VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI, KARNATAKA-590 018

A Research Internship

Submitted in partial fulfillment of the requirements for the award of Degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIECNE AND ENGINEERING

by

NISHA

USN: 4JK21CS040

Internship Carried out

at

NITK Surathkal

Internal Guide

Dr. Vedavati Bhandari
Assistant Professor
Dept of CSE, AJIET, Mangaluru

External Guide

Mrs. Radhika B S
Assistant Professor
Dept of CSE,NITK,Surathkal

Department of Computer Science and Engineering
A. J. INSTITUTE OF ENGINEERING AND TECHNOLOGY
(A unit of Laxmi Memorial Education Trust®)
NH-66, Kottara Chowki, Mangaluru – 575006
A.Y: 2024-2025

A. J. INSTITUTE OF ENGINEERING AND TECHNOLOGY

(AUNIT OF LAXMI MEMORIAL EDUCATION TRUST®)

NH-66,KOTTARA CHOWKI,MANGALURU -575006.

DEPARTMENT OF COMPUTER SCIECNE AND ENGINEERING

CERTIFICATE

Certified that the **Research Internship** report submitted by **NISHA**, USN: **4JK21CS040**, a bonafide student of **A. J. Institute of Engineering and Technology** in partial fulfillment for the award of Bachelor of Engineering in Mechanical Engineering of the **Visvesvaraya Technological University, Belagavi** during the year **2024-2025**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report and deposited in the departmental library. The Internship report has been approved as it satisfies the academic requirements in respect of Internship prescribed for the said degree.

Dr. Vedavati Bhandari
Internal Guide

Mrs. Ashwitha A Shetty
Internship coordinator

Dr. Antony PJ
Vice Principal and HOD

Dr. Shantharama Rai C
Principal

Name of the Examiner Signature with Date

1.

2.

ACKNOWLEDGEMENT

First and foremost, I thank my parents for what we are and where I am today, without whose hard work and sacrifice I would not be here today.

Hearty thanks to **Dr. Radhika B S,** Assistant Professor**,** NITK Surathkal for providing all the facilities that helped me in timely completion of Internship work.

Heartly thanks to **Dr. Vedavati Bhandari**, Assistant Professor, Computer Science and Engineering, for her support and encouragement.

I consider it a privilege to express my heartfelt gratitude to **Mrs. Ashwitha A Shetty**, my esteemed supervisor and Assistant Professor, Computer Science and Engineering, for her invaluable guidance and unwavering support throughout every phase of my internship.

I am grateful to **Dr Antony PJ**, Vice Principal Head of the Department, Computer Science and Engineering for his support and encouragement.

I acknowledge the continuous support extended by the Dean **Dr. P Mahabaleshwarappa** for the completion of Internship work.

I would like to take this opportunity to express our gratitude to the principal **Dr. Shantharama Rai C**, for giving me this opportunity to enrich our knowledge.

I have a great sense of pleasure in expressing our deep sense of gratitude to our respected President **Dr**. **A. J. Shetty** and respected Vice President **Mr. Prashanth Shetty** for having provided us with a great infrastructure and well-furnished labs.

Finally, I would like to thank all the teaching and non-teaching staff of Department of Computer Science Engineering for their valuable help and support.

NISHA

[4JK21CS040]

TABLE OF CONTENTS

Chapter	Title	Page No.
	ACKNOWLEDGEMENT	i
	TABLE OF CONTENTS	ii-iii
	LIST OF FIGURES	iv
Chapter 1	**INTRODUCTION**	**1-4**
1.1	About Research Internship	1
1.2	Objectives of Research Internship	1-2
1.3	Scope of Research Internship	2
1.4	Benefits of Research Internship	3
1.5	Organization of Report	3-4
Chapter 2	**TASK PERFORMED**	**5-32**
2.1	Understanding Android Forensics and Setting up Environment	5-7
2.2	Deep Dive into Android Architecture	8-9
2.3	Exploring Android File Systems	10-11
2.4	Investigating Data Types on Android Devices	12-14
2.5	Understanding Android Security and Anti-Forensics Techniques	15
2.6	Theoretical Study of Volatile Memory Forensics	16
2.7	Study and Analysis of Android Forensics Automator	17-18
2.8	Automated application and Permission Management	18-20
2.9	Simulating User Interactions	20-22
2.10	Understanding the Path Detector in AnForA	23-24

2.11	Challenges faced by the Retriever during Data Recovery	25-26
2.12	Overview of Difference Extractor for APK analysis	27-28
2.13	Structured Presentation of Analysis Results	29
2.14	Frontend Styling and User Interface	30
2.15	Overall Presentation and Development of the Project	31
Chapter 3	**REFLECTION**	**32-33**
Chapter 4	**CONCLUSION**	**34**
	REFERENCES	**v**

LIST OF FIGURES

Fig No	Tittle	Page No.
2.1.1	Installation of Android Studio	6
2.1.2	Launched Virtual Device	7
2.2.1	Android Architecture	8
2.3.1	Android File System	10
2.4.1	Extraction of Data	13
2.4.2	Database	14
2.7.1	AnForA Structure	18
2.8.1	Installing APK File	19
2.8.2	Granted Permission for APK files	20
2.9.1	Executing The Action Script	21
2.9.2	Successful Execution of the Script	22
2.10.1	Analyzation of APK file	23
2.10.2	Analysed Paths	24
2.11.1	Selecting the file to retrieve	25
2.11.2	Retrieved Snapshot	26
2.12.1	Comparing the pulled Snapshots	27
2.12.2	Difference Found	28
2.14.1	Home Page	30

CHAPTER 1

INTRODUCTION

internships are educational and career development opportunities, providing practical experience in a field or discipline. They are structured, short-term, supervised placements often focused around particular tasks or projects with defined timescales. An internship may be compensated, non-compensated, or sometimes partially paid. The internship has to be meaningful and mutually beneficial to both the intern and the organization. It is important that the objectives and the activities of the internship program are clearly defined and understood. Internships also help students bridge the gap between theoretical knowledge and real-world applications, enhancing their professional competence.

1.1 About the Research Internship

A Research Internship or Industrial Internship provides students with the opportunity to apply academic concepts to real-world projects in a professional setting. A Research Internship primarily focuses on academic research, innovation, and the development of new knowledge under the guidance of experts, often within universities or research organizations. In contrast, an Industrial Internship is more industry-oriented, where students work within companies to gain practical exposure to organizational processes, technologies, and professional practices. Both types of internships aim to enhance the intern's technical, analytical, and professional skills, preparing them for future academic pursuits or industry roles. These internships serve as a bridge between theoretical learning and practical application, offering invaluable insights into workplace expectations and career pathways.

1.2 Objectives of the Research Internship

The Research Internship / Industrial Internship is designed to provide students with valuable practical exposure, allowing them to apply academic knowledge to real-world problems. It helps students develop technical expertise, research skills, and professional competencies required for their future careers. Through hands-on experience, students gain a better understanding of industry standards, organizational structures, research methodologies, and emerging technologies. The internship aims to bridge the gap between theoretical learning and practical application while also promoting critical thinking, innovation, and ethical professionalism and internship must be organized.

Following are the intended objectives of the Research Internship :

1. Provide students with practical experience in applying theoretical concepts to real-world projects or research activities.
2. Enhance understanding of technical skills, research techniques, and industrial practices relevant to the field of study.
3. Develop problem-solving abilities, critical thinking, and innovative approaches.
4. Familiarize students with workplace environments, project management processes, and organizational structures.
5. Encourage professional conduct, teamwork, leadership, and effective communication skills.
6. Expose students to the latest technological advancements and industry or research trends.
7. Bridge the gap between academic education and professional requirements through practical engagement.
8. Build skills in technical report writing, documentation, and presentation of work.
9. Promote awareness of professional ethics, responsibilities, and safety standards.
10. Create opportunities for networking, career development, and employment readiness.

1.3 Scope of the Research Internship

The Research Internship / Industrial Internship plays a vital role in enhancing the practical knowledge and professional skills of students by exposing them to real-world working environments. It provides an opportunity to experience and understand research processes, industrial workflows, technical operations, and management practices. The internship allows students to apply theoretical knowledge to practical challenges, understand current industry and research trends, and develop the competencies required for future academic or professional success. while also promoting critical thinking, innovation, and ethical professionalism and internship must be organized.

It also fosters the development of soft skills, such as communication, teamwork, problem-solving, and ethical responsibility, essential for effective professional practice.

The scope of the Research Internship / Industrial Internship includes:

1. Applying academic knowledge to real-world industrial projects and research activities.
2. Gaining hands-on experience with modern tools, technologies, and professional practices.
3. Understanding industrial/research workflows, organizational structures, and management systems.
4. Enhancing communication skills, teamwork, leadership, and professionalism.
5. Preparing students for higher studies, career opportunities, and real-world challenges.

1.4 Benefits of the Research Internship

- An opportunity to get hired by the industry/ organization.
- Practical experience in an organizational setting.
- Understand how theoretical knowledge is applied to real-world scenarios.
- Helps them to decide if the industry and the profession is the best career option to purse.
- Opportunity to learn new skills and supplement knowledge.
- Opportunity to practice communication and teamwork skills.
- Opportunity to learn strategies like time management, multi-tasking etc. in an industrial setup.
- Opportunity to meet new people and learn networking skills.
- Makes a valuable addition to their resume.
- Enhances their candidacy for higher education.
- Creating networks, social circle and developing relationships with industry people.
- Provides opportunity to evaluate the organization before committing to a full-time.
- Gain real work experience and provide meaningful assistance to the company.
- Build industrial relations and improve institutional credibility & branding.
- Makes the placement process easier.

1.5 Organization of Report

An internship report typically follows a specific organizational structure to the ensure that all important information is included and presented in a clear, logical, and professional manner.

The structure adopted for this report includes the following sections:

- **Introduction:**

  This section provides a brief overview of the internship, including the duration, location, area of work, objectives, and the motivation behind undertaking the internship. It sets the context for the report.

- **Background of the Organization:**

  Detailed information is about the host of the organization, including its history, mission, vision, core activities, organizational structure, key products/services, and its role within the industry or sector. This helps in understanding the environment in which the internship was conducted.

- **Duties and Responsibilities:**

  A description of the roles were assigned during the internship. It outlines the needed daily tasks and special assignments, participation in meetings, projects contributions and in which jobs are undertaken team-based projects.

