats. The growing complexity of mobile environments further complicates the ability of static and dynamic analyses to capture the full spectrum of threats. The recent emergence of **zero-shot Large Language Models (LLMs)** offers a potential breakthrough. LLMs enable the creation of an **AI-driven security agent** that operates based on **natural language-based instructions** and **security insights**, mimicking the decision-making processes of security experts to detect and mitigate threats in real time.

**Current Tool Analysis**

Currently, most sandbox analysis are based on machine learning, which are composed of different machine learning algorithms and machine learning classifiers. In addition, there also exist some sandboxes which are mainly based on pattern, and machine learning is secondary. There are many machine learning algorithms that can be used to build sandbox models, such as Random Forest (RF), Neural Network (NN), Decision Tree (DT), k-Nearest Neighbor (k-NN), Naive Bayes (NB), and Support Vector Machine (SVM)[[2]](#x3m0igppgfo8).

Sandbox detection is dynamic, and it can be roughly divided into two steps: training and execution. But there are still many different ways to execute it. Some detection processes are divided into modules, and each functional module is used as a unit for detection, analysis and reporting. Some detection processes are based on the entire program as a unit (usually using machine learning models), running the entire program and listing the logs during the operation. Others start from the target, and stop the operation and give a report if suspicious points are found during the operation. Here’s the comparison between 3 advanced sandbox models: DroidXP[[3]](#9ouwvd5xfiuu), CamoDroid[[4]](#wzla1uv6yly7), and DroidHook[[5]](#vjzesvsnths0).

**DroidXP**

DroidXP is a benchmarking tool that operates through a simple Command Line Interface (CLI). It follows the pipes-and-filters architectural style and consists of three main components: i) **Instrumentation** defines and instruments APK files, enabling it to collect execution data, and conducts static analysis based on DroidFax[[7]](#gjvao9866z3h). ii) **Execution** installs the APKs on an Android emulator and runs each test case generation tool on the APKs within a defined period. iii) **Result Analysis** uses Logcat to collect logs capturing code coverage data and sensitive API access information, which are analyzed and outputted in the end.

**CamoDroid**

CamoDroid is an Android dynamic analysis framework designed to **cloak** the analysis environment from detection by malware. It simulates various device properties such as static attributes, network features, sensors, and user data to make the sandbox appear like a real device. By manipulating these properties, CamoDroid prevents malware from identifying the analysis environment. It also monitors sensitive API calls that require permissions to access critical resources, making it a more realistic environment. CamoDroid is highly extendable, enabling users to easily add new functionalities, such as cloaking additional APIs or monitoring new sensitive calls.

**DroidHook**

DroidHook is a dynamic instrumentation framework focusing on function **hooking** for Android applications. It allows security researchers and developers to intercept, monitor, and modify API calls at runtime. This tool is particularly useful for manipulating application behavior or analyzing how an application interacts with system resources. DroidHook emphasizes flexibility in hooking and altering API functions which makes it powerful for real-time API manipulation.

Although the implementation methods of the three sandbox models mentioned above are very different, DroidXP, CamoDroid and DroidHook all focus on responding to anti-sandbox methods[[6]](#y9ml8nj8h4h) generated by malware. Anti-sandbox methods are generated by malware as developers build sandboxes to prevent them. But there are still numerous differences among these 3 sandboxes. Here are their respective features:

Table. Features among DroidXP, CamoDroid, DroidHook

|  | User  Simulation | Device  Cloaking | API  Hooking | Extend-  able | Compu-  tation | Complex |
| --- | --- | --- | --- | --- | --- | --- |
| DroidXP | 1 | 0 | 0 | 0 | Medium | Medium |
| CamoDroid | 1 | 1 | 1 | 1 | High | High |
| DroidHook | 0 | 0 | 1 | 0 | Low | Low |

In conclusion,

i) **DroidXP:** The most **comprehensive** and **resource-intensive**, excels at identifying evasive behaviors in malware by simulating different environmental conditions to reveal hidden code paths;

ii) **CamoDroid:** The most **adaptable** and **complex**, focuses on cloaking analysis environments and simulating real device conditions to evade detection, with an emphasis on monitoring sensitive API calls;

iii) **DroidHook:** The most **lightweight** and **Intrusive**, is a hooking framework that allows analysts to intercept and modify API calls but lacks CamoDroid’s cloaking features.

