MOBIGUARD

SHIELDING YOUR MOBILE APPS

PROJECT DEVELOPED BY

MUZAFFAR ALI | NASIR SHARIF | TALHA FIAZ

MobiGuard is a comprehensive mobile security framework designed to ensure the safety and integrity of mobile applications. It offers powerful tools for static and dynamic analysis, making mobile security assessments more efficient and reliable.

SHIELDING YOUR MOBILE APPS

Special thanks to NAVTCC and Solocoicez .

The project was developed under the supervision of **Mehmood Ali**, ensuring robust security standards and cutting-edge technology.

ABSTRACT:

In a rapidly evolving digital landscape, ensuring the security and integrity of mobile applications is not merely a recommendation—it's a necessity. MobiGuard stands as a powerful solution in this space, offering robust static analysis for Android applications to detect vulnerabilities through APK file inspection.

This project focused on creating an intuitive, developer-friendly interface that not only simplifies the security review process but also integrates Docker to enable seamless, cross-environment deployment. By streamlining static analysis into a comprehensive security tool, MobiGuard helps developers preemptively identify potential flaws and enhance the overall security of their applications before they become targets.

Through this report, we delve into the technical journey behind MobiGuard's development, emphasizing its features, technologies, and the impact it stands to have on mobile security.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

THANK YOU MESSAGE:

We would like to take this opportunity to express our sincere gratitude to the **National Vocational and Technical Training Commission (NAVTC)** for their generous sponsorship of this course. NAVTC's unwavering support for technical education and skills development has been instrumental in opening doors to brighter futures for individuals like us. By sponsoring this program, you have not only provided us with access to valuable knowledge and skill-building opportunities but have also empowered us to become more competitive and capable in today's ever-evolving job market. Your dedication to uplifting technical education in Pakistan is truly praiseworthy, and we are deeply honored to have been a part of this initiative.

Our heartfelt thanks also go to **Solochoicez**, whose seamless management and meticulous execution of the course have made the learning experience both smooth and highly rewarding. From organizing the curriculum to ensuring every aspect of the course was delivered with precision, Solochoicez played a pivotal role in making this program a success. The team's professionalism and attention to detail allowed us to focus on learning, knowing that everything behind the scenes was handled efficiently and with care. We are truly appreciative of the time and effort Solochoicez invested in ensuring that this course met the highest standards of industry, education and training.

We would like to extend a special note of thanks to **Mr. Mehmood Ali**, our dedicated course instructor, whose guidance, expertise, and passion for teaching made a profound impact on our learning journey. Mr. Mehmood Ali's deep knowledge of the subject matter, combined with his patient and thoughtful approach to instruction, allowed us to not only grasp complex concepts but also apply them in practical scenarios. His ability to break down difficult topics and provide real-world examples was instrumental in helping us connect theory to practice. We are grateful for his commitment to our growth and for going above and beyond to ensure that we left each session more confident and knowledgeable than before.

In conclusion, we are incredibly thankful to **NAVTC** for their sponsorship, to **Solochoicez** for managing and conducting the course so effectively, and to **Mr. Mehmood Ali** for his exceptional teaching. This course has been a transformative experience, equipping us with skills and knowledge that we will carry forward into our professional lives. We are inspired and motivated to apply what we have learned, and we will always remember the support and encouragement we received from NAVTC, Solochoicez, and Mr. Mehmood Ali throughout this journey.

Thank you once again for this invaluable opportunity.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

ABOUT STUDENTS:

MUZAFFAR ALI:

A recent graduate with a passion for technology and innovation. His strong foundation in computer science and programming has equipped him with the skills necessary to navigate the evolving landscape of modern technology. Eager to apply his knowledge in real-world projects, Muzaffar is driven by a desire to solve complex problems and contribute to impactful technological advancements. His fresh perspective and enthusiasm make him a promising young professional ready to take on new challenges in the tech industry.

LinkedIn Profile Link: https://www.linkedin.com/in/muzaffar-ali-34095915a/

MUHAMMAD TALHA:

A fresh graduate with a keen interest in software development and mobile application security. With a solid academic background and hands-on experience gained through various projects, Talha is well-prepared to embark on his professional journey. His strong analytical skills and ability to adapt quickly to new technologies set him apart as a motivated and capable individual, eager to contribute to the development of innovative solutions in the field of technology.

LinkedIn Profile Link: https://www.linkedin.com/in/talha-fayyaz-a69897202/

NASIR SHARIF:

A recent graduate who is passionate about leveraging his technical knowledge to develop cutting-edge solutions in the world of mobile application security and software development. His academic achievements reflect his commitment to learning and his eagerness to stay ahead of industry trends. Nasir's dedication, coupled with his problem-solving abilities, positions him as a valuable asset to any team, ready to make significant contributions to the tech world.

LinkedIn Profile Link: https://www.linkedin.com/in/nasir-sharif-b1744124b/

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

Table of Contents

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

1. INTRODUCTION:

1.1. Problem Statement: Addressing the Importance of Mobile Application Security:

As mobile applications continue to permeate every aspect of modern life, the security of these applications has become an urgent concern. The sheer volume of sensitive personal and financial data being exchanged via mobile apps makes them prime targets for malicious actors. Vulnerabilities within mobile applications can lead to unauthorized access, data breaches, and identity theft, all of which pose significant risks to both users and developers. MobiGuard addresses this critical need by offering a security framework that empowers developers to identify vulnerabilities in their Android applications before they can be exploited.

1.2. Goals and Objectives:

The primary goal of **MobiGuard** is to create a robust mobile security tool that performs static analysis on Android APK files, allowing developers to detect potential security flaws without running the application. The tool is designed to be user-friendly, enabling developers to quickly upload APK files, perform in-depth security assessments, and generate actionable reports. This project also aims to simplify the deployment process by utilizing Docker for containerization, ensuring cross-platform compatibility and consistent performance across different environments.

1.3. Significance of Developing MobiGuard:

Mobile application security is more critical than ever, given the increasing sophistication of cyber-attacks. Developing **MobiGuard** contributes to the overall mobile security ecosystem by providing an easy-to-use tool that offers developers a straightforward approach to identifying and fixing vulnerabilities in their apps. This helps mitigate the risk of data breaches and improves the overall safety of mobile applications for end-users.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

2. TECHNOLOGIES AND TOOLS USED:

2.1. Programming Language: Python:

The primary programming language used in **MobiGuard** is **Python**. It offers the flexibility needed for both file handling and back-end development. Python's extensive libraries, such as Flask and Django, allow for efficient data processing, security checks, and report generation.

2.2. Docker:

Docker was employed to containerize the entire **MobiGuard** application, ensuring that it can run consistently across various environments. Docker simplifies the deployment process and ensures that the tool's dependencies are packaged within a single container, eliminating potential environment-related issues.

2.3. Front-End: HTML, CSS, and JavaScript:

For the front-end, **HTML**, **CSS**, and **JavaScript** were used to create an intuitive and responsive user interface. These technologies ensured that the interface is modern, user-friendly, and compatible with various browsers.

2.4. Libraries and Frameworks:

Several libraries and frameworks were crucial for the development of **MobiGuard**, including:

- **Django**: For building the back-end web framework.
- **Bootstrap**: To create a responsive and user-friendly design for the front-end.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

3. PROJECT SCOPE AND FEATURES:

3.1. Overview of MobiGuard's Features:

MobiGuard - Shielding Your Mobile Apps is a comprehensive framework designed to offer developers and security analysts the tools they need to fortify mobile applications. This project primarily targets the static analysis of **Android APK files**, focusing on exposing vulnerabilities that may otherwise go undetected during dynamic testing. Below, we outline its key features and scope:

3.2. Static Analysis of Android Applications:

MobiGuard dissects APK files to inspect the underlying code, manifest files, and resources. This inspection flags potential security vulnerabilities such as insecure permissions, exposed services, and unencrypted API calls. The detailed analysis aids developers in pinpointing specific areas where their app may be compromised.

3.3. Intuitive Front-End Design:

The front-end interface was meticulously designed to ensure a smooth user experience. Developers can easily upload APK files and receive clear, actionable reports without the need for complex interactions. The simplicity in navigation is a cornerstone of the platform, catering to users with varied levels of technical expertise.

3.4. Security Assessments and Reporting:

Upon analyzing the APK file, MobiGuard generates a detailed security report, highlighting key vulnerabilities, such as the misuse of sensitive permissions, weak encryption algorithms, or unprotected API endpoints. These reports are not only informative but also provide guidance on how developers can address and resolve the identified issues.

3.5. Dockerized Environment:

By leveraging Docker, MobiGuard ensures consistency across all environments. Whether developers run the application on Linux, Windows, or macOS, the deployment process remains uniform and free from dependency concerns. This portability makes MobiGuard accessible to a global audience of developers and analysts.