- **Work Experience and Contributions:**
  A comprehensive account of the work performed, skills applied, techniques and tools used, and the impact of the work done.
- **Skills and Knowledge Acquired:**
  An evaluation of the technical, professional, and soft skills were gained during the internship. It discusses how these skills have enhanced the intern's competence and their relevance to future academic and professional endeavours.
- **Challenges Faced and Solutions Implemented:**
  A critical discussion of the problems were encountered during the internship period. It highlights problem-solving strategies, adaptability, teamwork, and innovative thinking applied to overcome obstacles.
- **Reflection on the Internship Experience:**
  A personal reflection on the overall internship experience, focusing on lessons learned, strengths identified, areas for improvement, and career path clarification. It also includes self-assessment on professional growth.
- **Conclusion:**
  A summary highlighting the major experiences, outcomes, and key learnings from the internship. This section wraps up the report with final thoughts and insights.
- **Recommendations:**
  Suggestions for future interns on maximizing their internship experience and recommendations for the host organization to improve its internship program. It could also suggest improvements in project processes or organizational activities observed during the internship.
- **Appendices:**
  Supporting documents such as charts, tables, photographs, certificates, project snapshots and a supplementary material referenced in the report are included in this section.
- **References:**
  A list of books, articles, websites, organizational documents, and other sources consulted while preparing the report, properly cited to maintain academic integrity.

CHAPTER 2

TASK PERFORMED

Android forensics is a critical branch of digital forensics that focuses on the extraction, preservation, and analysis of data from Android devices. With smartphones playing a central role in daily life, Android devices often store important information such as SMS messages, call logs, app data, browsing history, location details, and even deleted files, making them valuable sources of evidence in investigations. The primary goal in Android forensic tasks is to recover and analyse this data without altering or damaging the original evidence, ensuring it remains admissible in legal or investigative contexts. Techniques commonly used include accessing devices through Android Debug Bridge (ADB), creating forensic images, analysing app storage, recovering deleted data, and studying system logs. A strong understanding of the Android operating system, file structures, security mechanisms, and application behaviour is essential to perform effective forensic analysis. Tasks performed during Android forensics may involve extracting call records, SMS content, application-specific data, hidden or private files, and attempting to recover deleted items from internal storage or emulators. Specialized tools and custom scripts are often deployed to automate and validate the data collection process. Overall, Android forensics plays a vital role in uncovering critical evidence, supporting investigations, solving cybercrimes, and enhancing mobile security research.

2.1 Understanding Android forensics and Setting up Environment (Week 1)

In the initial week, we focused on understanding what Android forensics involves, including the legal and ethical aspects of mobile evidence handling. A forensic investigator must ensure that extracted data remains untampered and legally admissible. We explored the fundamental differences between digital evidence and physical evidence, emphasizing the importance of maintaining a strict chain of custody and following forensically sound procedures. Key forensic principles such as maintaining evidence integrity and legal admissibility were emphasized. Additionally, we examined the core principles of forensic readiness and analyze mobile data without compromising its integrity we also discussed how digital evidence is different from traditional evidence and why a forensic sound approach is necessary.

2.1.1 Android Studio Installation Process:

To simulate real-world mobile forensic scenarios, setting up an Android development and testing environment was critical. The steps followed for Android Studio installation were:

- Downloading Android Studio**:** The official installer was downloaded from the Android Developers website.

- Installation**:** We executed the installer package and followed the setup wizard, which installed Android Studio along with the necessary SDK tools and the Android Virtual Device (AVD) Manager.
- First Launch Configuration**:** On first launch, we configured the IDE by setting up the SDK path, accepting license agreements, and downloading essential components as shown in fig 2.1.1, like platform tools and build tools.

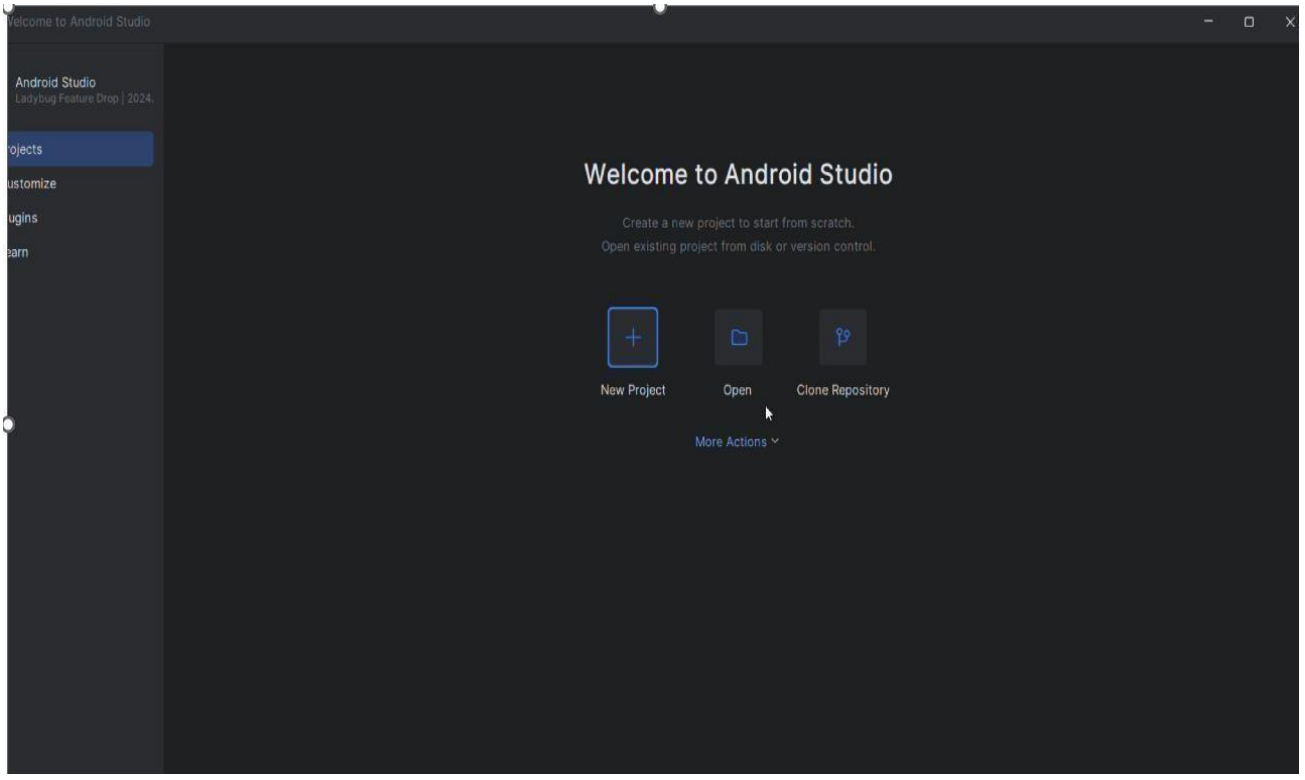

Fig 2.1.1 Installation of Android studio

2.1.2 Emulator Creation and Configuration:

After installing Android Studio, we proceeded to create an Android Virtual Device (AVD) to simulate mobile environments:

- Accessing AVD Manager: From Android Studio's toolbar, we opened the AVD Manager.
- Creating a New Virtual Device: We selected a device model (e.g., Pixel 4) and chose a system image (such as Android 11 or 12) suitable for testing.
- Emulator Settings**:** Settings like device orientation, RAM size, and resolution were customized for better performance.
- Launching the Emulator: The AVD was successfully launched as shown fig 2.1.2, providing a fully functional Android environment to install APKs, create dummy data (like SMS, call logs), and perform forensic tasks.

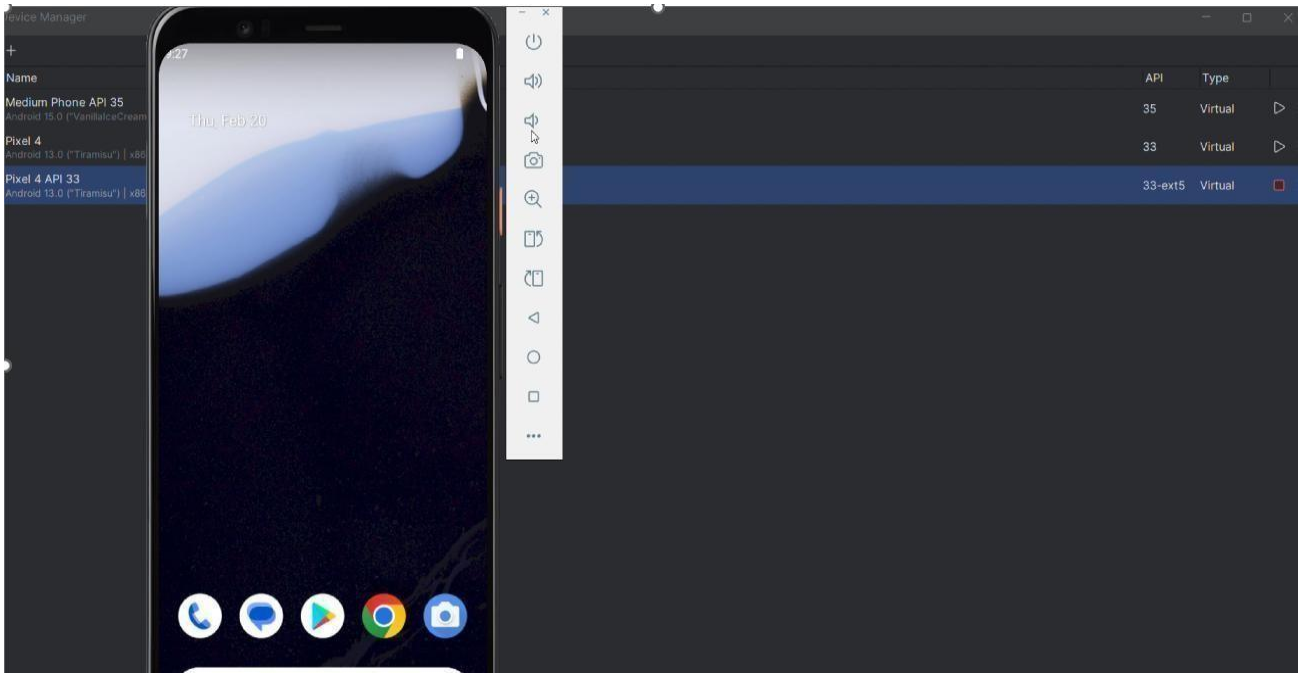

Fig 2.1.2 Launching of Virtual Device

2.1.3 ADB (Android Debug Bridge):

ADB was introduced as a command-line tool that enables direct communication with an Android emulator or a connected device. To begin, Developer Options were enabled, and USB Debugging was turned on within the emulator. Using the adb devices command, the list of connected devices and emulators was verified. Basic commands such as adb shell, adb pull, and adb logcat were practiced to understand how data can be remotely accessed and extracted from the device.

Through adb shell, we explored the internal file system of the device, gaining access to directories and system files that are otherwise hidden from regular users. The adb pull command allowed us to retrieve important files and artifacts from specific locations, demonstrating the tool's effectiveness in remote data acquisition. Meanwhile, adb logcat provided real-time logs of system events and app activities, showcasing its value in dynamic forensic analysis and debugging scenarios.