## 2.2 Current Operations

At present, the **average user** typically relies on **antivirus software** or mobile security apps, which primarily employ static analysis to detect potential threats. These tools scan files, links, and other forms of content for **known malicious patterns** or signatures. While they are generally effective against well-documented malware, they often fail to detect newer, more sophisticated attacks that **adapt** or disguise their behavior during execution. This gap leaves users vulnerable to threats like **phishing**, **privacy violations**, and **malicious file executions**.

On the other hand, **dynamic analysis tools**, such as sandboxes, offer a more in-depth assessment by running suspicious files or links in a **controlled environment** and monitoring their behavior in real time. However, these tools are typically employed by **security researchers** or advanced users due to their complexity and resource requirements. For the average user, dynamic analysis is not widely accessible, making it difficult for them to proactively defend against real-time threats.

The divide between static and dynamic tools highlights a critical vulnerability: most users lack sufficient protection against **evolving threats**. Current solutions focus more on **post-attack identification** rather than **real-time prevention**, resulting in a **reactive** security approach rather than a **proactive** one. As attacks grow more sophisticated and harder to detect, the need for solutions that can handle both **static** and **dynamic threats** in real time, without compromising usability or performance, becomes increasingly apparent.

# 

# 3 Proposed System

## 3.1 Overview

Wopa is designed to address the security challenges faced by non-technical users who rely on mobile messaging platforms. Most users do not have the technical expertise to assess the safety of the files, links, and content they interact with daily. They simply want a solution that, once installed, works autonomously in the background, providing continuous protection without requiring any active involvement or complex configurations.

Wopa is a mobile security tool that runs seamlessly in the background, similar to antivirus software or firewalls, safeguarding users from both known and emerging threats in real time. Users should feel confident that the system is working silently and effectively to protect them from phishing attacks, file-based malware, and privacy violations, without disrupting their daily interactions on messaging platforms. They don’t want to worry about running security checks or handling complex security settings—they expect the tool to manage everything for them.

This system is solving the increasing complexity of security risks that existing static analysis or traditional antivirus solutions cannot fully address. Wopa introduces a new combination of AI-powered log analysis, dynamic behavior simulation, and sandbox technology. By utilizing zero-shot Large Language Models (LLMs) and advanced sandboxing, Wopa is able to detect not only traditional, static threats but also dynamic, behavior-based attacks that unfold during real-time use.

What previously couldn’t be done by traditional security tools—such as detecting threats that evolve during execution, preventing privacy violations in real time, and analyzing system calls and API interactions dynamically—can now be achieved through this system. Wopa’s unique points lie in its ability to adapt to new threats on the fly, simulate user interactions with apps to detect hidden malicious behaviors, and provide comprehensive, real-time security assessments, all without user intervention.

In summary, Wopa is designed for users who want to "install and forget"—a future where their mobile security is constantly monitored and protected, with minimal disruption and maximum reliability. The system’s goal is to provide proactive, real-time threat detection and prevention, effectively handling security risks that traditional methods cannot, while maintaining simplicity and ease of use for non-technical users.

## 3.2 Conceptual Model - User Scenarios

In this section, we provide examples of how **Wopa** will address typical real-world scenarios where mobile users face security risks in messaging platforms. These scenarios illustrate the system’s role in identifying and mitigating both dynamic and static threats in real time.

### **Scenario 1: Impersonation Attack via Malicious Link**

#### **Role: User (Non-technical individual using a messaging platform)**

#### **Action: Receives a message with a malicious link**

#### **Reason: Prevent account theft through phishing**

A user receives a message on a messaging platform from someone pretending to be their friend. The message contains a link and a friendly note:  
"Hey, check out this cool new mod for Steam! Click here!"