3.6. Key Functionality Provided to the User:

- **Uploading Mobile Applications for Analysis**: Users can upload APK files through the front-end interface.
- **Generating Security Reports**: MobiGuard processes the uploaded APK files and generates comprehensive security reports.
- **User-Friendly Interface**: The interface allows users to easily navigate results and reports.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

4. SYSTEM ARCHITECTURE:

4.1. High-Level Architecture:

The architecture of MobiGuard is designed to balance security, scalability, and flexibility, all while ensuring seamless interaction between the user interface and the backend.
The system architecture can be visualized as follows:

4.2. Components of the System

Below, we break down the key components that make up MobiGuard's architecture:

- **Back-End**: Developed using Python, the back-end handles APK file uploads, performs static analysis, and generates security reports.
- **Front-End**: The front-end, built using HTML, CSS, and JavaScript, allows users to upload APKs and view the results.
- **Security Analysis**: The static analyzer, built into the back-end, deconstructs the APK file and inspects its components for security vulnerabilities.
- **Reporting**: A database stores the analysis results, which are then displayed in the form of security reports.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of MEHMOOD ALI

5. DEVELOPMENT PROCESS:

5.1. Planning and Requirements Gathering:

Before development began, the team outlined the core features and objectives of the project. The primary focus was on creating a tool that could easily be used by developers to perform static analysis on mobile applications.

5.2. Design Phase:

- **Wireframes and UI Design**: Wireframes were created for the front-end to plan the layout and functionality of the user interface.
- **Back-End Structure**: The back-end was designed to handle file uploads and perform static analysis on APK files.

5.3. Development Phase:

- **Python Programming:** The back-end was developed using Python, leveraging various libraries for APK analysis.
- **Secure Environment Setup:** Docker was used to create a secure, isolated environment for the analysis process.
- **Front-End Development:** HTML, CSS, and JavaScript were used to develop the user interface.
- **Static Analyzer Integration:** The static analyzer was integrated into the back-end, enabling the tool to inspect APK files.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

6. DOCKER INTEGRATION FOR MOBIGUARD:

6.1. Introduction to Docker:

Docker was critical to the development of **MobiGuard** as it simplified the deployment process. By containerizing the application, Docker ensured consistency across different environments.

6.2. Docker Installation Process:

Linux/Mac/Windows: Docker can be installed on any of these platforms. Instructions for setting up Docker and activating WSL for Windows were provided to ensure a seamless development experience.

Installation Guide:

*This section provides a comprehensive guide for setting up the **MobiGuard** project on both **Windows** (using WSL) and **Linux** environments, including Docker installation, WSL activation (for Windows users), and adding a development environment for Docker.*

1-WSL Activation (For Windows Users):

*To enable a seamless development experience on Windows, **Windows Subsystem for Linux (WSL)** allows you to run Linux distributions alongside your existing system. Here's how to activate WSL on your Windows machine:*

Video Link for Activation and Installation: *https://www.youtube.com/watch?v=WDEdRmTCSs8*

Step 1: Enable WSL

- *Open **PowerShell** as an Administrator (right-click on the Start menu, then select PowerShell (Admin)).*
- *Run the following command to enable WSL:*

```bash
wsl --install
```

This command will automatically enable WSL and install Ubuntu (the default Linux distribution). You can later choose a different distribution if needed.

Windows 10: *If you are using an older version, make sure to update Windows and use wsl --set-version 2 to ensure you are using WSL 2.*

Step 2: Install Your Preferred Linux Distribution

*By default, **Ubuntu** will be installed, but if you want to install another distribution, such as Debian or Kali, you can do so via the **Microsoft Store**.*

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

Step 3: Update and Upgrade Packages:

Once the WSL setup is complete, open ***Ubuntu*** *(or your preferred distribution) from the Start menu. Update the package lists using the following commands:*

```
bash

sudo apt update && sudo apt upgrade -y
```

2-Docker Installation for Windows (with WSL 2 Integration)

Step 1: Install Docker Desktop

A. *Download* ***Docker Desktop*** *for Windows from the official Docker website.*
B. *During installation, ensure that the* ***"Use WSL 2 based engine"*** *option is selected.*
C. *After installation, launch Docker Desktop.*

Step 2: Enable WSL 2 Integration with Docker

A. *Open* ***Docker Desktop****.*
B. *Go to* ***Settings > Resources > WSL Integration****.*
C. *Enable integration with your installed Linux distribution (e.g., Ubuntu).*

Docker Installation for Linux:

For Linux users, Docker can be installed natively. Here's how you can install Docker on ***Ubuntu*** *(similar commands apply to other Linux distributions):*

Step 1: Install Docker

A. *Update your existing packages:*

```
bash                                                    Copy code

sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

B. *Add Docker's official GPG key and repository: Push the following commands in terminal*

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

C. *Install Docker:*

```
bash

sudo apt update
sudo apt install docker-ce
```

D. *Start Docker and enable it to run at startup:*

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

```bash
sudo systemctl start docker
sudo systemctl enable docker
```

Step 2: Verify Docker Installation:

After Docker is installed, check that it's working correctly by running:

docker –version

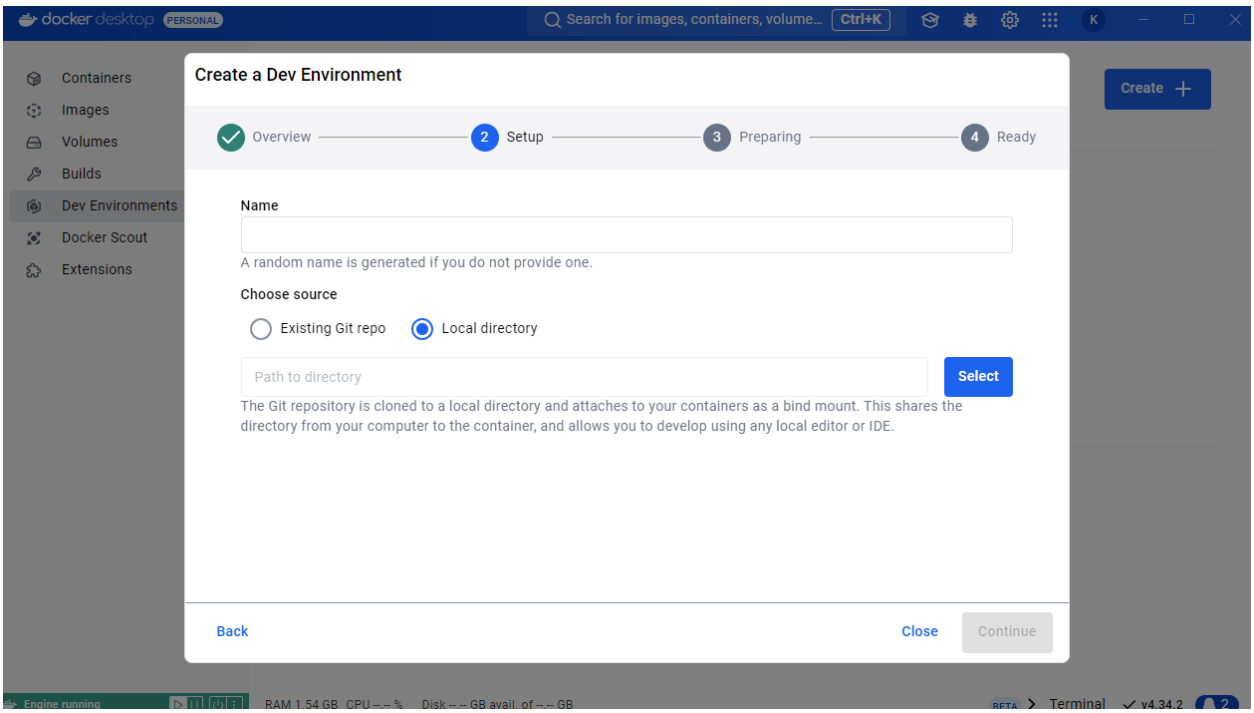
3-Setting Up the Development Environment in Docker:

With Docker installed, the next step is to set up a development environment for MobiGuard.

A. Open Desktop Docker and Click on "Dev Environment"

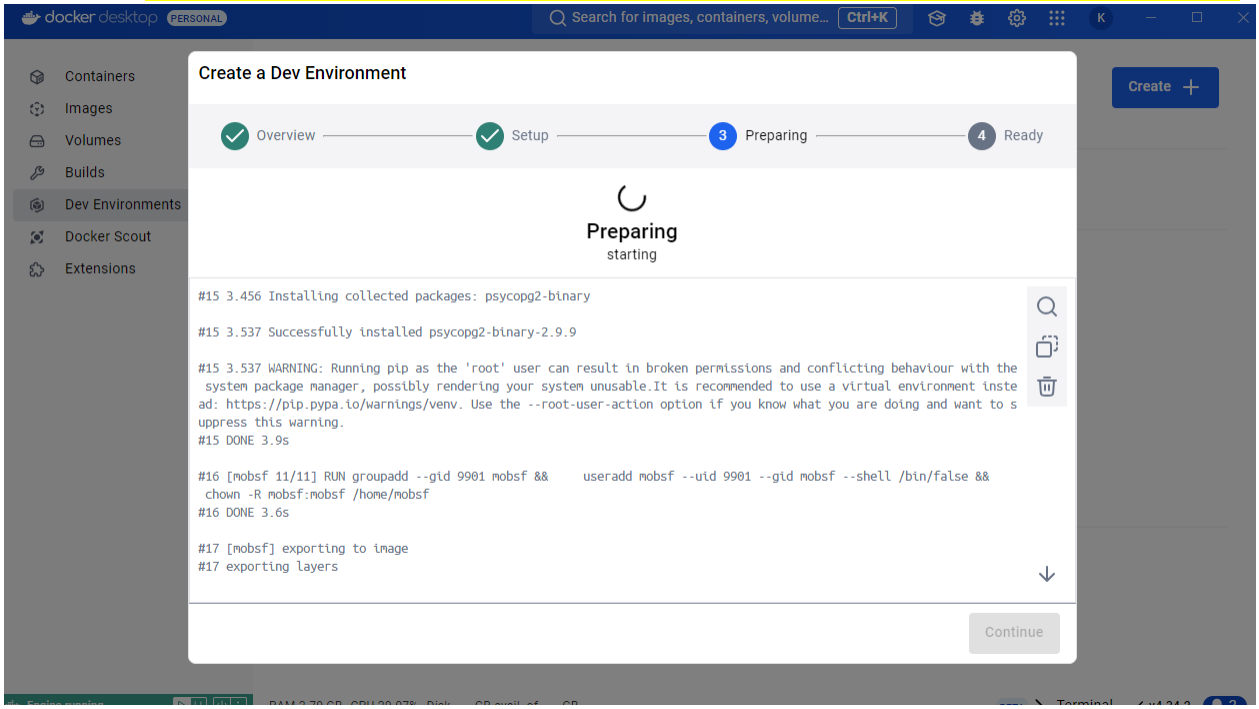
B. Create new Environment:

C. Select Local Directory option and then select the MobiGuard folder from your computer.



Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of MEHMOOD ALI

D. After selection of the project folder, let Docker create the environment and install the dependencies:



E. After installing complete click on Continue, and you dev Environment has been created.

6.3. Dockerizing the MobiGuard Application:

- **Writing the Dockerfile:** The Dockerfile specifies the environment and dependencies needed to run MobiGuard.
- **Building the Docker Image:** The Docker image for MobiGuard was built using this Docker file.

Navigate to the project directory where the Dockerfile is located. Run the following command to build the Docker image:

```bash
docker build -t mobiguard .
```

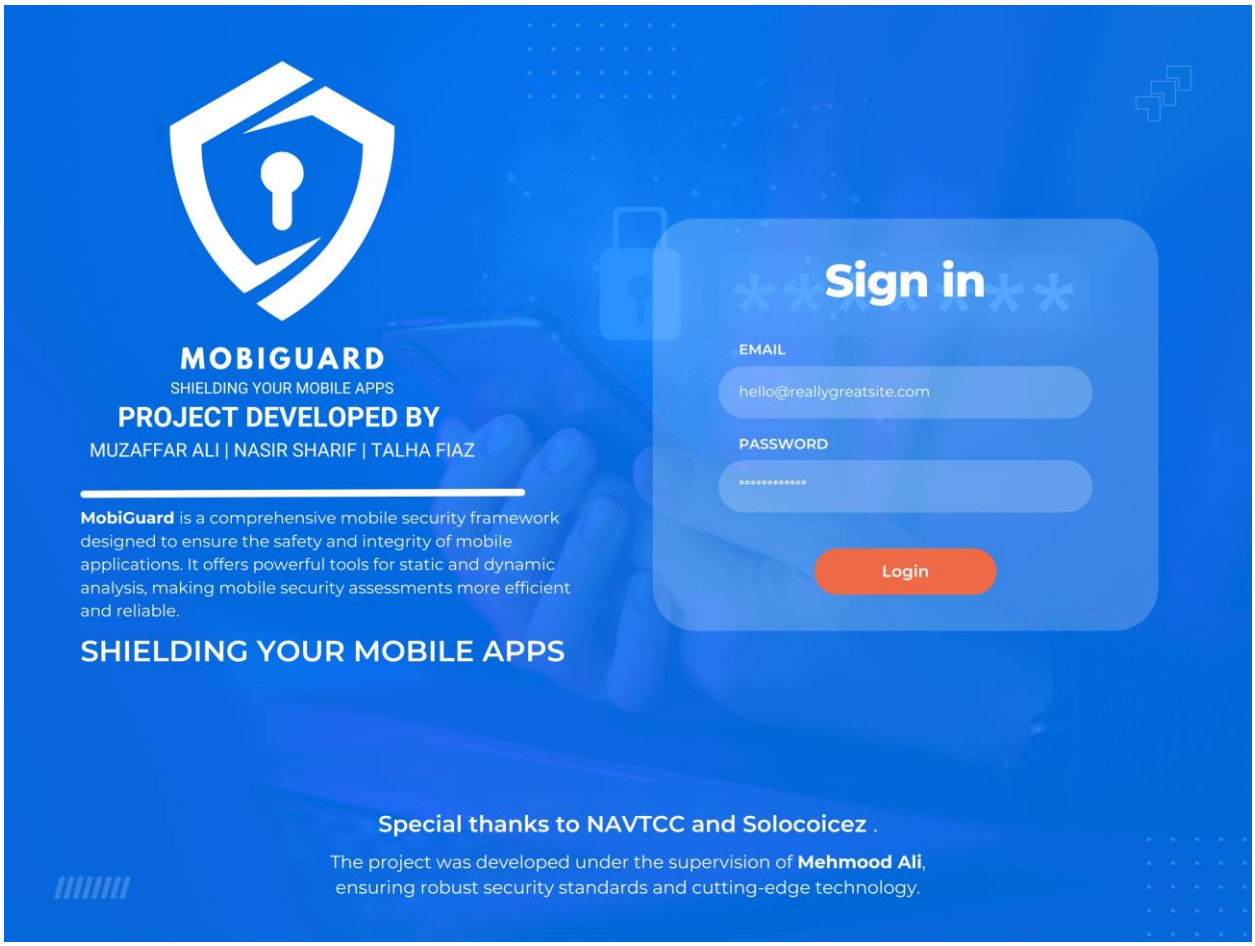
This command tells Docker to build an image named mobiguard using the Dockerfile.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of MEHMOOD ALI

- **Running the Container:** Once the Docker image is built, MobiGuard can be run in a containerized environment, ensuring consistent performance across platforms.