Understanding the capabilities and limitations of ADB is vital for forensic investigations, as it offers a non-intrusive method to collect evidence without altering the device significantly. It also introduced us to the concept of securing extraction, deleted file recovery, and dynamic analysis of apps. The knowledge of ADB operations will serve as a critical skill when handling real-world investigations. extractions in a legally sound manner, preparing us for more advanced forensic techniques. As we progress, this knowledge will enable us to tackle complex forensic challenges, uphold chain-of-custody standards, and contribute meaningfully to professional investigative environments.

2.2 Deep Drive into Android Architecture (Week 2)

Android operating system follows a layered architecture, each layer providing specific services and functionalities critical to device operation. At the base is the Linux Kernel, which acts as the heart of Android. It is responsible for core services like device drivers, memory management, process management, and power management. For forensic investigators, the kernel is important because it governs how files are accessed, how device permissions are handled, and how secure communication between applications and hardware takes place. Since Android devices depend heavily on the underlying Linux Kernel for hardware communication, understanding this layer provides insight into how data is physically stored and managed on the device.

Above the kernel lies the Native Libraries and Android Runtime (ART) layers. Native libraries include important libraries like WebKit (web rendering), OpenGL (graphics), SQLite (database storage), and SSL (network security). These libraries provide essential services to applications. Forensics investigators often find critical evidence stored in SQLite databases, such as chat histories, app settings, or cached information. The Android Runtime (ART) replaced the Dalvik VM in later Android versions. ART executes app code and manages memory more efficiently, using ahead-of-time (AOT) compilation. Understanding ART is important because some temporary data, logs, and compiled code structures could become part of forensic evidence.

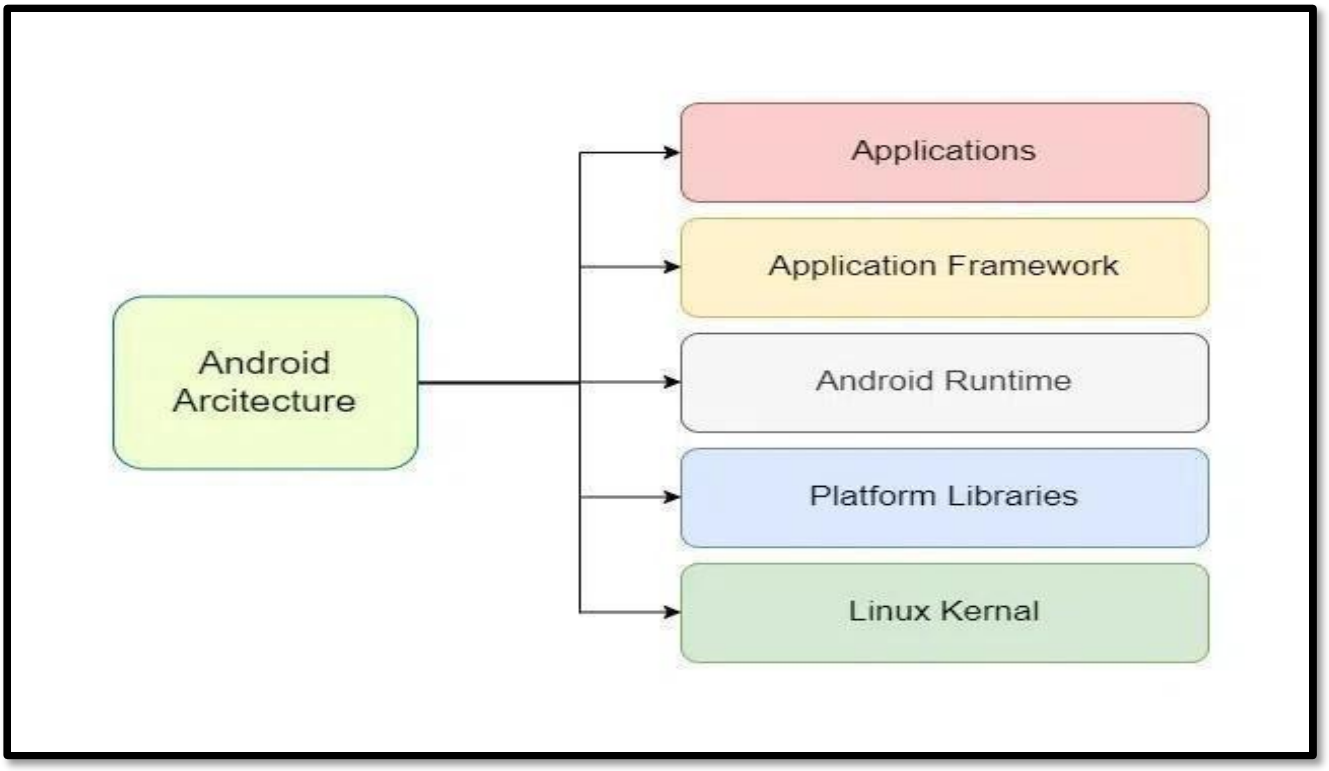

Figure 2.2.1 Android Architecture

Moving upward, we reach the Application Framework layer. This layer offers a wide range of higher-level services that developers use to build apps. Core components like Activity Manager, Window Manager, Content Providers, and Resource Manager exist here. For forensic work, the Application Framework is essential because it determines how apps interact with each other and how data is shared. For instance, Content Providers can be exploited to extract data shared between apps, and understanding Broadcast Receivers can help us track how apps respond to system events. Weaknesses or misconfigurations at this level may lead to data leaks, application lifecycle management, memory allocation, and garbage collection. From a forensic perspective, understanding ART helps in analyzing how application data is executed, cached, and stored temporarily or permanently. which forensic analysts can use to reconstruct activities or retrieve hidden data.

At the topmost layer are the Applications both system apps and user-installed apps. Pre-installed apps like Contacts, Phone, and Settings often contain valuable system-level logs and user interaction histories. Meanwhile, third-party apps (like WhatsApp, Facebook, or banking apps) store critical personal data, communication records, media, and location traces. We studied how application data is organized inside the internal storage (/data/data/<package_name>) and external storage (SD card or shared folders). Understanding where and how apps store data helped us develop strategies for logical acquisition (pulling data through backup methods) and physical acquisition (direct file system extraction).

During the hands-on activities, we worked extensively with the Android Emulator to gain a deeper understanding of how data is stored and managed within the Android operating system. A key focus was differentiating between system apps (such as Contacts, Settings, and Phone, which are part of the OS and stored in /system/app/) and user-installed apps (located in /data/app/). We examined various critical directories including:

- /data/data/ – where each app stores its private data such as databases, shared preferences, and cache.
- /sdcard/ – used for shared storage like media files and documents, which is accessible by multiple apps.
- /system/ – a read-only partition containing core OS files and built-in applications.

We installed sample APKs on the emulator and observed how permissions declared in the AndroidManifest.xml file influenced what data each app could access, such as camera, storage, or SMS. This exercise highlighted the importance of permission-based access control, which is critical when investigating how malware or unauthorized apps exploit system privileges.

2.3 Exploring Android File System (Week 3)

During the third week of the project, the primary focus was placed on exploring the internal storage architecture of Android devices, emphasizing the underlying file systems that govern data management. Android devices predominantly employ the EXT4 (Fourth Extended File System) and YAFFS2 (Yet Another Flash File System 2) formats. EXT4 is widely used in modern Android devices due to its journaling capabilities, improved data integrity, and efficient performance. In contrast, older devices, particularly those based on NAND flash storage, utilize YAFFS2, optimized specifically for flash memory reliability.

2.3.1 Technical Study of Android File Systems

An in-depth study was conducted on the file systems analysis as shown in figure 3.2.1 predominantly used within Android environments, namely:

- EXT4 (Fourth Extended File System): A journaling file system that ensures data integrity, crash resilience, and efficient space utilization. EXT4 is the standard file system for most modern Android devices.
- YAFFS2 (Yet Another Flash File System 2): A specialized file system designed for NAND flash memory, commonly found in legacy Android devices.

Understanding the technical differences between EXT4 and YAFFS2 is critical for forensic investigators working with Android devices. EXT4, being more advanced, offers better support for modern forensic techniques like logical extraction and data recovery, but presents more challenges in data recovery and evidence preservation due to its lack of journaling.

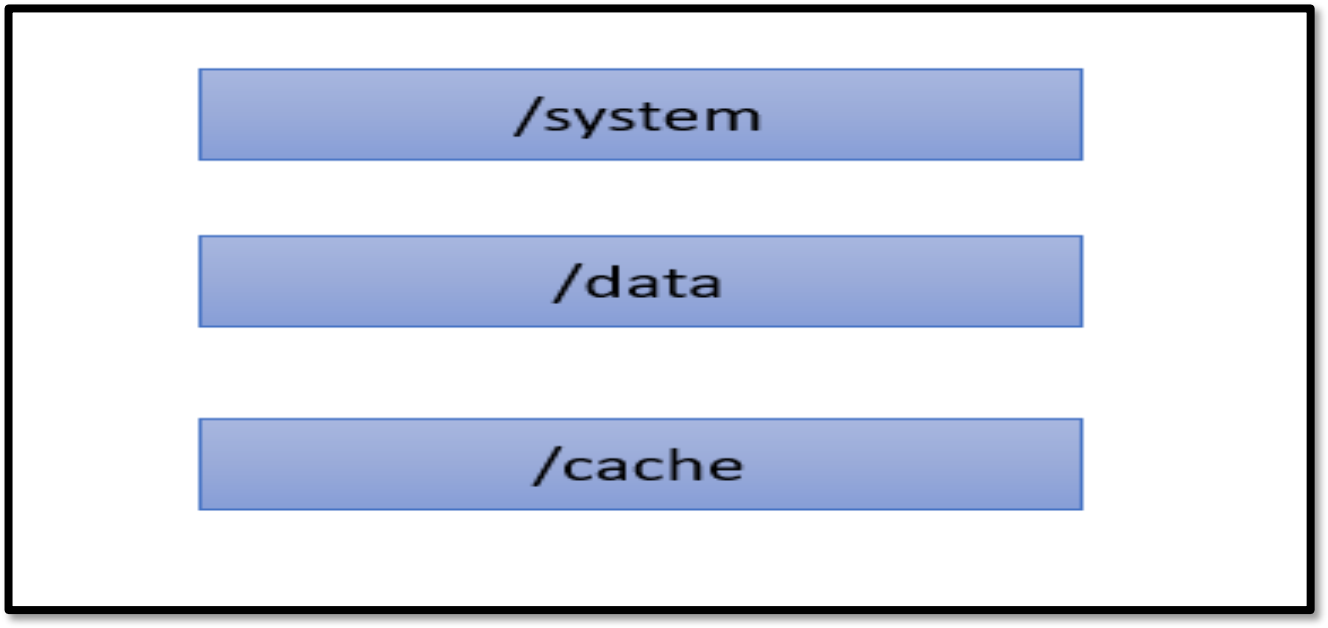

Figure 2.3.1 Android file system

2.3.2 Practical Exploration of Storage Structures

Hands-on sessions were conducted using Android Emulators configured within Android Studio, along with ADB (Android Debug Bridge) tools, to navigate internal storage components. Practical exploration of storage structures in the context of computer science typically involves learning about different data storage techniques used in systems and applications.