Unknown to the user, this is a **phishing attack** aimed at stealing their **Steam account** credentials. The link redirects to a fake Steam login page designed to steal the user's **Steam cookies**, which attackers can then use to gain unauthorized access to the account without needing a password.

**How Wopa addresses this:**

* Wopa identifies the suspicious link and automatically runs it in a secure sandbox environment.
* It analyzes the link’s destination and the web page it opens, recognizing that it is a phishing site mimicking Steam’s login page.
* The system then alerts the user in real time, warning them not to enter any credentials and flagging the link as dangerous, preventing the attacker from stealing their Steam account.

### **Scenario 2: Suspicious File Sent by Unknown Contact**

#### **Role: User (Non-technical individual receiving unsolicited messages)**

#### **Action: Receives a file from an unknown sender**

#### **Reason: Prevent malware infection and phishing**

A user receives a message from an unknown contact with a file named "The Way to Hack PayPal and Steal Money.pdf". The message teases the user to click on the file out of curiosity, potentially exposing them to malware. This type of file often contains malicious code or phishing attempts that attempt to trick users into giving away sensitive information or executing harmful software on their devices.

**How Wopa addresses this:**

* Wopa detects the suspicious nature of the file, noting the enticing filename designed to bait the user.
* It runs the file in a sandbox, checking its contents for malicious behavior, such as attempts to access sensitive device resources or execute hidden scripts.
* Upon detecting potential threats (e.g., unauthorized attempts to access financial data or manipulate system files), Wopa flags the file as dangerous and notifies the user before they can open it, thus protecting their device from harm.

## 3.3 Functional Model - Use Case Model

The following use cases outline the core functionalities that Wopa will provide to protect users from security threats in real-time communication environments. Each use case represents a key security feature, detailing how users interact with the system and how Wopa responds to different types of threats.

**Use Cases**

| **Name:** | Secure Sandbox **Select** File/Link |
| --- | --- |
| **Actor:** | User |
| **Entry**  **Conditions:** | None |
| **Flow of**  **Events:** | 1. User clicks the upper box to **request** to select the suspicious file ( or straightly drags the link/file into the box ) 2. System **responses** and opens file manager for user to select 3. User **confirms** the file to check 4. System **displays** the selected file/link in the upper box |
| **Exit**  **Conditions:** | The suspicious link/file has been executed, and feedback about it has been provided. |

| **Name:** | Secure Sandbox **Run** |
| --- | --- |
| **Actor:** | User |
| **Entry**  **Conditions:** | Suspicious link or file has been displayed in the sandbox. |
| **Flow of**  **Events:** | 1. User clicks button run to **request** for the analysis of the suspicious link/file 2. System **responses** to run the AI agent to analyze the suspicious link/file 3. System displays feedbacks about the suspicious in lower box |
| **Exit**  **Conditions:** | The suspicious link/file has been executed, and feedback about it has been displayed. |