```bash
docker run -p 8000:8000 mobiguard
```

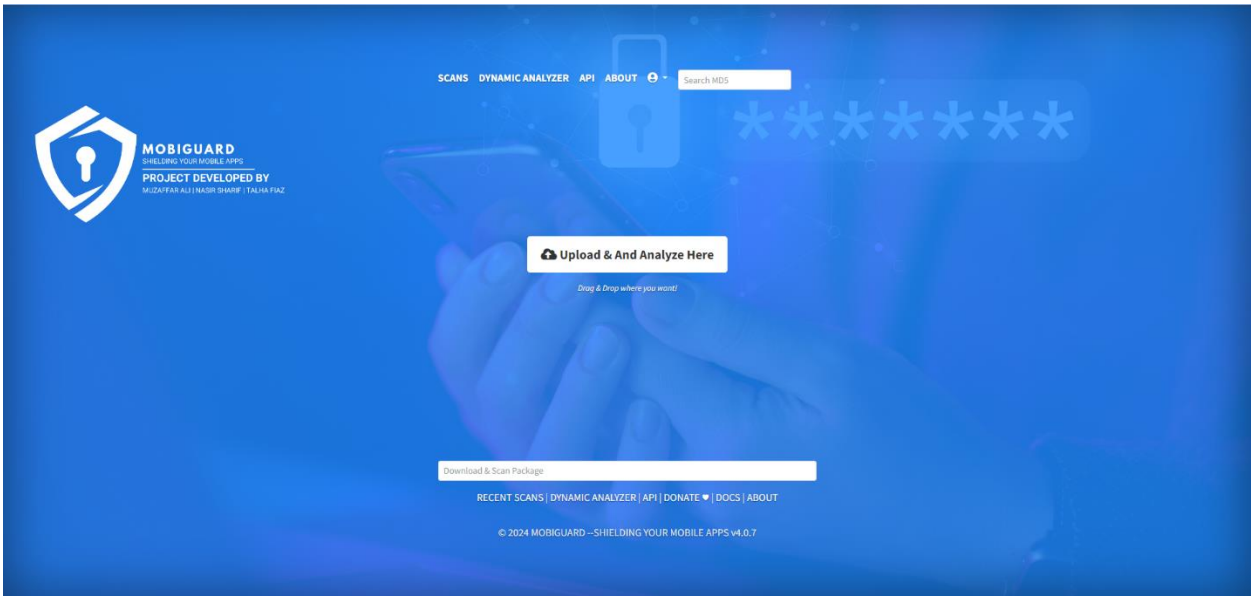
This command runs the container and maps port 8000 of the container to port 8000 on your local machine. The application will now be accessible at `http://localhost:8000`
The default credentials are:
Username: mobsf
Password: mobsf



7. STATIC ANALYZER FOR MOBILE APPLICATIONS

7.1. Static Analysis:

The static analyzer in **MobiGuard** inspects APK files for potential security risks. This is done by analyzing the code, manifest file, and other resources, without executing the application.



7.2. Steps Involved in Static Analysis:

- **File Inspection:** The APK file is deconstructed, and its components are inspected.
- **Manifest Analysis:** The AndroidManifest.xml file is checked for insecure permissions or other vulnerabilities.
- **API Call Checking:** The analyzer checks for unsafe API calls that may expose the application to security risks.

7.3. Identifying Security Risks:

The static analyzer flags potential vulnerabilities and generates reports to help developers mitigate security issues before deployment.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

8. SECURITY ANALYSIS WORKFLOW IN MOBIGUARD

8.1. How Apps Are Uploaded and Analyzed

Users upload APK files through the front-end. The back-end processes the file, and the static analyzer inspects its components.

8.2. Extracting and Inspecting App Components

The static analyzer deconstructs the APK, checking each component for vulnerabilities.

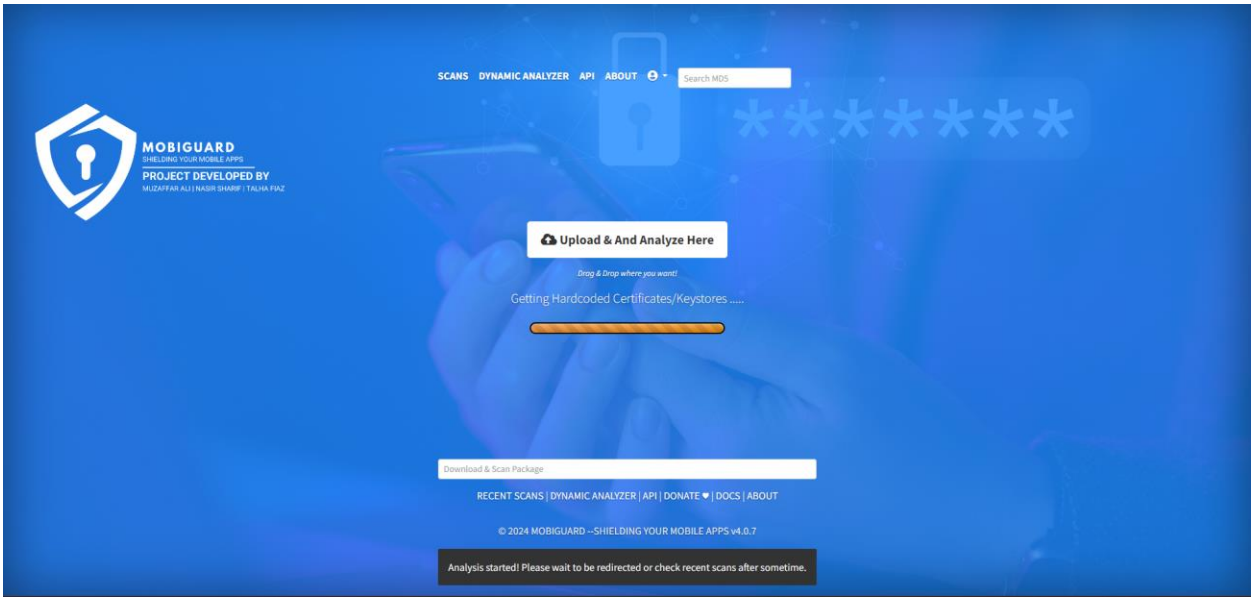
8.3. Example Analysis Results

The generated security report categorizes vulnerabilities by severity, helping developers prioritize fixes.

9. User Guide

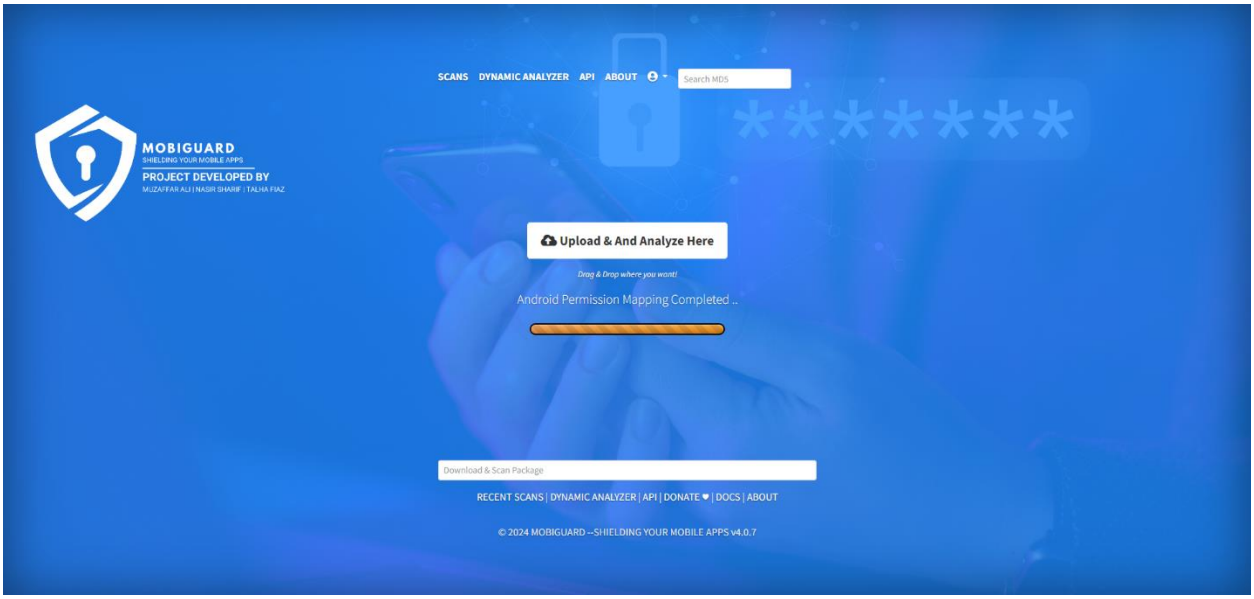
9.1. Step-by-Step Guide for Using MobiGuard

- **Uploading an APK File:** Use the upload feature to add your APK file for analysis.



Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of MEHMOOD ALI

- **Running the Analysis:** After receiving the APK, MobiGuard's static analyzer springs into action. The APK is deconstructed into its various components, starting with the **AndroidManifest.xml** file, which defines the app's structure and permissions. The analyzer also inspects the compiled code, checking for unsafe API calls, hardcoded credentials, and insecure coding practices that could pose a threat to the app's security. Resource files, such as images and configuration files, are examined for potential vulnerabilities as well.

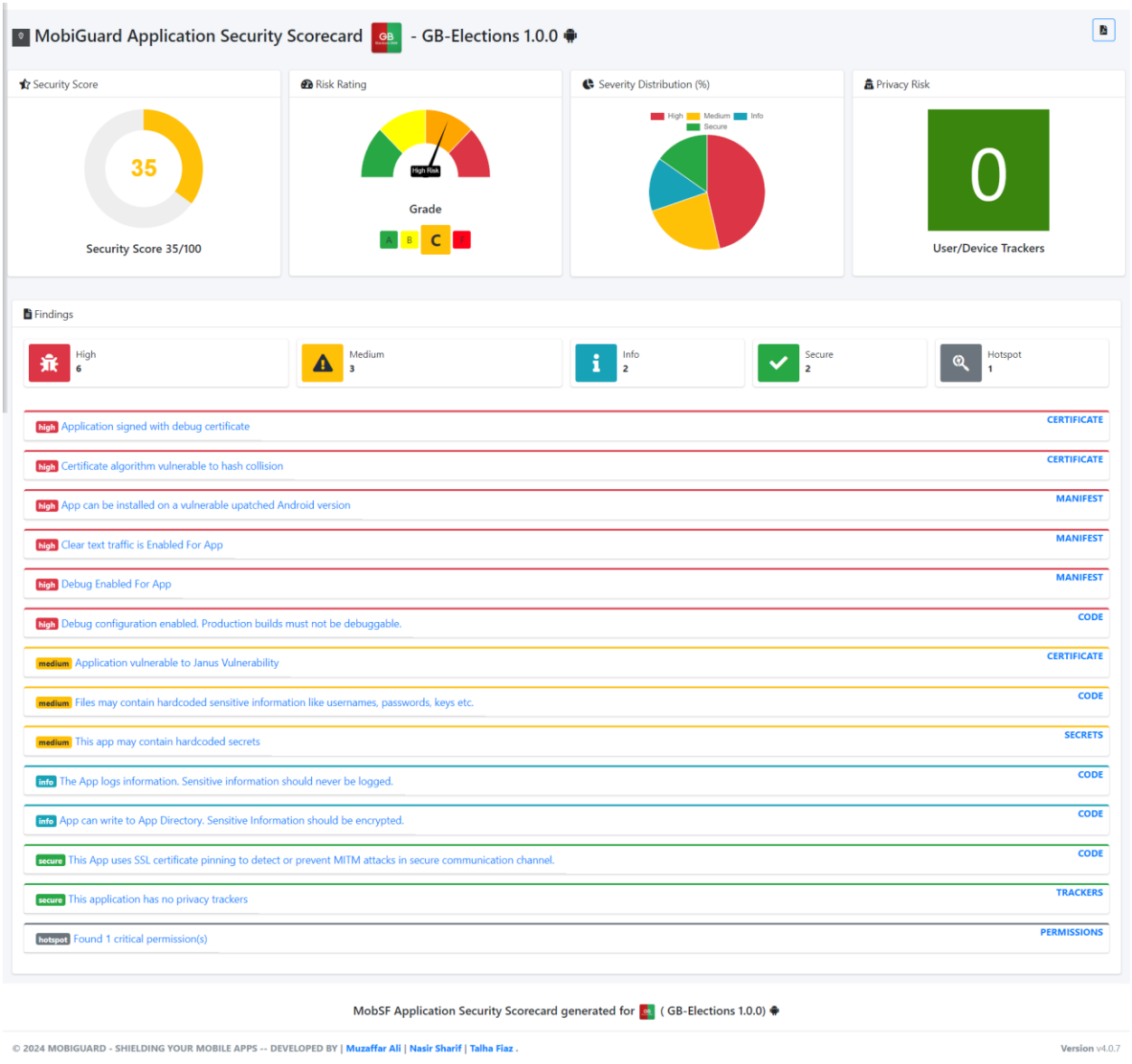


- **Interpreting Results:** The security report provides details on any vulnerabilities found, categorized by severity.
- **Review and Action:** The security report is displayed through MobiGuard's front-end interface. The user can review the vulnerabilities identified in their APK, filter the results by severity, and prioritize fixes based on the recommendations provided. For those requiring offline access, the report can be downloaded in PDF format, making it easy to share with other team members or stakeholders.

The streamlined nature of MobiGuard's security analysis workflow ensures that developers can swiftly identify and address security concerns within their mobile applications. This proactive approach to mobile security is crucial in today's fast-

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

paced development environments, where vulnerabilities can quickly escalate into severe threats if left unchecked.



Note: A Detailed Reports are attached separately at the end of this report:

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of MEHMOOD ALI

10. CHALLENGES AND SOLUTIONS:

Like any ambitious development project, the creation of MobiGuard came with its own set of challenges. Each hurdle presented an opportunity to innovate, refine, and improve the tool to ensure it met its objectives. Below are some of the key challenges encountered during the development of MobiGuard, along with the solutions that were implemented to overcome them:

10.1. Handling Large APK Files

- **Challenge:** Mobile applications come in all sizes, and some APK files can be particularly large, containing numerous resources and libraries. The size and complexity of these files posed a challenge for MobiGuard's static analyzer, leading to longer processing times and occasional memory constraints.
- **Solution:** To address this, we optimized the file processing mechanism on the back-end. By implementing multi-threading in Python, MobiGuard was able to handle larger APK files more efficiently, distributing the workload and significantly reducing analysis times. Additionally, improvements in memory management were made to ensure that the application could process complex files without running into memory limits.

10.2. Cross-Platform Compatibility:

- **Challenge**: One of the primary objectives for MobiGuard was to ensure that it could run seamlessly across both Windows and Linux environments. However, the differences in dependencies and configurations between operating systems initially caused compatibility issues during testing.
- **Solution**: The integration of **Docker** proved to be a game-changer. By containerizing the application, we were able to package MobiGuard and all its dependencies into a single Docker container, ensuring that it would run consistently on any platform. Docker provided an isolated environment that eliminated compatibility issues, making MobiGuard accessible to users regardless of their operating system.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

10.3. Streamlining Front-End Usability:

- **Challenge**: During early testing, it became evident that the initial front-end design was not as intuitive as we had hoped. Users found it somewhat cumbersome to navigate, particularly when uploading files and interpreting the results of the analysis.
- **Solution**: We conducted several rounds of user testing and collected feedback from a diverse group of developers. Based on this feedback, we made iterative improvements to the front-end interface. The design was streamlined, improving navigation and responsiveness, and the file upload process was simplified. We also enhanced the layout of the security reports, making it easier for users to interpret the findings and take action on the identified vulnerabilities.

10.4. Docker Environment Setup:

- **Challenge**: While Docker provided the necessary solution for cross-platform compatibility, setting up the Docker environment itself posed a challenge, particularly for developers who were not familiar with Docker.
- **Solution**: To mitigate this, we created a comprehensive **installation guide** that provided step-by-step instructions for setting up Docker on both Windows (with **WSL 2** integration) and Linux environments. This guide ensured that users could quickly get MobiGuard up and running, even if they were new to Docker. We also simplified the Docker configuration, reducing the number of manual steps required during installation.

Through innovative problem-solving and continuous iteration, we were able to overcome these challenges, resulting in a more robust and user-friendly product. Each challenge pushed the team to find

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

better solutions, ultimately strengthening MobiGuard's capabilities and enhancing the overall user experience.

11. TESTING AND VALIDATION

Testing and validation were integral to ensuring that MobiGuard operated smoothly and delivered accurate results. A rigorous approach was taken to verify the functionality, performance, and usability of the system across different platforms and use cases. Below are the key aspects of the testing and validation process:

11.1. Functional Testing:

The functional testing phase focused on ensuring that all features of MobiGuard performed as expected. This phase involved:

11.1.1. Login and Authentication:

The login functionality was tested with various user credentials to verify secure access to the system. Edge cases, such as incorrect passwords and invalid usernames, were handled to ensure a robust authentication mechanism.

11.1.2. APK File Uploads:

The upload process was tested with multiple APK files of varying sizes and complexities. MobiGuard handled both small and large files without any loss of functionality, and the analysis began automatically upon upload.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

11.1.3. Static Analysis:

Functional testing of the static analyzer involved running APK files with known vulnerabilities through the system. The static analyzer was able to correctly identify security issues and generate reports, ensuring the accuracy of the analysis.

11.1.4. Report Generation:

The generated reports were reviewed to ensure that all detected vulnerabilities were categorized appropriately, with severity levels assigned to each issue.

11.2. Performance Testing:

In addition to functional validation, MobiGuard underwent extensive performance testing to assess how well it handled large loads and complex APK files.

11.2.1. Stress Testing:

To ensure the application could manage multiple users and APK files simultaneously, stress testing was conducted by uploading multiple APKs at once. MobiGuard demonstrated resilience under high load, with no significant delays or system crashes.

11.2.2. Scalability:

The static analyzer's performance was tested with increasingly larger APK files, ensuring that the system scaled efficiently without compromising speed or accuracy. Optimization of memory usage and multi-threading helped MobiGuard maintain fast response times, even with resource-intensive files.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

11.3. Usability Testing:

Given the emphasis on a user-friendly interface, usability testing was conducted to ensure that MobiGuard was intuitive and easy to navigate.

11.3.1. User Feedback:

Developers and security professionals were invited to test the interface, focusing on file uploads, report viewing, and navigation. Feedback was gathered on the design and usability of the front-end, and iterative improvements were made based on this feedback.

11.3.2. Improved User Experience:

Usability testing led to enhancements in the file upload process and report layout. The drag-and-drop feature, as well as clearer navigation options, contributed to a more streamlined experience for users.

Through this multi-layered testing and validation process, MobiGuard was refined to meet high standards of performance, functionality, and user satisfaction. The combination of functional, performance, and usability testing ensured that MobiGuard could reliably analyze mobile applications while offering a seamless experience for users.

12. CONCLUSION:

The development of **MobiGuard - Shielding Your Mobile Apps** has resulted in a comprehensive and versatile tool that addresses one of the most pressing challenges in mobile development—security. By providing developers and security analysts with a

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

powerful static analysis framework, MobiGuard allows for early detection of vulnerabilities within Android APK files, enabling proactive measures to safeguard applications before they are deployed.

MobiGuard's intuitive interface, Dockerized environment, and detailed security reporting make it an invaluable resource for developers seeking to enhance the security of their mobile applications. The use of Docker ensures that the tool runs consistently across different platforms, while the static analyzer provides actionable insights into potential vulnerabilities. The project has not only achieved its primary objectives but has also set the stage for future innovations in mobile app security.

Through this report, we have highlighted the technologies, processes, and challenges involved in developing MobiGuard, showcasing its contribution to the field of mobile application security. MobiGuard's ability to identify vulnerabilities without executing the code ensures a non-invasive, efficient solution for developers to protect their applications from potential threats.

Looking ahead, MobiGuard has the potential to evolve further, with possibilities including dynamic analysis, multi-platform support, and machine learning-based vulnerability detection. These future improvements will keep MobiGuard at the forefront of mobile security solutions, providing developers with the tools they need to build secure, robust applications in an increasingly complex digital landscape.

13. FUTURE SCOPE

While **MobiGuard** provides a robust solution for static analysis of Android APK files, there are several opportunities for future development that could significantly enhance its capabilities. The following outlines key areas where MobiGuard can evolve to remain at the forefront of mobile security innovation:

13.1.1. Dynamic Analysis:

One of the major next steps for MobiGuard is to introduce **dynamic analysis**. Unlike static analysis, which inspects code without execution, dynamic analysis

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

observes an application in runtime. This enhancement would enable MobiGuard to detect vulnerabilities such as insecure network communications, runtime privilege escalations, or behavioral anomalies that may not be apparent in static code analysis. By simulating real-world conditions, MobiGuard can offer a more comprehensive view of an app's security posture.

13.1.2. Cloud-Based Analysis:

Moving MobiGuard to the cloud could enhance its accessibility and scalability. A cloud-based version would allow developers and security professionals to perform static analysis without the need for local installation. This would also enable the storage and retrieval of analysis reports from anywhere, making it easier for teams to collaborate. Additionally, cloud infrastructure would facilitate scalability, allowing MobiGuard to handle larger volumes of APK files simultaneously.

13.1.3. Multi-Platform Support:

While MobiGuard currently focuses on Android applications, expanding support to other mobile platforms like **iOS** could significantly broaden its impact. By enabling cross-platform analysis, MobiGuard would cater to a larger audience of developers and security professionals, offering security insights across both major mobile ecosystems. This expansion would require adapting the static analyzer to work with iOS app packages and implementing corresponding security rules.

13.1.4. Machine Learning for Vulnerability Detection:

Incorporating **machine learning** into MobiGuard's core could dramatically improve its ability to detect new or evolving security threats. By training machine learning models on existing APK files and their vulnerabilities, MobiGuard could begin identifying patterns in previously unseen threats. This feature would allow for real-time, adaptive security analysis that evolves alongside emerging threats, making the tool even more proactive in its protection capabilities.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

13.1.5. Integration with CI/CD Pipelines:

To better support developers and security teams, MobiGuard could integrate with **Continuous Integration/Continuous Deployment (CI/CD) pipelines**. This would allow automated security scans to occur as part of the development cycle, ensuring that security is a built-in step in the software development process. With this integration, vulnerabilities could be detected and addressed earlier, improving overall development efficiency and product security.

By focusing on these areas for future enhancement, MobiGuard has the potential to grow into a fully comprehensive mobile security platform. The ongoing evolution of the project will ensure that it remains relevant in the ever-changing landscape of mobile application security, continuing to serve the needs of developers and security professionals alike.

14. References

The development of MobiGuard relied on a variety of external tools, libraries, and resources to ensure its functionality and scalability. Below is a list of key references and documentation that were instrumental in the project's success:

14.1.1. Python Documentation:

Python was the primary language used for the development of MobiGuard's back-end, handling file analysis, report generation, and integration with the static analyzer. The official Python documentation provided critical insights into Python's libraries and best practices.

Python Official Documentation: https://docs.python.org/

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

14.1.2. Docker Documentation:

Docker was a crucial tool for containerizing MobiGuard, enabling seamless cross-platform deployment. The Docker documentation was referenced extensively to ensure proper setup and usage across both Linux and Windows environments.
Docker Official Documentation: https://docs.docker.com/

14.1.3. Django Framework:

Django was used to develop the web framework for MobiGuard's front-end, providing a lightweight and flexible structure.
DJANGO Documentation: https://docs.djangoproject.com/en/5.1/

14.1.4. Android Static Analysis Tools:

MobiGuard's static analyzer was built using various open-source libraries designed to inspect Android APK files. Documentation from these libraries was crucial in understanding how to effectively deconstruct and analyze APK files for security vulnerabilities.

Androguard (Android Application Reverse Engineering Tool): https://github.com/androguard/androguard

14.1.5. JavaScript & Front-End Technologies:

The front-end interface for MobiGuard was developed using HTML, CSS, and JavaScript, along with Bootstrap for a responsive design. The documentation for these technologies provided guidance for creating a clean and intuitive user experience.
Bootstrap Documentation: https://getbootstrap.com/

14.1.6. Git for Version Control:

Git was used for version control throughout the development of MobiGuard. This ensured that the codebase remained organized and that changes could be tracked and managed efficiently.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

Git Documentation: https://git-scm.com/doc

These references were essential in shaping the development process of MobiGuard and ensured the project's smooth execution.

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

A DETAILS REPORTS ARE ATTACHED BELOW

Special thanks to NAVTCC and SOLOCOICEZ
The project was developed under the supervision of **MEHMOOD ALI**

APP SCORES

Security Score 35/100
Trackers Detection 0/432
MobSF Scorecard

FILE INFORMATION

File Name GB-Elections(V-1.0.6).apk
Size 41.21MB
MD5 a05dafa2869bd65b5725df73ea7c08fa
SHA1 4edcc9ed95cc0ecb589c60d491f9261dd0049bbe
SHA256 1a9b27589211e96ec3150c98aa1be7f0cb4ca8261fd3c567ac25ac904af22185

APP INFORMATION

App Name GB-Elections
Package Name com.rapidzz.gbvoters
Main Activity com.rapidzz.gbvoters.activities.SplashActivity
Target SDK 29 Min SDK 15 Max SDK
Android Version Name 1.0.0 Android Version Code 1

8 ACTIVITIES	1 SERVICES	0 RECEIVERS	0 PROVIDERS
View	View	View	View
Exported Activities 0	Exported Services 0	Exported Receivers 0	Exported Providers 0

SCAN OPTIONS

Rescan | Manage Suppressions | Start Dynamic Analysis | Scan Logs

DECOMPILED CODE

View AndroidManifest.xml | View Source | View Smali | Download Java Code | Download Smali Code | Download APK

SIGNER CERTIFICATE