The key storage directories examined included:

- **/data/data/** – Housing private application data such as SQLite databases, SharedPreferences, and internal caches.
- **/storage/emulated/0/** – Representing user-accessible shared storage containing media files, downloads, and app-generated content.
- **/system/** – Storing critical operating system binaries, default apps, and supporting libraries.

The activities included:

- Enumerating application package directories,
- Locating sensitive artifacts (databases, preferences),
- Identifying the separation between user data and system data.

2.3.3 Analysis of Storage Virtualization and Symbolic Links

Detailed exploration was conducted into Android's storage abstraction model, including the use of symbolic links. It was observed that paths like /storage/emulated/0/ internally redirect to device partitions, a design that allows the system to manage user profiles and shared storage transparently. Recognizing such abstractions is essential in forensic investigations to avoid missing critical evidence during logical or physical extractions.

Key observations included:

- Dynamic Path Redirection**:** Android dynamically manages user storage spaces through symbolic links, allowing separate data directories without physically isolating them on the flash memory.
- Multi-User Profile Management**:** Each user on an Android device is assigned a virtual storage path, with the abstraction hiding the complexity from both the apps and the users.
- Unified Access for Applications**:** Applications accessing /sdcard/ or /storage/self/primary/ are automatically redirected to the appropriate emulated paths by Android’s Media Storage service.

When performing a logical extraction, which typically retrieves user-accessible files like images, videos, documents, and app-generated data, the tools often follow these symbolic links to gather data. However, during physical extractions—which involve capturing the entire raw memory of the device including unallocated space, deleted data, and hidden partitions analysts.

2.4 Analysis of Storage Virtualization and Symbolic Links (week 4)

2.4.1 Forensic artifacts on android devices

Forensic artifacts are pieces of evidence or data that can be extracted from a device during a forensic investigation. These artifacts are crucial in understanding the activity and behaviour of the user on the device. Here's a closer look at some common forensic artifacts on Android devices

- **System Logs:**

  In Android forensic investigations, system logs serve as a critical source of operational data. These logs record information about device operations, system errors, application crashes, and other significant system events. Stored typically under directories such as /data/log/, system logs provide valuable insights into device behavior leading up to a specific incident. Investigators often use tools like logcat (accessible via ADB) to extract detailed real-time or buffered logs, capturing vital evidence of user and application activity.

- **SMS (Short Message Service) and Call Logs:**

  Another important category includes SMS (Short Message Service) and call logs. SMS logs capture records of incoming and outgoing text messages, storing the sender/receiver information, message content, timestamps, and occasionally location data. Similarly, call logs contain metadata such as phone numbers, call types (missed, incoming, outgoing), call duration, and timestamps. These artifacts are usually housed in SQLite databases.

- **Contact logs**

  Contacts represent another vital evidence source, generally maintained within a centralized SQLite database.Contact entries often include names, phone numbers, email addresses as shown in the figure 2.4.1 and sometimes additional personal details such as physical addresses or birthdays. Since many Android devices sync this data with cloud investigators must assess both local storage and potential cloud backups.

- **Media files**

  Media files, including photos, videos, and audio recordings, are typically located in directories such as /DCIM/ or /Pictures/. These files not only contain visual evidence but also embed metadata (such as EXIF data) that records information about the time, date, GPS location, and the device used to capture the media. This metadata can be crucial for verifying timelines and geolocations associated with criminal activities.

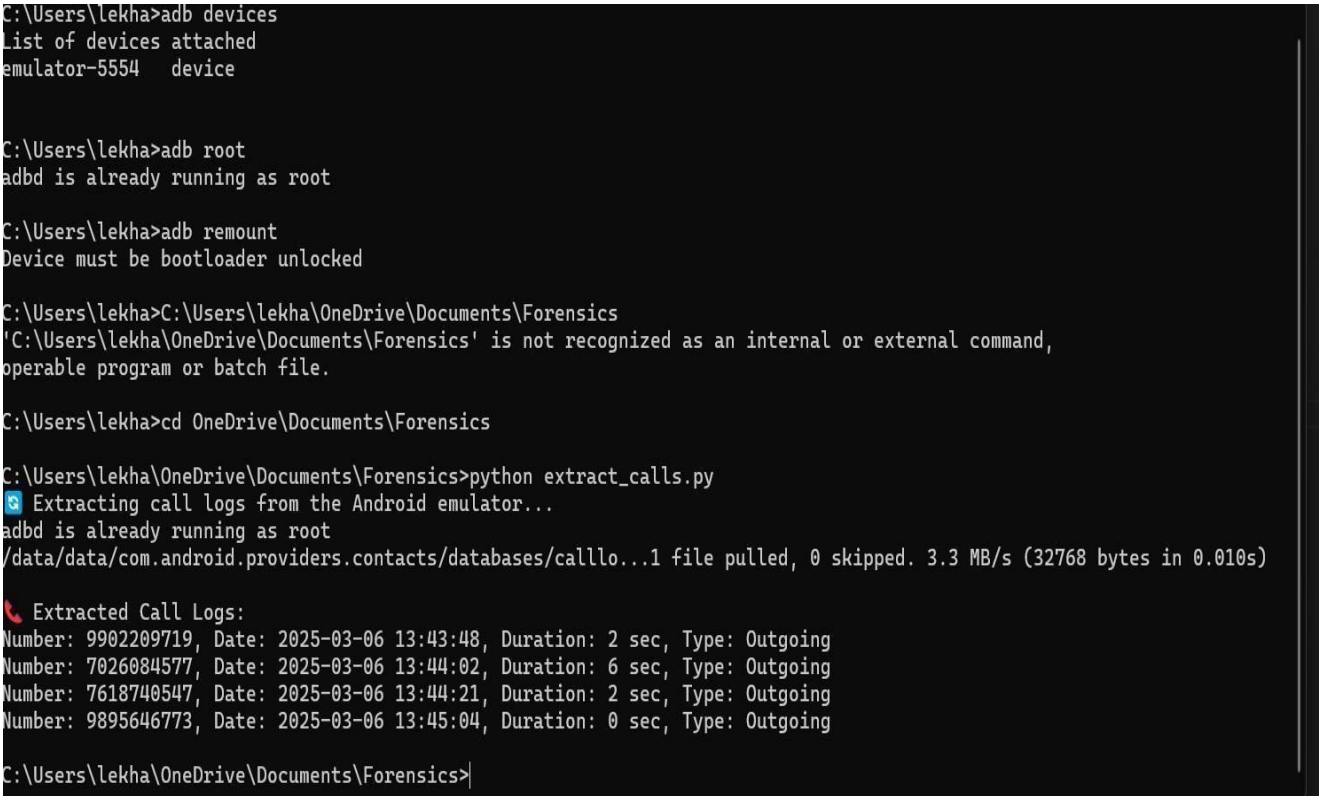

Figure 2.4.1 Extraction of Data

2.4.2 Locating and Analyzing SQLite Databases

SQLite is a lightweight, file-based database management system commonly used in Android devices to store structured data. It does not require a separate server, making it ideal for mobile applications where simplicity and efficiency are important.

- MMSSMS.db

  The mmssms.db database stores all SMS messages on an Android device. It contains important tables like sms, which holds information such as the sender/receiver number (address), the timestamp (date), and the actual message content (body). Another table, threads, organizes messages into conversation groups based on contacts. Investigators use tools like DB Browser for SQLite to open and explore this database, allowing them to view, filter, and analyze text message data effectively.

- Call.db

  The calls.db database records all call activities, including incoming, outgoing, and missed calls. Each record provides details like the phone number involved, call duration, call type (incoming, outgoing, or missed), and the timestamp of the call. Using SQL queries, forensic analysts can filter these records based on specific dates, phone numbers, or call types to better understand communication patterns over time.

- Contacts.db

  The contacts.db database contains stored contact information as shown in figure 2.4.2 on the device. It has tables like contacts (for names), phones (for associated phone numbers), and email (for email addresses). Analyzing this database provides insight into the user's personal and professional relationships.

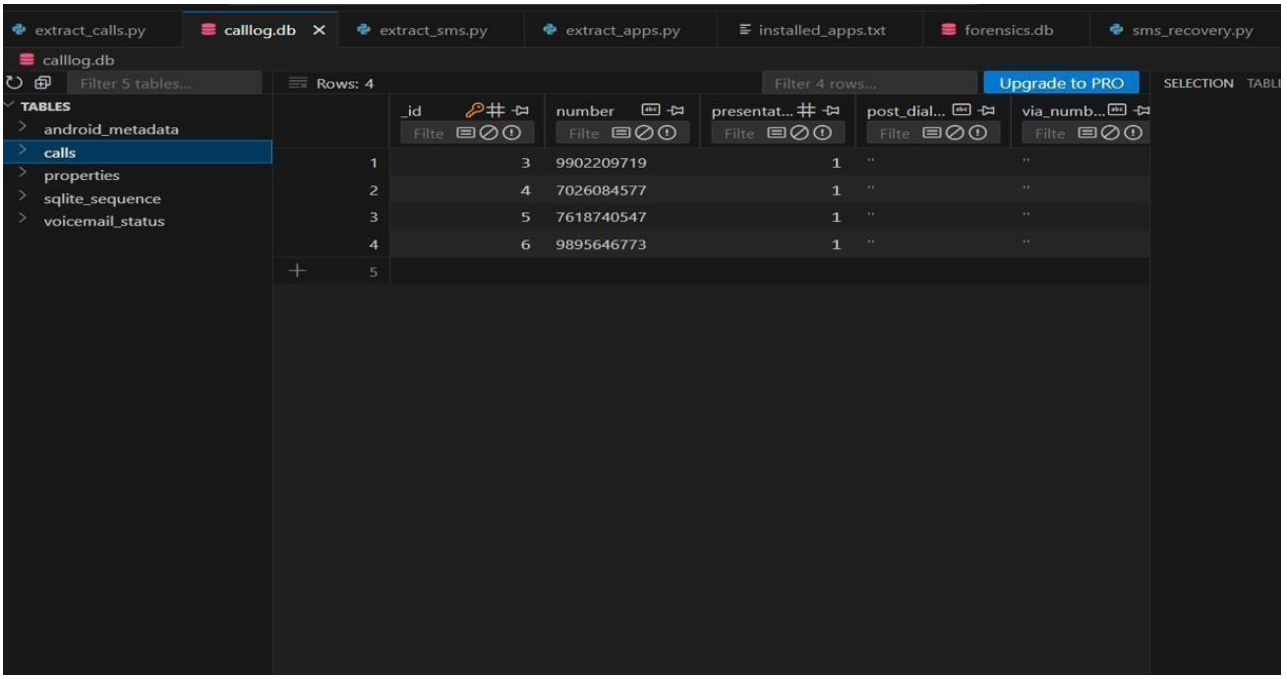

Figure 2.4.2 Database

2.4.3 Tools for Data Extraction and Analysis

- **DB Browser for SQLite:**

  DB Browser for SQLite is a graphical tool that simplifies opening and analyzing SQLite databases. It allows users to view the database structure, run SQL queries, and export data in formats like CSV or Excel for further analysis.