Figure. Storyboard Diagram of Secure Sandbox

**Functional Use Case Diagram**

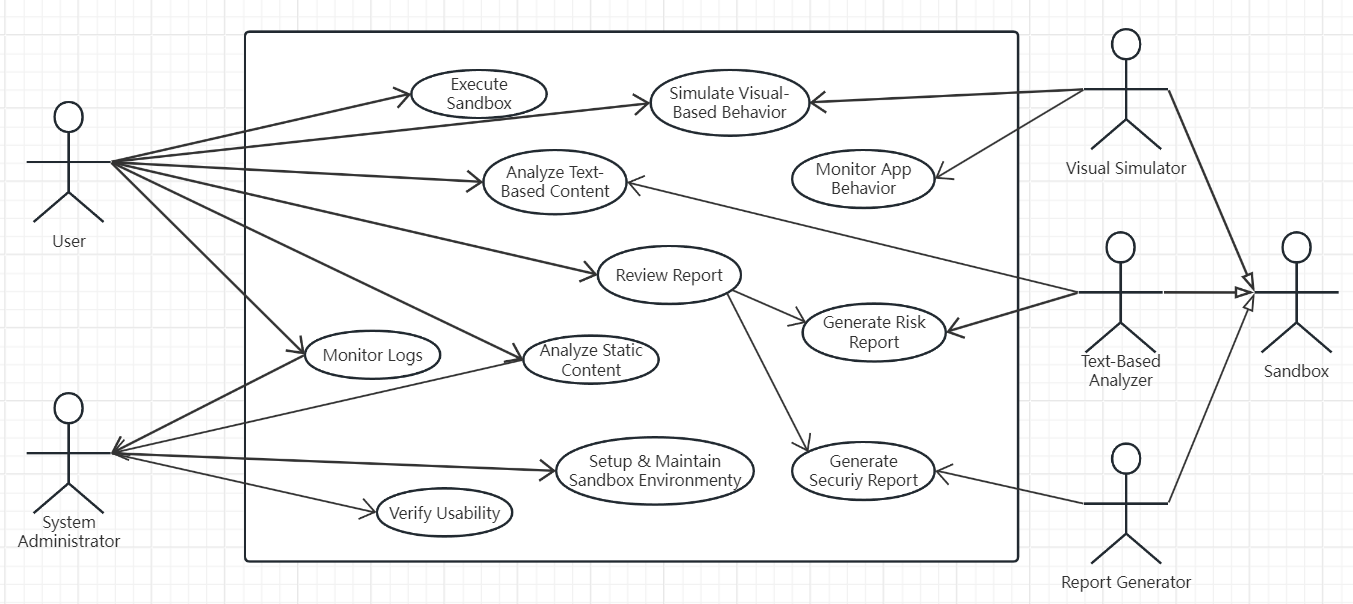


Figure. Functional Use Case Diagram

In the functional use case diagram (as seen in the second image provided), the system supports various actors:

* **User**: The non-technical individual interacting with the sandbox environment to analyze suspicious files or links.
* **System Administrator**: Oversees the maintenance of the sandbox environment, ensuring it functions properly and remains updated with the latest threat data.

Key functionalities include:

* **Executing Sandbox**: Users can initiate the sandbox environment to analyze suspicious files or links.
* **Simulating Visual-Based Behavior**: The system simulates user interactions to detect dynamic threats.
* **Analyzing Text-Based Content**: Using AI, the system analyzes logs, network traffic, and system calls to detect both static and dynamic threats.
* **Generating Security and Risk Reports**: Wopa generates detailed reports based on its analysis, providing users and administrators with actionable insights.

## 3.4 Analysis Model – Object Model

The object model below outlines the key classes and their relationships, representing the system's structure for supporting the functionalities described in the use cases. The classes encapsulate responsibilities for analyzing suspicious files or links, simulating user interactions, and generating reports based on the findings. Each class is designed to operate within the sandbox environment, ensuring safe execution and analysis of potentially malicious content.