```
Binary is signed
v1 signature: True
v2 signature: True
v3 signature: False
v4 signature: False
X.509 Subject: CN=Android Debug, O=Android, C=US
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2019-10-15 06:08:22+00:00
Valid To: 2049-10-07 06:08:22+00:00
Issuer: CN=Android Debug, O=Android, C=US
Serial Number: 0x1
Hash Algorithm: sha1
md5: b62e66d9b65ac69394912fc657c0f27d
sha1: 7012d6c3639140052076cf0b782e9d91e06d31a6
sha256: cca454b5c15c75bebe6d83408cef0fb8de33193791153b570d7922c3411a8ce8
sha512: 4aef2b53d2bec2bca9d2ad13b55a0e952f901cfe8ea2412c7787627e29f1c8a8edacf2cb653f7185670273b090f7f499de40821fa30854ddeaefeaaf5937e176
PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: e6904222047a0862e28c780d2bdee8fa59ac79a7e56c2baaa6fe787af9a47462
Found 1 unique certificates
```

APPLICATION PERMISSIONS

Search:

PERMISSION	STATUS	INFO	DESCRIPTION	CODE MAPPINGS
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.	Show Files
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents	Allows an application to write to external storage.	Show Files

Showing 1 to 2 of 2 entries
Previous 1 Next

ANDROID API

Search:

API	FILES
Crypto	Show Files
Inter Process Communication	Show Files
Java Reflection	Show Files
Local File I/O Operations	Show Files
Message Digest	Show Files
Starting Activity	Show Files
TCP Socket	Show Files

Showing 1 to 7 of 7 entries
Previous 1 Next

BROWSABLE ACTIVITIES

Search:

ACTIVITY	INTENT
No data available in table	

Showing 0 to 0 of 0 entries
Previous Next

NETWORK SECURITY

Search:

NO	SCOPE	SEVERITY	DESCRIPTION
No data available in table			

Showing 0 to 0 of 0 entries
Previous Next

CERTIFICATE ANALYSIS

HIGH 2 | WARNING 1 | INFO 1

Search:

TITLE	SEVERITY	DESCRIPTION
Application signed with debug certificate	high	Application signed with a debug certificate. Production application must not be shipped with a debug certificate.
Application vulnerable to Janus Vulnerability	warning	Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android 5.0-8.0, if signed only with v1 signature scheme. Applications running on Android 5.0-7.0 signed with v1, and v2/v3 scheme is also vulnerable.
Certificate algorithm vulnerable to hash collision	high	Application is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues.
Signed Application	info	Application is signed with a code signing certificate

Showing 1 to 4 of 4 entries
Previous 1 Next

MANIFEST ANALYSIS

HIGH 3 | WARNING 0 | INFO 0 | SUPPRESSED 0

Search:

NO	ISSUE	SEVERITY	DESCRIPTION	OPTIONS
1	App can be installed on a vulnerable upatched Android version Android 4.0.3-4.0.4, [minSdk=15]	high	This application can be installed on an older version of android that has multiple unfixed vulnerabilities. These devices won't receive reasonable security updates from Google. Support an Android version => 10, API 29 to receive reasonable security updates.	
2	Clear text traffic is Enabled For App [android:usesCleartextTraffic=true]	high	The app intends to use cleartext network traffic, such as cleartext HTTP, FTP stacks, DownloadManager, and MediaPlayer. The default value for apps that target API level 27 or lower is "true". Apps that target API level 28 or higher default to "false". The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering; a network attacker can eavesdrop on transmitted data and also modify it without being detected.	
3	Debug Enabled For App [android:debuggable=true]	high	Debugging was enabled on the app which makes it easier for reverse engineers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes.	

Showing 1 to 3 of 3 entries
Previous 1 Next

CODE ANALYSIS

HIGH 1 | WARNING 1 | INFO 2 | SECURE 1 | SUPPRESSED 0

Search:

NO	ISSUE	SEVERITY	STANDARDS	FILES	OPTIONS
1	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	**CWE:** CWE-312: Cleartext Storage of Sensitive Information **OWASP Top 10:** M9: Reverse Engineering **OWASP MASVS:** MSTG-STORAGE-14	com/rapidzz/gbvoters/model/ResponseModels/User.java com/rapidzz/gbvoters/model/Source/AppConstants.java	
2	The App logs information. Sensitive information should never be logged.	info	**CWE:** CWE-532: Insertion of Sensitive Information into Log File **OWASP MASVS:** MSTG-STORAGE-3	com/edittextpicker/aliazaz/EditTextPicker.java com/rapidzz/gbvoters/activities/MainActivity.java com/rapidzz/gbvoters/model/Source/Repository/DataRepository.java	
3	App can write to App Directory. Sensitive Information should be encrypted.	info	**CWE:** CWE-276: Incorrect Default Permissions **OWASP MASVS:** MSTG-STORAGE-14	com/rapidzz/gbvoters/Utils/SessionManager.java	
4	Debug configuration enabled. Production builds must not be debuggable.	high	**CWE:** CWE-919: Weaknesses in Mobile Applications **OWASP Top 10:** M1: Improper Platform Usage **OWASP MASVS:** MSTG-RESILIENCE-2	com/rapidzz/gbvoters/BuildConfig.java	
5	This App uses SSL certificate pinning to detect or prevent MITM attacks in secure communication channel.	secure	**OWASP MASVS:** MSTG-NETWORK-4	com/rapidzz/gbvoters/model/Source/RetrofitClientInstance.java	

Showing 1 to 5 of 5 entries
Previous 1 Next

SHARED LIBRARY BINARY ANALYSIS

No Shared Objects found.

Search:

NO	SHARED OBJECT	NX	STACK CANARY	RELRO	RPATH	RUNPATH	FORTIFY	SYMBOLS STRIPPED
No data available in table								

Showing 0 to 0 of 0 entries
Previous Next

NIAP ANALYSIS v1.3

Search:

NO	IDENTIFIER	REQUIREMENT	FEATURE	DESCRIPTION
No data available in table				

Showing 0 to 0 of 0 entries
Previous Next

FILE ANALYSIS

Search:

NO	ISSUE	FILES
No data available in table		

Showing 0 to 0 of 0 entries
Previous Next

APKiD ANALYSIS

Search:

DEX	DETECTIONS
classes.dex	FINDINGS: Compiler — DETAILS: r8 Showing 1 to 1 of 1 entries
classes2.dex	FINDINGS: Anti-VM Code — DETAILS: Build.FINGERPRINT check, Build.MODEL check, Build.MANUFACTURER check, Build.PRODUCT check FINDINGS: Compiler — DETAILS: r8 without marker (suspicious) Showing 1 to 2 of 2 entries

Showing 1 to 2 of 2 entries
Previous 1 Next

QUARK ANALYSIS

Search:

POTENTIAL MALICIOUS BEHAVIOUR	EVIDENCE
No data available in table	

Showing 0 to 0 of 0 entries
Previous Next

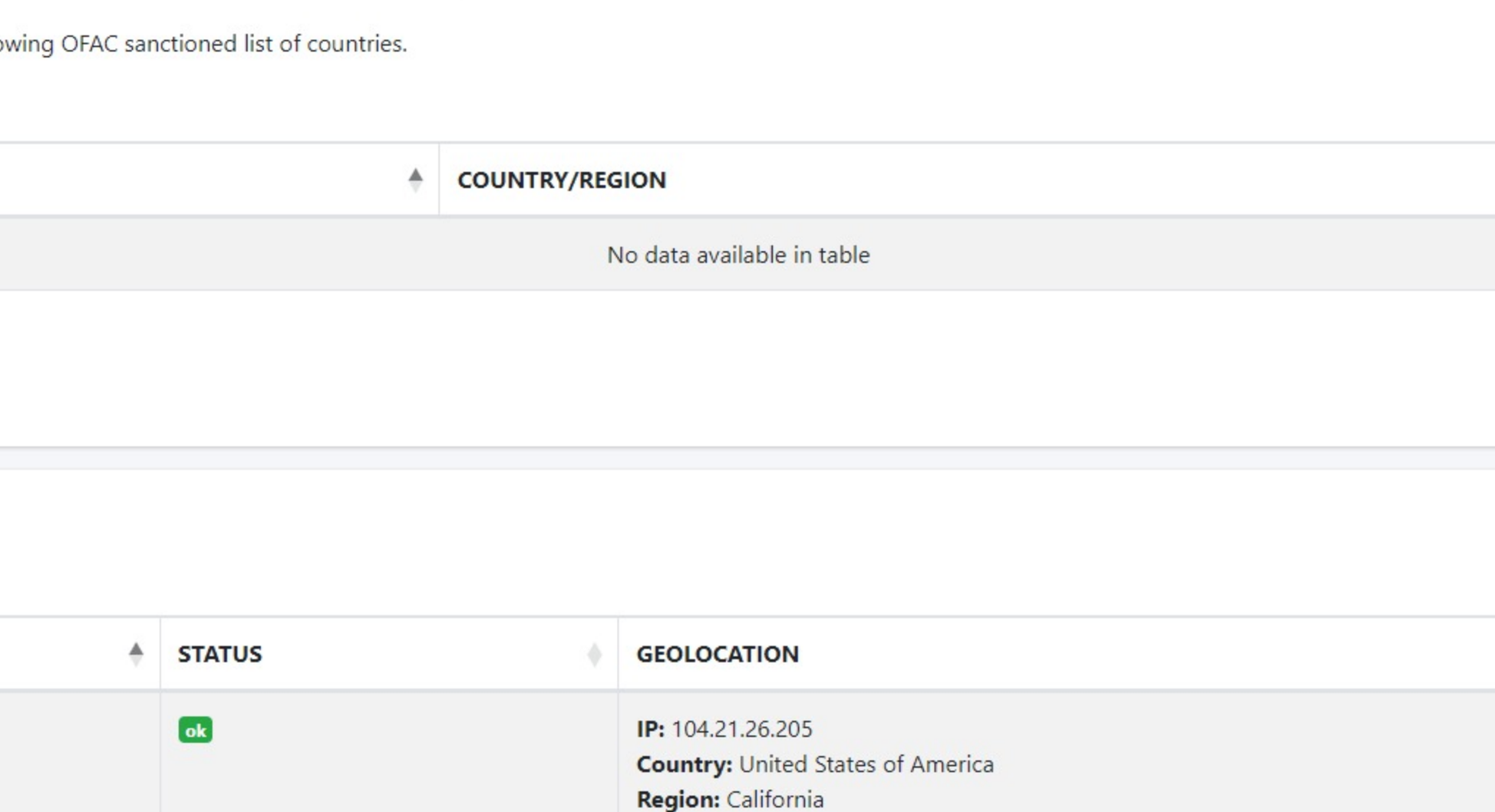
ABUSED PERMISSIONS

Top Malware Permissions 2/24
android.permission.WRITE_EXTERNAL_STORAGE, android.permission.INTERNET

Other Common Permissions 0/45

Malware Permissions are the top permissions that are widely abused by known malware.
Other Common Permissions are permissions that are commonly abused by known malware.

SERVER LOCATIONS



This app may communicate with the following OFAC sanctioned list of countries.

Search:

DOMAIN	COUNTRY/REGION
No data available in table	

Showing 0 to 0 of 0 entries
Previous Next

DOMAIN MALWARE CHECK

Search:

DOMAIN	STATUS	GEOLOCATION
ptilahore.com	ok	**IP:** 104.21.26.205 **Country:** United States of America **Region:** California **City:** San Francisco **Latitude:** 37.775700 **Longitude:** -122.395203 **View:** Google Map

Showing 1 to 1 of 1 entries
Previous 1 Next

URLS

Search:

URL	FILE
http://localhost/	retrofit2/Response.java
https://ptilahore.com/	com/rapidzz/gbvoters/model/Source/RetrofitClientInstance.java

Showing 1 to 2 of 2 entries
Previous 1 Next

FIREBASE DATABASE

EMAILS

TRACKERS

Search:

TRACKER NAME	CATEGORIES	URL
No data available in table		

Showing 0 to 0 of 0 entries
Previous Next

POSSIBLE HARDCODED SECRETS

▼ Showing all **5** secrets
79EAC9D0-BAF9-11CE-8C82-00AA004BA90B
79EAC9E0-BAF9-11CE-8C82-00AA004BA90B
258EAFA5-E914-47DA-95CA-C5AB0DC85B11
596d815750e394ff2ea553da32506363
08a6a5044d20aacadddb2e6a7009ecbf

STRINGS

From APK Resource

▶ Show all **4863** strings

From Code

▶ Show all **12609** strings

From Shared Objects

ACTIVITIES

▼ Showing all **8** activities
com.rapidzz.gbvoters.activities.MainActivity
com.rapidzz.gbvoters.activities.SplashActivity
com.rapidzz.gbvoters.activities.SearchResultActivity
com.rapidzz.gbvoters.activities.HomeActivity
com.rapidzz.gbvoters.activities.PollingStationsActivity
com.rapidzz.gbvoters.activities.VoterActivity
com.rapidzz.gbvoters.activities.ConstituencyActivity
com.rapidzz.gbvoters.activities.LoginActivity

SERVICES

▼ Showing all **1** services
androidx.room.MultiInstanceInvalidationService

RECEIVERS

PROVIDERS

LIBRARIES

FILES

▶ Show all **880** files

MOBIGUARD

SHIELDING YOUR MOBILE APPS

PROJECT DEVELOPED BY

MUZAFFAR ALI | NASIR SHARIF | TALHA FIAZ