- **ADB (Android Debug Bridge):**

  ADB is a powerful command-line tool used to communicate with Android devices. It enables users to pull important files, such as SQLite databases, from the device to a local computer for forensic examination.

- **Rooting the Device:**

  In certain cases, rooting the Android device becomes necessary to extract forensic data. Rooting grants elevated permissions (root access), allowing forensic analysts to access protected system files and hidden directories that are normally restricted.

2.5 Understanding Android Security and Anti-Forensics Techniques (week 5)

In Week 5, we undertook a deep study of the security frameworks and protections implemented in the Android operating system. Key components such as Security-Enhanced Android (SEAndroid), Verified Boot, and Full-Disk Encryption (FDE) were examined to understand how Android ensures device integrity, user data confidentiality, and application sandboxing.

2.5.1 Android security

- Security-Enhanced

  We learned that SEAndroid extends SELinux (Security-Enhanced Linux) policies to Android. It enforces Mandatory Access Control (MAC) to limit the actions of applications and system processes, even if they become compromised. Hands-on exercises involved viewing SEAndroid logs using dmesg and examining policy violations.
- Booting

  We explored the Verified Boot process, which ensures that the device's bootloader, kernel, and system partition have not been tampered with. Practical tasks included intentionally corrupting system images in emulators and observing how devices refused to boot, preserving system integrity.
- Full-Disk

  Full-Disk Encryption ensures that user data is protected even if the device is physically compromised. We observed how modern Android devices use FDE and File-Based Encryption (FBE) where different encryption keys protect different files.

2.5.2 Anti-Forensics Techniques Studied

We also delved into anti-forensics strategies used by malicious actors to hide, destroy, or obfuscate evidence:

- Secure Data Wiping: Tested apps that perform overwriting of files and free space to prevent forensic recovery.
- File Hiding and Obfuscation: Used utilities that renamed sensitive files with misleading extensions and moved them into obscure directories.
- Encryption Apps: Explored applications that encrypt specific folders or media to hide evidence from casual file system browsing.
- Intentional Data Corruption: Practiced deliberate database corruption and partial overwriting to understand how attackers sabotage forensic analysis.

2.6 Theoretical Study of Votali Memory (week 6)

During Week 9, we focused deeply on understanding volatile memory (RAM) forensics in Android environments. Volatile memory analysis is a crucial aspect of digital forensics, as RAM holds live and active data, which often disappears when the device powers off. Through theoretical learning, we understood that capturing RAM content can reveal a range of highly sensitive information including decrypted usernames and passwords, active chat conversations, encryption keys, open web sessions, loaded applications, and unsaved documents.

We studied the challenges involved in acquiring RAM dumps, especially from Android emulators where access to low-level memory acquisition is restricted. Practical memory extraction typically demands root access or specialized kernel modules. Despite emulator limitations, we examined real-world methods like using LiME (Linux Memory Extractor) an open-source tool that enables live memory acquisition on Android devices by inserting a kernel module.

We also learned about the Volatility Framework, an advanced memory forensics tool capable of parsing RAM dumps to extract artifacts such as running processes, network connections, memory-resident files, and system call traces to address these, we studied best practices like timing the memory acquisition as early as possible during an incident and maintaining a strict chain of custody to ensure the integrity and admissibility of volatile evidence in legal proceedings.

Theoretical exercises included preparing procedures for dumping RAM, configuring modules, setting up appropriate device drivers, and using analysis plugins in Volatility (such as pslist, netscan, files can, and cmdscan) to extract forensic evidence. Special attention was paid to the importance of maintaining forensic soundness while capturing volatile memory, ensuring that the acquisition process does not alter or destroy crucial artifacts. Additionally, we discussed scenarios where volatile memory forensics becomes critical — such as during investigations into malware infections, credential theft, or insider threats. Understanding these concepts broadened our knowledge of the live analysis domain and prepared us for advanced mobile forensic tasks in real-world settings.

Another significant aspect covered was the categorization of artifacts retrievable from volatile memory, including process memory maps, clipboard contents, encryption keys for protected files, and fragments of instant messaging conversations. We learned how advanced malware often resides only in RAM to evade detection, making memory forensics indispensable for uncovering sophisticated threats. Additionally, we explored the challenges posed by modern Android security measures such as memory encryption and Address Space Layout Randomization (ASLR), which complicate traditional acquisition and analysis methods.

2.7 Study and Analyses of Android Forensics Automator (week 7)

As part of seventh week academic activities, I conducted an in-depth review of the research paper titled "AnForA: Automating Android Application Forensic Analysis." This study greatly enhanced my understanding of the challenges and solutions related to the forensic examination of Android applications. Manual forensic processes are often time-consuming, error-prone, and lack repeatability. In contrast, AnForA presents a dynamic, automated approach that significantly improves the reliability and scalability of mobile forensic investigations.

AnForA operates by simulating real user interactions on Android applications installed in Android Virtual Devices (AVDs). The framework automatically installs the application, grants necessary permissions, and executes simulated user actions to trigger app functionalities. During this process, AnForA continuously monitors the file system using FSWatcher, which is based on the Linux inotify mechanism. This allows the system to detect any file creation, deletion, or modification events in real-time, ensuring that no critical forensic artifact is missed during analysis. A major learning from this paper was the emphasis on key forensic principles embedded within AnForA's design: fidelity, ensuring that interactions closely mimic genuine user behavior; artifact coverage, guaranteeing complete retrieval of forensic artifacts; artifact precision, ensuring that extracted data is relevant and accurate; repeatability, allowing experiments to be consistently reproduced; and generality, enabling the system to work across diverse types of applications without customization. These qualities ensure that evidence collected through AnForA remains legally admissible and scientifically verifiable.

A key part of the analysis involved in understanding AnForA's architecture as shown in figure 2.7.1, which is composed of several coordinated modules:

- The App Installer handles the APK installation and permission management.
- The User Emulator uses UI automation to mimic real user interactions with the app interface.
- The Analysis Paths Detector identifies application directories that may store relevant forensic artifacts.
- FSWatcher tracks real-time changes in the filesystem using Linux's inotify mechanism.
- The Retriever pulls filesystem data after each interaction step.
- The Difference Extractor compares sequential filesystem snapshots to isolate files created, modified, or deleted during app execution.
- Finally, a GUI Interface facilitates streamlined interaction for forensic analysts.
- Captures and logs all network activity during app execution, enabling the detection.

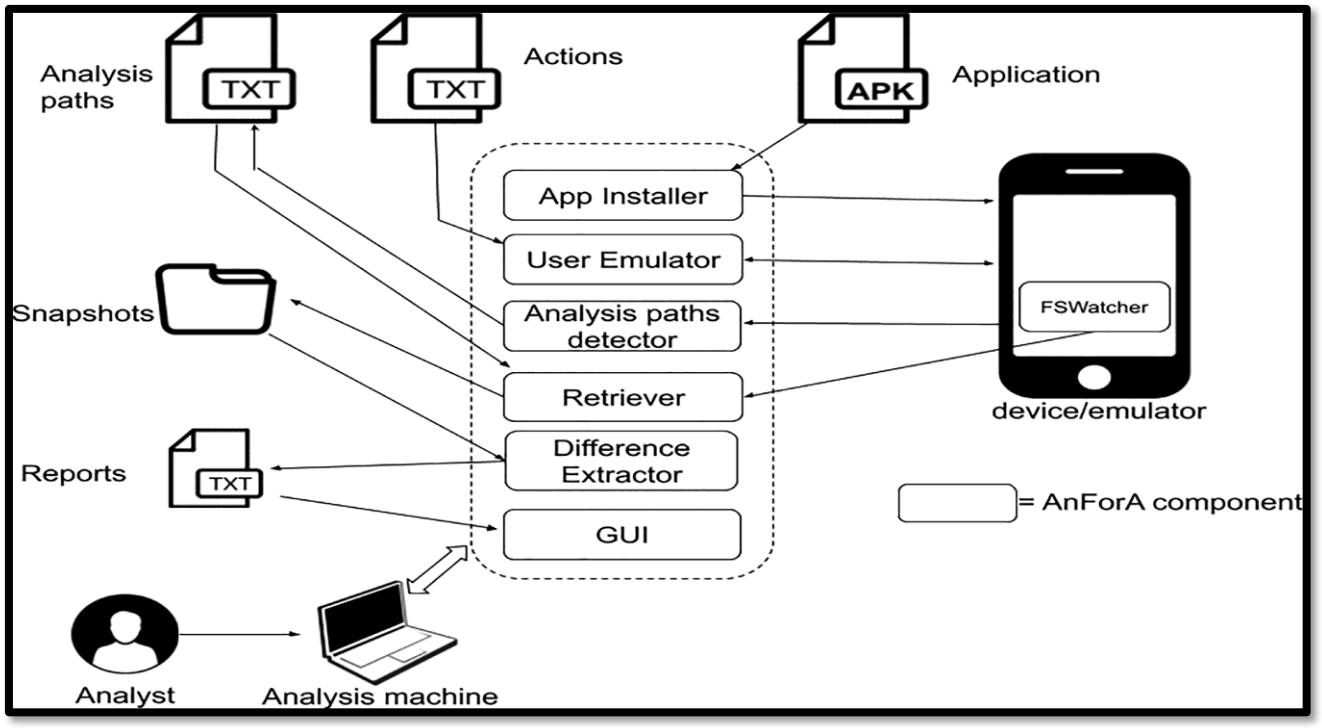

Figure 2.7.1 Anfora Architecture

2.8 Automated application and permission management (week 8)

The App Installer is the first and foundational component of the AnForA framework. Its primary role is to automate the process of deploying Android application packages (APKs) onto an emulated Android device, known as an Android Virtual Device (AVD). This component ensures that the target application is installed consistently, reliably, and with all necessary permissions granted, setting the stage for an effective forensic analysis.