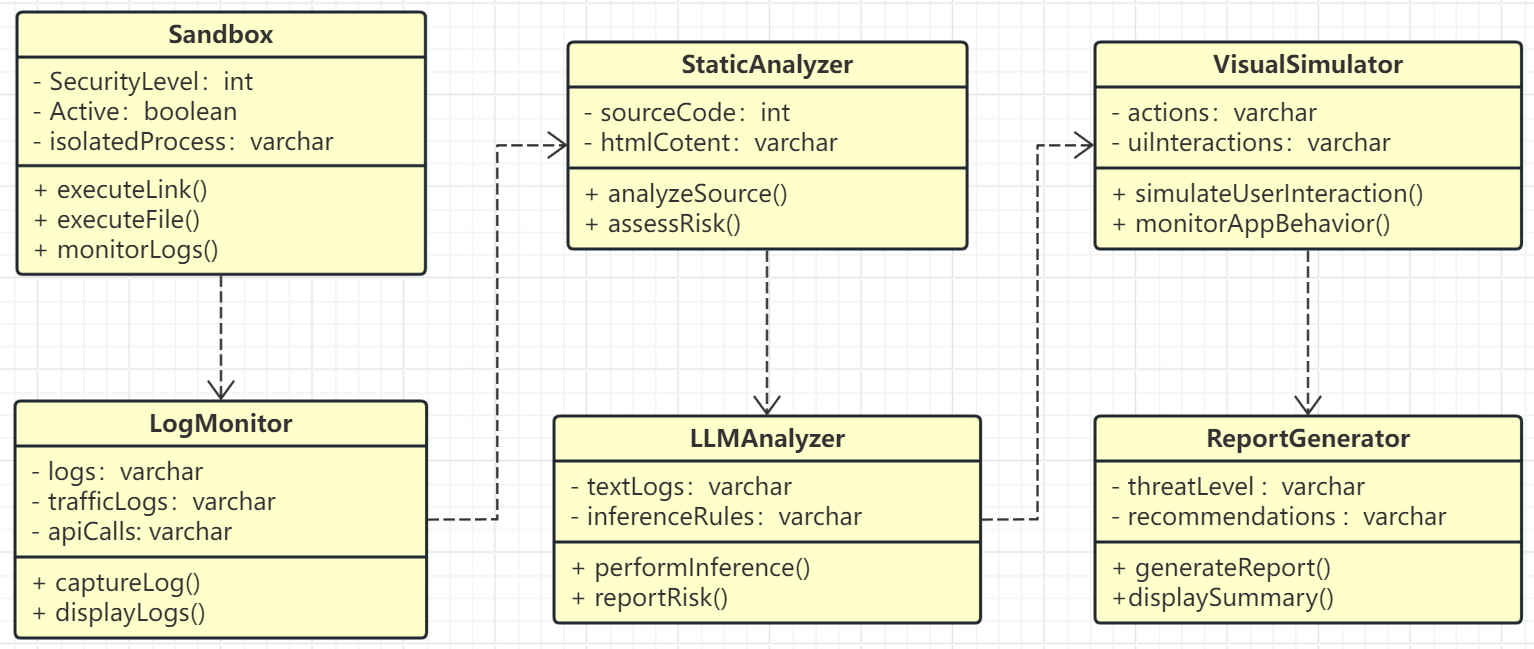


Figure. WOPA UML Diagram

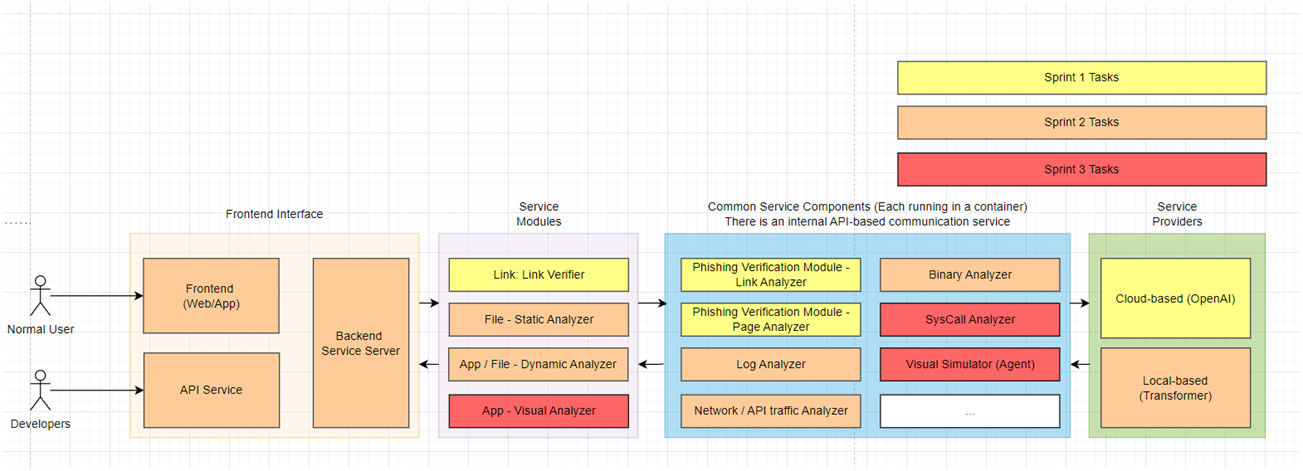


Figure. WOPA Architecture

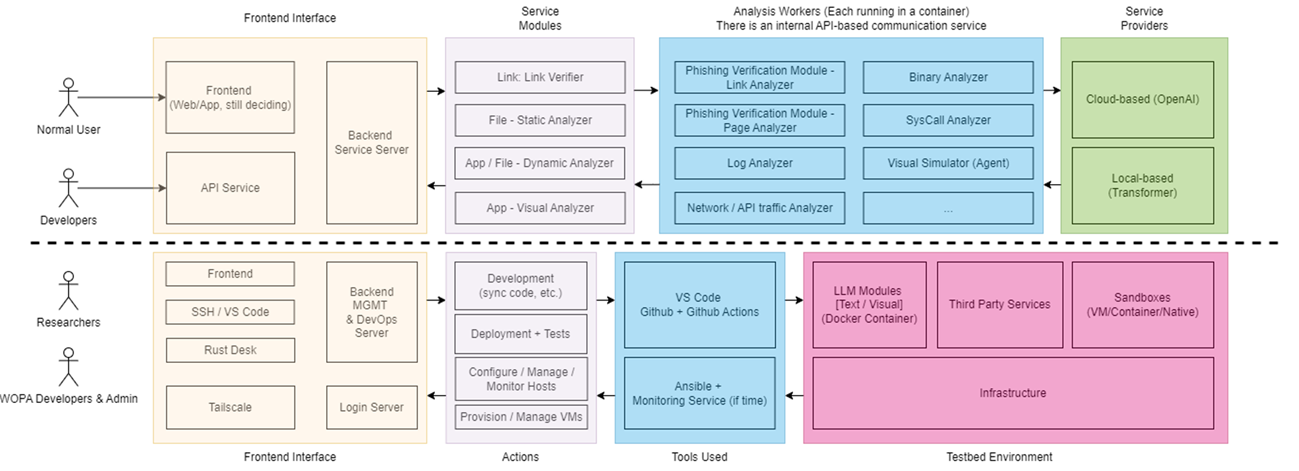


Figure. WOPA Dev FLow

# 

# 4 Requirements

This section outlines the functional and nonfunctional requirements for Wopa. These requirements define both the necessary operations of the system and the constraints under which it must function. The requirements satisfy the '4-C' criteria—Clear, Correct, Complete, and Consistent—ensuring that the system effectively meets user needs and operates smoothly.

Each requirement are written to be verifiable, meaning that it can be measured or evaluated to ensure correct implementation. The rationale for each requirement has also been considered to guide design objectives, clarify system functions, and help prioritize development efforts.

## 4.1 Functional Requirements

The functional requirements describe the core operations that Wopa must perform to protect users from security threats in real-time communication environments. These requirements support the use cases described above, focusing on threat detection, user interaction, and reporting.

1. **Secure Sandbox Environment:**
   * The system must provide a secure environment to safely open suspicious links or execute files, isolating potential threats from the user's device.
   * The system should monitor logs such as system calls, API interactions, and network traffic, making them accessible for analysis.
   * The system must support static analysis of content, like evaluating HTML source code, to assess potential risks.
2. **Text-Based Analysis Module:**
   * The tool must use Large Language Models (LLMs) to analyze textual logs without prior training (zero-shot inference), leveraging natural language rule-based instructions.
   * The module must assess the legitimacy of links/files, providing security risk reports with confidence scores based on log analysis within a set timeframe.
3. **Visual-Based Behavior Simulation Module:**
   * The tool must simulate user interactions with mobile apps, such as clicking links and navigating interfaces.
   * It must autonomously perform actions to simulate user-like behavior and provide real-time insights by monitoring the app during these simulations.
4. **Threat Detection and Reporting:**
   * The tool should combine static and dynamic analysis to detect threats with high accuracy, minimizing false positives and negatives.
   * It must generate security reports based on the detected patterns and behavior of applications.
5. **Testing and Evaluation:**
   * The tool must be tested on at least 30 Android apps to evaluate its effectiveness in detecting potential threats, particularly privacy abuses.
   * The tool must undergo a user-study session to assess its perceived usability and effectiveness.