ANDROID STATIC ANALYSIS REPORT

GB-Elections (1.0.0)

File Name:	GB-Elections(V-1.0.6).apk
Package Name:	com.rapidzz.gbvoters
Scan Date:	Oct. 3, 2024, 3:39 p.m.
App Security Score:	35/100 (HIGH RISK)
Grade:	C

FINDINGS SEVERITY

HIGH	MEDIUM	INFO	SECURE	HOTSPOT
6	3	2	2	1

FILE INFORMATION

File Name: GB-Elections(V-1.0.6).apk
Size: 41.21MB

MD5: a05dafa2869bd65b5725df73ea7c08fa
SHA1: 4edcc9ed95cc0ecb589c60d491f9261dd0049bbe
SHA256: 1a9b27589211e96ec3150c98aa1be7f0cb4ca8261fd3c567ac25ac904af22185

APP INFORMATION

App Name: GB-Elections
Package Name: com.rapidzz.gbvoters
Main Activity: com.rapidzz.gbvoters.activities.SplashActivity
Target SDK: 29
Min SDK: 15
Max SDK:
Android Version Name: 1.0.0
Android Version Code: 1

APP COMPONENTS

Activities: 8
Services: 1
Receivers: 0
Providers: 0
Exported Activities: 0
Exported Services: 0
Exported Receivers: 0
Exported Providers: 0

CERTIFICATE INFORMATION

Binary is signed
v1 signature: True
v2 signature: True
v3 signature: False
v4 signature: False

X.509 Subject: CN=Android Debug, O=Android, C=US
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2019-10-15 06:08:22+00:00
Valid To: 2049-10-07 06:08:22+00:00
Issuer: CN=Android Debug, O=Android, C=US
Serial Number: 0x1
Hash Algorithm: sha1
md5: b62e66d9b65ac69394912fc657c0f27d
sha1: 7012d6c3639140052076cf0b782e9d91e06d31a6
sha256: cca454b5c15c75bebe6d83408cef0fb8de33193791153b570d7922c3411a8ce8
sha512: 4aef2b53d2bec2bca9d2ad13b55a0e952f901cfe8ea2412c7787627e29f1c8a8edacf2cb653f7185670273b090f7f499de40821fa30854ddeaefeaaf5937e176
PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: e6904222047a0862e28c780d2bdee8fa59ac79a7e56c2baaa6fe787af9a47462
Found 1 unique certificates

APPLICATION PERMISSIONS

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents	Allows an application to write to external storage.
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.

APKID ANALYSIS

FILE	DETAILS

<table>
<tr><th>FILE</th><th>DETAILS</th></tr>
<tr><td>classes.dex</td><td>
<table>
<tr><th>FINDINGS</th><th>DETAILS</th></tr>
<tr><td>Compiler</td><td>r8</td></tr>
</table>
</td></tr>
<tr><td>classes2.dex</td><td>
<table>
<tr><th>FINDINGS</th><th>DETAILS</th></tr>
<tr><td>Anti-VM Code</td><td>Build.FINGERPRINT check
Build.MODEL check
Build.MANUFACTURER check
Build.PRODUCT check</td></tr>
<tr><td>Compiler</td><td>r8 without marker (suspicious)</td></tr>
</table>
</td></tr>
</table>

NETWORK SECURITY

NO	SCOPE	SEVERITY	DESCRIPTION

CERTIFICATE ANALYSIS

HIGH: 2 | WARNING: 1 | INFO: 1

TITLE	SEVERITY	DESCRIPTION
Signed Application	info	Application is signed with a code signing certificate
Application vulnerable to Janus Vulnerability	warning	Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android 5.0-8.0, if signed only with v1 signature scheme. Applications running on Android 5.0-7.0 signed with v1, and v2/v3 scheme is also vulnerable.
Application signed with debug certificate	high	Application signed with a debug certificate. Production application must not be shipped with a debug certificate.
Certificate algorithm vulnerable to hash collision	high	Application is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues.

MANIFEST ANALYSIS

HIGH: **3** | WARNING: **0** | INFO: **0** | SUPPRESSED: **0**

NO	ISSUE	SEVERITY	DESCRIPTION
1	App can be installed on a vulnerable upatched Android version Android 4.0.3-4.0.4, [minSdk=15]	high	This application can be installed on an older version of android that has multiple unfixed vulnerabilities. These devices won't receive reasonable security updates from Google. Support an Android version => 10, API 29 to receive reasonable security updates.
2	Clear text traffic is Enabled For App [android:usesCleartextTraffic=true]	high	The app intends to use cleartext network traffic, such as cleartext HTTP, FTP stacks, DownloadManager, and MediaPlayer. The default value for apps that target API level 27 or lower is "true". Apps that target API level 28 or higher default to "false". The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering; a network attacker can eavesdrop on transmitted data and also modify it without being detected.

NO	ISSUE	SEVERITY	DESCRIPTION
3	Debug Enabled For App [android:debuggable=true]	high	Debugging was enabled on the app which makes it easier for reverse engineers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes.

</> CODE ANALYSIS

HIGH: 1 | WARNING: 1 | INFO: 2 | SECURE: 1 | SUPPRESSED: 0

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	CWE: CWE-312: Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE-14	com/rapidzz/gbvoters/model/ResponseModels/User.java com/rapidzz/gbvoters/model/Source/AppConstants.java
2	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	com/edittextpicker/aliazaz/EditTextPicker.java com/rapidzz/gbvoters/activities/MainActivity.java com/rapidzz/gbvoters/model/Source/Repository/DataRepository.java
3	App can write to App Directory. Sensitive Information should be encrypted.	info	CWE: CWE-276: Incorrect Default Permissions OWASP MASVS: MSTG-STORAGE-14	com/rapidzz/gbvoters/Utils/SessionManager.java
4	Debug configuration enabled. Production builds must not be debuggable.	high	CWE: CWE-919: Weaknesses in Mobile Applications OWASP Top 10: M1: Improper Platform Usage OWASP MASVS: MSTG-RESILIENCE-2	com/rapidzz/gbvoters/BuildConfig.java

NO	ISSUE	SEVERITY	STANDARDS	FILES
5	This App uses SSL certificate pinning to detect or prevent MITM attacks in secure communication channel.	secure	OWASP MASVS: MSTG-NETWORK-4	com/rapidzz/gbvoters/model/Source/RetrofitClientInstance.java

NIAP ANALYSIS v1.3

NO	IDENTIFIER	REQUIREMENT	FEATURE	DESCRIPTION

ABUSED PERMISSIONS

TYPE	MATCHES	PERMISSIONS
Malware Permissions	2/24	android.permission.WRITE_EXTERNAL_STORAGE, android.permission.INTERNET
Other Common Permissions	0/45	

Malware Permissions:
Top permissions that are widely abused by known malware.
Other Common Permissions:
Permissions that are commonly abused by known malware.

OFAC SANCTIONED COUNTRIES

This app may communicate with the following OFAC sanctioned list of countries.

DOMAIN	COUNTRY/REGION

DOMAIN MALWARE CHECK

DOMAIN	STATUS	GEOLOCATION
ptilahore.com	ok	**IP:** 104.21.26.205 **Country:** United States of America **Region:** California **City:** San Francisco **Latitude:** 37.775700 **Longitude:** -122.395203 **View:** Google Map

HARDCODED SECRETS

POSSIBLE SECRETS
79EAC9D0-BAF9-11CE-8C82-00AA004BA90B
79EAC9E0-BAF9-11CE-8C82-00AA004BA90B
258EAFA5-E914-47DA-95CA-C5AB0DC85B11
596d815750e394ff2ea553da32506363
08a6a5044d20aacadddb2e6a7009ecbf

SCAN LOGS

Timestamp	Event	Error
2024-10-03 15:39:45	Generating Hashes	OK
2024-10-03 15:39:45	Extracting APK	OK
2024-10-03 15:39:45	Unzipping	OK
2024-10-03 15:39:47	Getting Hardcoded Certificates/Keystores	OK
2024-10-03 15:39:53	Parsing AndroidManifest.xml	OK
2024-10-03 15:39:53	Parsing APK with androguard	OK
2024-10-03 15:39:54	Extracting Manifest Data	OK
2024-10-03 15:39:54	Performing Static Analysis on: GB-Elections (com.rapidzz.gbvoters)	OK
2024-10-03 15:39:54	Fetching Details from Play Store: com.rapidzz.gbvoters	OK

2024-10-03 15:39:55	Manifest Analysis Started	OK
2024-10-03 15:39:55	Checking for Malware Permissions	OK
2024-10-03 15:39:55	Fetching icon path	OK
2024-10-03 15:39:55	Library Binary Analysis Started	OK
2024-10-03 15:39:55	Reading Code Signing Certificate	OK
2024-10-03 15:39:57	Running APKiD 2.1.5	OK
2024-10-03 15:40:01	Updating Trackers Database....	OK
2024-10-03 15:40:01	Detecting Trackers	OK
2024-10-03 15:40:03	Decompiling APK to Java with jadx	OK
2024-10-03 15:40:27	Converting DEX to Smali	OK
2024-10-03 15:40:27	Code Analysis Started on - java_source	OK

2024-10-03 15:40:31	Android SAST Completed	OK
2024-10-03 15:40:31	Android API Analysis Started	OK
2024-10-03 15:40:37	Android Permission Mapping Started	OK
2024-10-03 15:40:40	Android Permission Mapping Completed	OK
2024-10-03 15:40:40	Finished Code Analysis, Email and URL Extraction	OK
2024-10-03 15:40:40	Extracting String data from APK	OK
2024-10-03 15:40:40	Extracting String data from Code	OK
2024-10-03 15:40:40	Extracting String values and entropies from Code	OK
2024-10-03 15:40:42	Performing Malware check on extracted domains	OK
2024-10-03 15:40:44	Saving to Database	OK

Mobile Guarding Security Framework (MOBIGUAD) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.