In traditional forensic workflows, the installation of applications is often a manual task involving multiple steps: transferring the APK to the device, triggering installation, and manually accepting permission requests during runtime. Such manual intervention introduces the risk of inconsistencies, incomplete setups, and human errors that can severely affect the reliability of the forensic findings. To eliminate these risks, AnForA's App Installer automates the entire process using a systematic, script-driven approach built on top of the ADB (Android Debug Bridge) communication protocol.

The automated installation process begins as shown in the figure 2.8.1 with the system identifying the APK to be analyzed. The App Installer uses ADB commands to push the APK file onto the emulator, ensuring the application is placed in the correct file path. Subsequently, the adb install command is executed, which installs the APK silently without requiring manual confirmation from a graphical interface. This ensures that the application is installed in a controlled and repeatable manner every time the forensic experiment is executed.

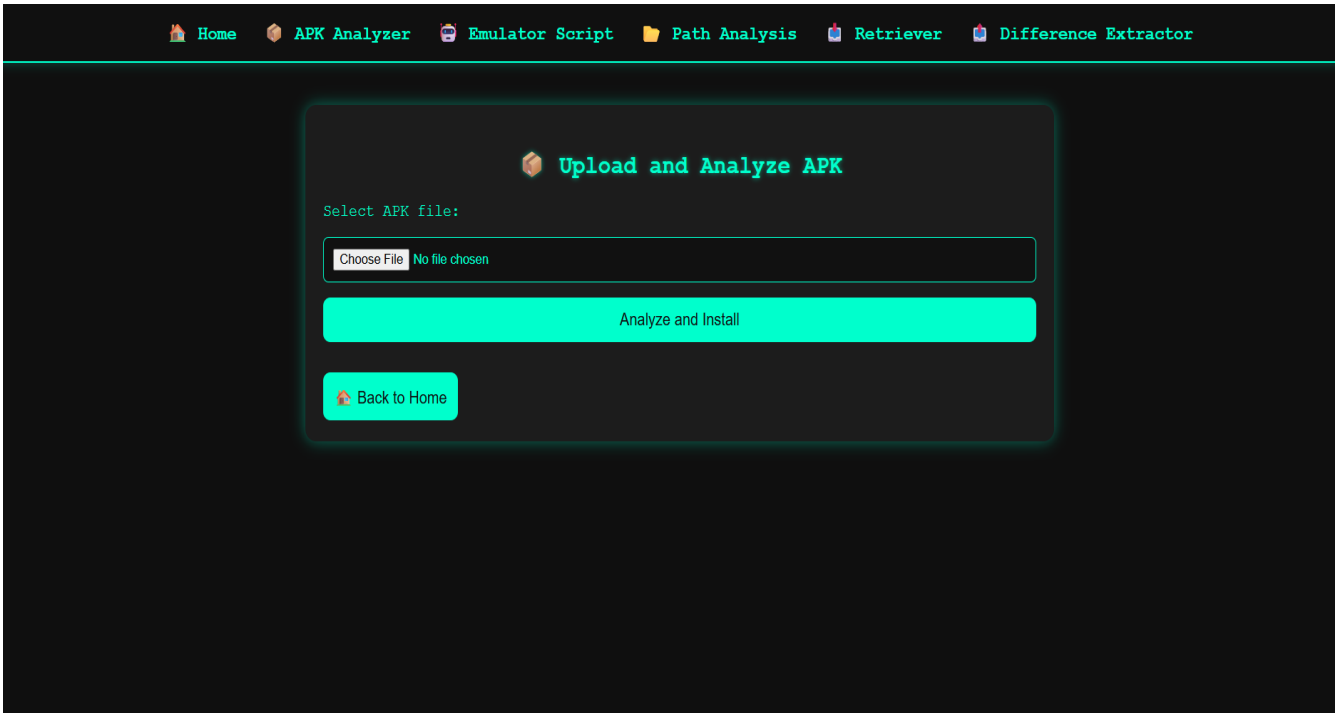

Figure 2.8.1 Analysing and Installation Process

A notable feature of the App Installer is automated permission granting as shown in the figure 2.8.1, Many modern Android applications request access to sensitive device components such as the camera, contacts, SMS, call logs, location, and storage. Without granting these permissions, several app functionalities might not work, potentially hiding critical evidence. To address this, the App Installer intelligently parses the application's declared permissions (using Android's package manager tools or predefined permission lists) and automatically grants them through shell commands. After the installation and permission setup, the App Installer performs a validation check to ensure that the application has been correctly installed. It verifies the presence of the application package by querying the installed apps list. If the installation fails or if necessary permissions are missing, the installer flags the error immediately, allowing the forensic analyst to correct the issue before proceeding.

Through this carefully orchestrated process, the App Installer upholds essential forensic qualities:

- Repeatability: Every installation is performed in a uniform and verifiable way across different runs and app versions.
- Fidelity: Application behavior remains authentic, closely reflecting a real user environment without manual distortion.
- Integrity: Evidence is preserved by ensuring that the app operates with all necessary permissions, preventing evidence loss due to blocked functionalities.
- Efficiency: Saves substantial setup time, particularly in large-scale investigations involving multiple apps or different device configurations.

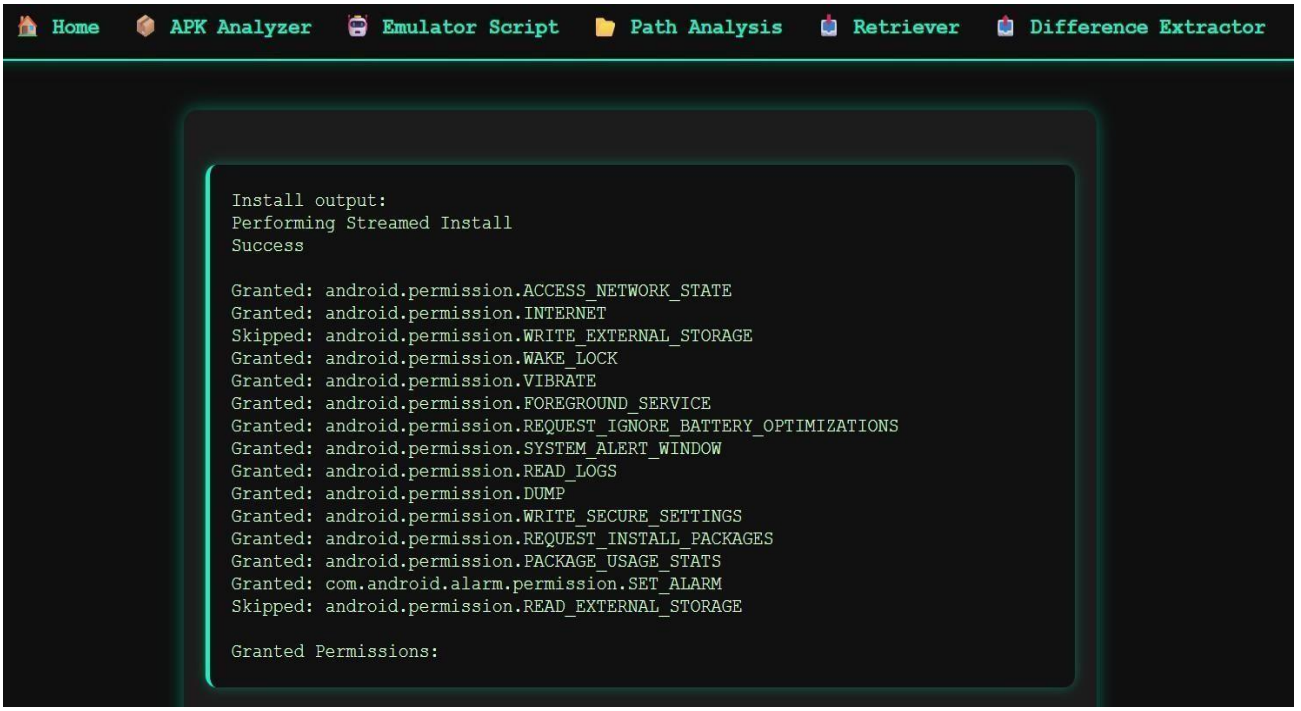

Figure 2.8.2 Granted Permission for APK File

2.9 Automated Simulation of User Interactions (week 9)

The second major component of the AnForA framework is the User Emulator as shown in figure 2.9.1, which focuses on simulating realistic user interactions within Android Virtual Devices (AVDs). Its primary function is to automatically interact with the installed application by mimicking common behaviors such as opening the app, navigating menus, entering text, and triggering various in-app actions. This is essential because applications typically generate forensic artifacts—such as databases, cached files, and logs—only after being actively used. Without automation, manually simulating such usage would be time-consuming, inconsistent, and prone to human error.

To overcome this, the User Emulator utilizes a combination of basic ADB input commands and more sophisticated UI automation techniques. At a basic level, it sends ADB shell commands to simulate taps, swipes, typing, and button presses, enabling scripted, repeatable interaction sequences as shown in figure 2.9.2, For more dynamic apps, the emulator employs UI-based automation frameworks like UIAutomator, allowing it to detect and interact with UI elements intelligently based on their properties, such as resource IDs or class names. The App Installer uses ADB commands to push the APK file onto the emulator, ensuring the application is placed in the correct file path. Subsequently, the adb install command is executed, which installs the APK silently without requiring manual confirmation from a graphical interface.