## 4.2 Nonfunctional Requirements

Nonfunctional requirements describe the operational constraints on the system, focusing on performance, usability, reliability, and security. These constraints ensure that the system delivers its functionalities efficiently, securely, and in a user-friendly manner.

1. **Performance:**
   * The tool must perform analysis and generate reports within a reasonable timeframe, ensuring timely threat detection.
   * It must operate efficiently in the background without causing performance degradation for the user.
2. **Usability:**
   * The user interface must be intuitive, allowing users to initiate detailed link or file analysis easily.
   * The tool should automatically monitor messages and provide security reports unobtrusively in the background.
3. **Reliability:**
   * The tool must have a low error rate in detecting threats, maintaining a balance between static and dynamic analysis to minimize false positives and negatives.
4. **Scalability:**
   * The tool must be capable of integrating new analysis modules or updating existing ones as required by evolving security standards and threats.
5. **Security:**
   * The sandbox must be robust, ensuring that no potential threats escape the environment and compromise the user’s system.
   * All logs and analysis must be securely stored and protected from unauthorized access.
6. **Extensibility:**
   * The system must be adaptable, allowing developers to easily add new features, such as additional APIs for analysis or cloaking techniques, as new threats emerge.

# 

# 5 Glossary

**5.1 Terms**

| *LLM* | Large Language Model, a type of artificial intelligence (AI) program that uses deep learning to understand and analyze language |
| --- | --- |
| *Zero-shot learning* | A machine learning technique that allows an AI model to recognize and categorize new objects or concepts without being trained on examples of those concepts. The model uses knowledge transfer from pre-training on large unlabeled datasets to make predictions about new objects. |

**5.2 Works Cited**

1. Costa, Francisco Handrick da et al. “Exploring the Use of Static and Dynamic Analysis to Improve the Performance of the Mining Sandbox Approach for Android Malware Identification.” J. Syst. Softw. 183 (2021): 111092. https://doi.org/10.1016/j.jss.2021.111092.

1. Alhaidari F, Shaib NA, Alsafi M, Alharbi H, Alawami M, Aljindan R, Rahman AU, Zagrouba R. ZeVigilante: Detecting Zero-Day Malware Using Machine Learning and Sandboxing Analysis Techniques. Comput Intell Neurosci. 2022 May 9;2022:1615528. doi: 10.1155/2022/1615528.

1. F. H. d. Costa et al., "DroidXP: A Benchmark for Supporting the Research on Mining Android Sandboxes," 2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM), Adelaide, SA, Australia, 2020, pp. 143-148, doi: 10.1109/SCAM51674.2020.00021.

1. Farnood Faghihi, Mohammad Zulkernine, and Steven Ding. 2022. CamoDroid: An Android application analysis environment resilient against sandbox evasion. J. Syst. Archit. 125, C (Apr 2022). https://doi.org/10.1016/j.sysarc.2022.102452

1. Yuning Cui, Yi Sun, and Zhaowen Lin. 2023. DroidHook: a novel API-hook based Android malware dynamic analysis sandbox. Automated Software Engg. 30, 1 (May 2023). https://doi.org/10.1007/s10515-023-00378-w

1. Songsong Liu, Pengbin Feng, Shu Wang, Kun Sun, and Jiahao Cao. 2022. Enhancing malware analysis sandboxes with emulated user behavior. Comput. Secur. 115, C (Apr 2022). https://doi.org/10.1016/j.cose.2022.102613

1. H. Cai and B. G. Ryder, "DroidFax: A Toolkit for Systematic Characterization of Android Applications," 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 2017, pp. 643-647, doi: 10.1109/ICSME.2017.35.