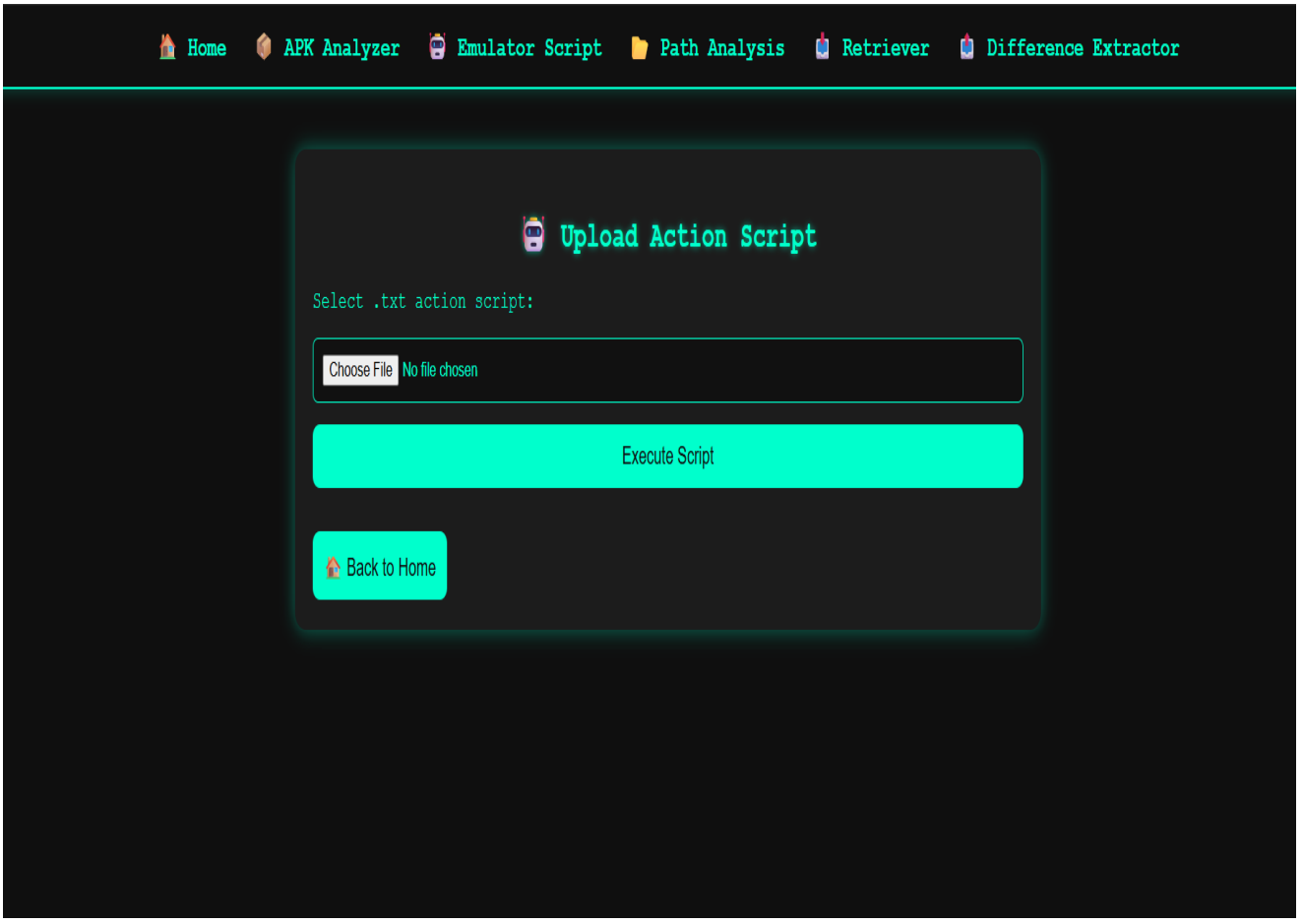

Figure 2.9.1 Uploading of Action Scrip

The User Emulator begins with basic input simulation by sending ADB commands such as adb shell input tap, swipe, text entry, and key events to mimic touch actions, typing, and navigation. These actions can be precisely scripted to match expected app behavior, ensuring predictable and repeatable interactions. For more complex applications, the emulator leverages UI-based interaction automation tools like UI Automator or Monkey Runner, which allow it to detect and interact with UI elements dynamically based on their properties (e.g., resource ID, class name). This enables intelligent interaction with buttons, text fields, checkboxes, and other components, closely replicating real user behavior.

An important aspect of the User Emulator is its ability to manage timing and control flow, incorporating intentional pauses and checks to ensure that UI elements are properly loaded before actions are taken. This prevents premature actions that might cause app crashes or inconsistent behavior. Additionally, it includes error recovery mechanisms, such as retries and fallback actions, to enhance the robustness of the interaction process. Every action performed by the emulator is carefully logged with timestamps, creating a verifiable activity trail that supports forensic validation. Through automated, realistic interaction.

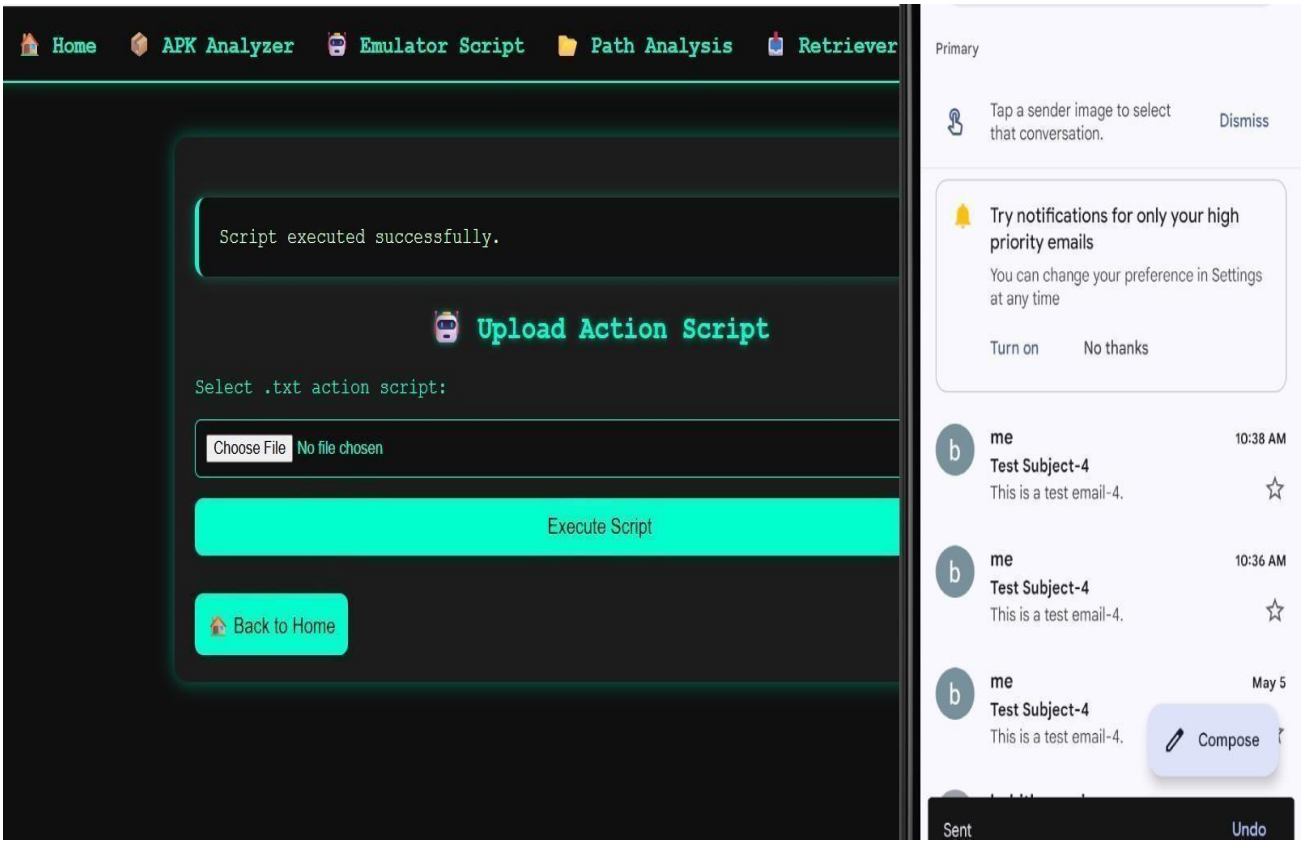

Figure 2.9.1 Uploading of Action Script

2.10 Understanding the Path Detector in Android App Analysis (week 10)

Modern Android applications, storage operations have become more complex due to changes in the Android permission model (like Scoped Storage introduced in Android 10). The Path Detector must adapt to these changes by not only recognizing classical storage paths but also monitoring app-specific directories and temporary storage areas that are dynamically allocated. This is crucial because many apps now create and delete data on the fly, based on user activity, cloud synchronization, or service interactions. Another important aspect of path analysis is detection of sensitive data leakage. Applications may inadvertently or intentionally store sensitive information like access tokens, passwords, personal identifiers, or cached user content in insecure locations such as public folders on external storage. A Path Detector flags such risky behavior by comparing detected paths against security policies. This proactive identification helps developers and auditors fix vulnerabilities before an application is deployed publicly. A sophisticated Path Analysis system also considers file lifetime and deletion tracking. Sometimes, apps create files that are meant to be temporary but leave them behind unintentionally (a phenomenon known as "data residue"). In forensic investigations, even deleted or residual files can be recovered to extract meaningful evidence. By maintaining a timeline of file creation and deletion activities as shown in figure 2.10.1.

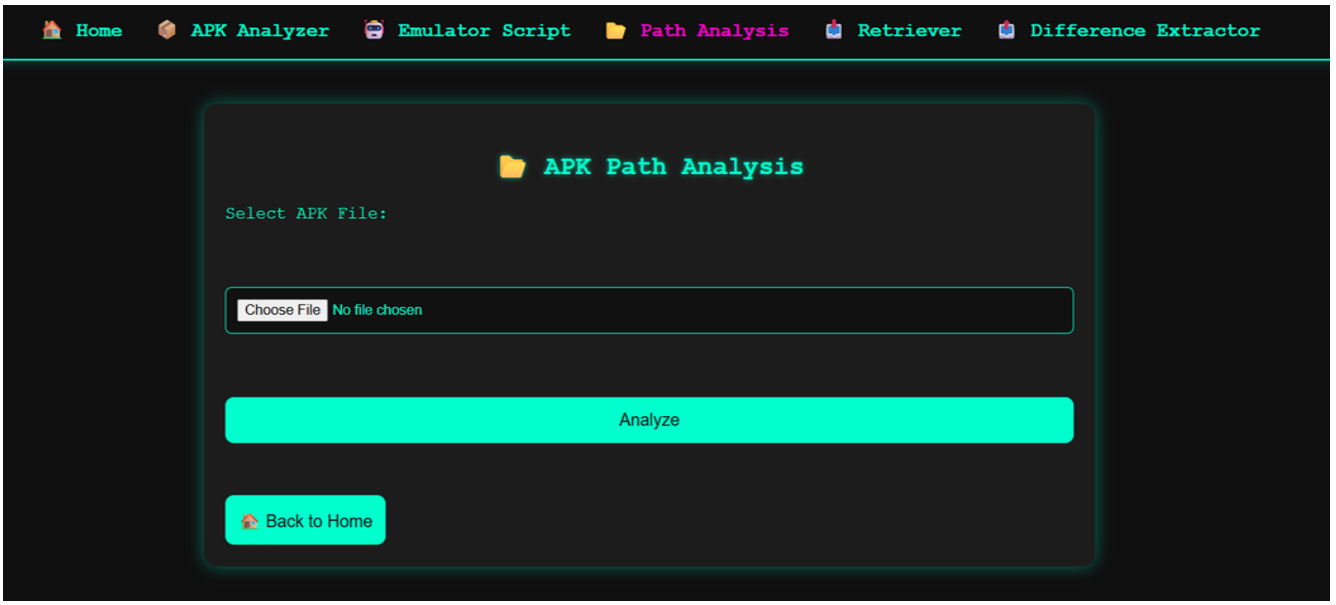

Figure 2.10.1 Analyzation of APK File

Path Detection is not limited to standalone app analysis. In enterprise security, multiple apps may interact with shared storage or external devices. In such cases, the Path Detector can be extended to monitor inter-app storage interactions as shown in figure 2.10.2, identifying unauthorized access or potential data leaks between different apps, a scenario often exploited by malicious software. Moreover, a robust Path Detector often includes metadata collection for each path it finds. It may capture details such as file permissions, last modification times, file sizes, and access ownership. This metadata is critical in assessing security posture — for instance, identifying if a sensitive file is world-readable or if user data is being written to publicly accessible storage.

In the context of cybersecurity and digital forensics, the Path Detector becomes an essential tool. Malware analysts use it to identify hidden files or rogue operations were malicious apps store payloads, logs, or stolen data. Forensic investigators rely on it to map an app's footprint on a seized device, helping reconstruct user activity or recover vital evidence from app-specific folders even after uninstallation. Ultimately, the Path Detector bridges the gap between app behaviour and file system impact, enabling security professionals to verify whether applications comply with best practices for data storage and privacy. Its ability to surface invisible activities and risky file operations makes it invaluable in application testing, malware detection, forensic analysis, and data leak prevention. An important aspect of the User Emulator is its ability to manage timing and control flow, incorporating intentional pauses and checks to ensure that UI elements are properly loaded before actions are taken. This prevents premature actions that might cause app crashes or inconsistent behavior. Additionally, it includes error recovery mechanisms, such as retries and fallback actions.

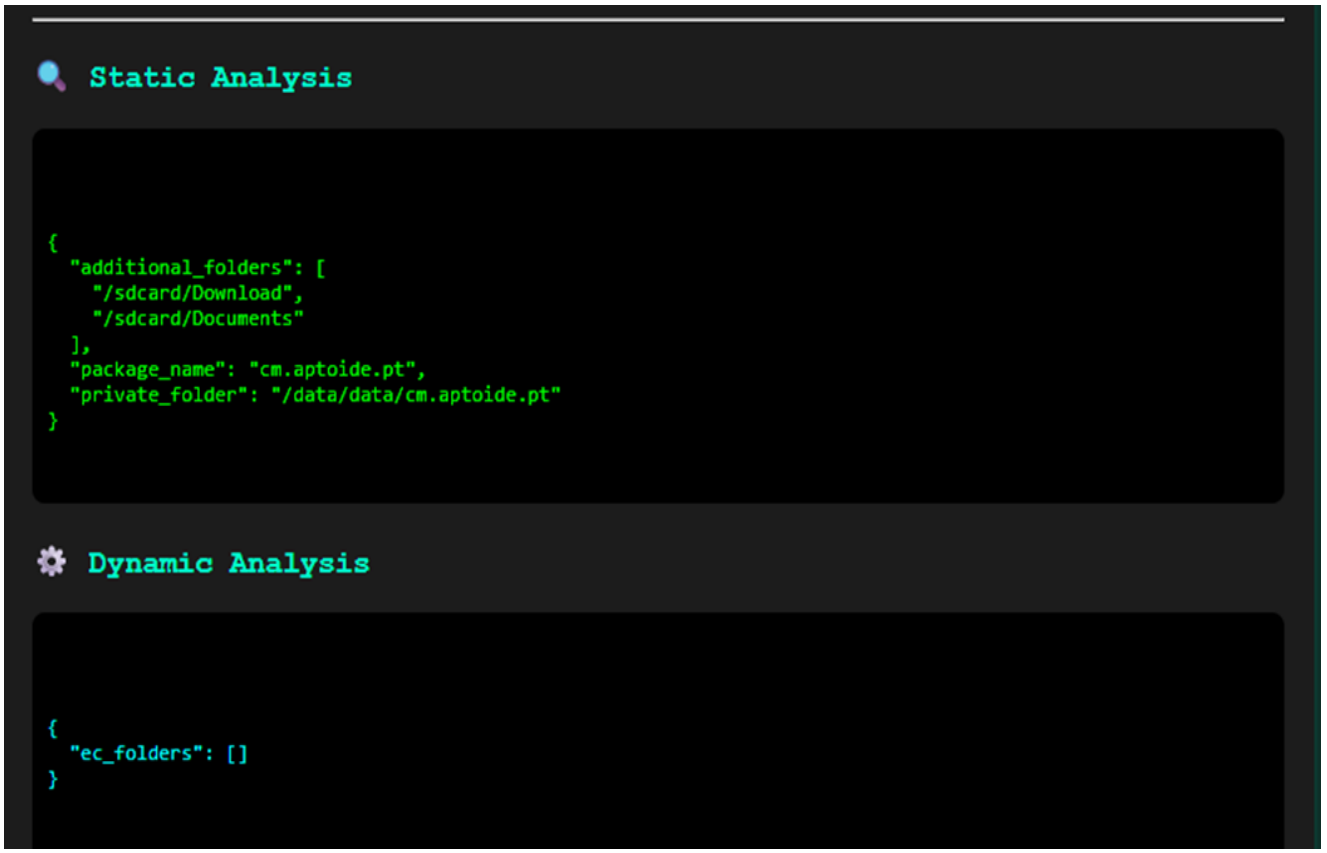

Figure 2.10.2 Analyzed Paths

2.11 Challenges Faced by the Retriever During Data Recovery (week 11)

The Retriever is a critical module used in Android app analysis, digital forensics, and security auditing to extract or recover data from a device's storage. Its main function is to retrieve files, databases, logs, and artifacts either from the live file system or from areas where files have been deleted but not yet overwritten. By accessing internal storage, external SD cards, or app-private directories, the Retriever enables analysts to recover both visible files and hidden or deleted ones that are crucial for investigation or app testing as shown in figure 2.11.1. at its core, the Retriever scans through accessible storage areas such as internal storage, external storage (SD card), app-private directories, and sometimes even low-level block storage to find files related to an app's operation. It can identify active files (still present and visible) as well as deleted files that still exist in unallocated space until they are overwritten.

Retrieval operations are often performed using a variety of techniques. For live files, simple adb pull commands or app backups are used to extract data directly from accessible paths. In cases where files have been deleted, more advanced methods like file carving are applied, where the Retriever scans raw storage blocks looking for recognizable file patterns such as headers and footers of common formats like JPEG, SQLite, or PDF. This allows partial or full recovery of data that standard file

listings would no longer show in a dynamic environment like an emulator; the Retriever can also capture snapshots of the file system at different stages of app usage. By comparing snapshots before and after user activities, it can detect newly created files, modified content, or deleted artifacts that indicate important operations performed by the app.

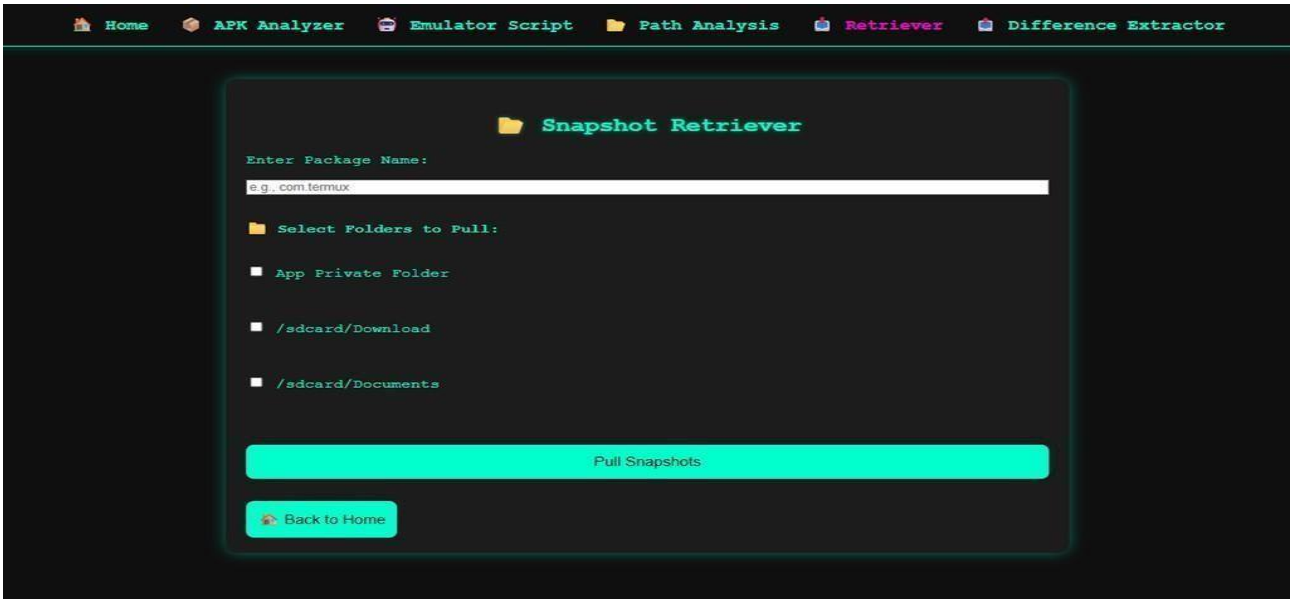

Figure 2.11.1 Pulling Snapshot

The Retriever plays an important role in exposing security flaws. For example, if an app claims to delete user data but the Retriever can still recover old files from the device's internal storage, it highlights a privacy vulnerability. Similarly, forensic investigators rely heavily on Retriever modules to gather evidence in criminal investigations, piecing together user activities even after app uninstallation or data deletion attempts.overall, the Retriever acts as a powerful tool that uncovers the hidden traces left behind by applications. Whether it's used for recovering lost user data, identifying forensic evidence, auditing app behaviors, or assessing data leakage risks, the Retriever ensures that no critical information is missed during an Android storage analysis.

Despite its powerful capabilities, the Retriever faces several challenges during the recovery process as shown in figure 2.11.2. One major obstacle is secure deletion techniques, where apps overwrite deleted files with random data, making recovery practically impossible. Similarly, file encryption adds another layer of difficulty — many apps encrypt their sensitive files, and without the decryption keys, the Retriever can only recover unreadable data blobs. The introduction of Scoped Storage in newer Android versions also limits direct access to certain directories, especially app-private folders, unless the device is rooted or special permissions are granted. In some cases, quick file overwriting by the operating system further reduces the chances of recovering deleted files.

Additionally, anti-forensics techniques, where apps intentionally hide or obfuscate their file paths, present another challenge that requires more sophisticated scanning and analysis methods. These hurdles mean that while the Retriever is extremely useful, its effectiveness often depends on the device's state, the app's behavior, and the overall security architecture of the Android system.

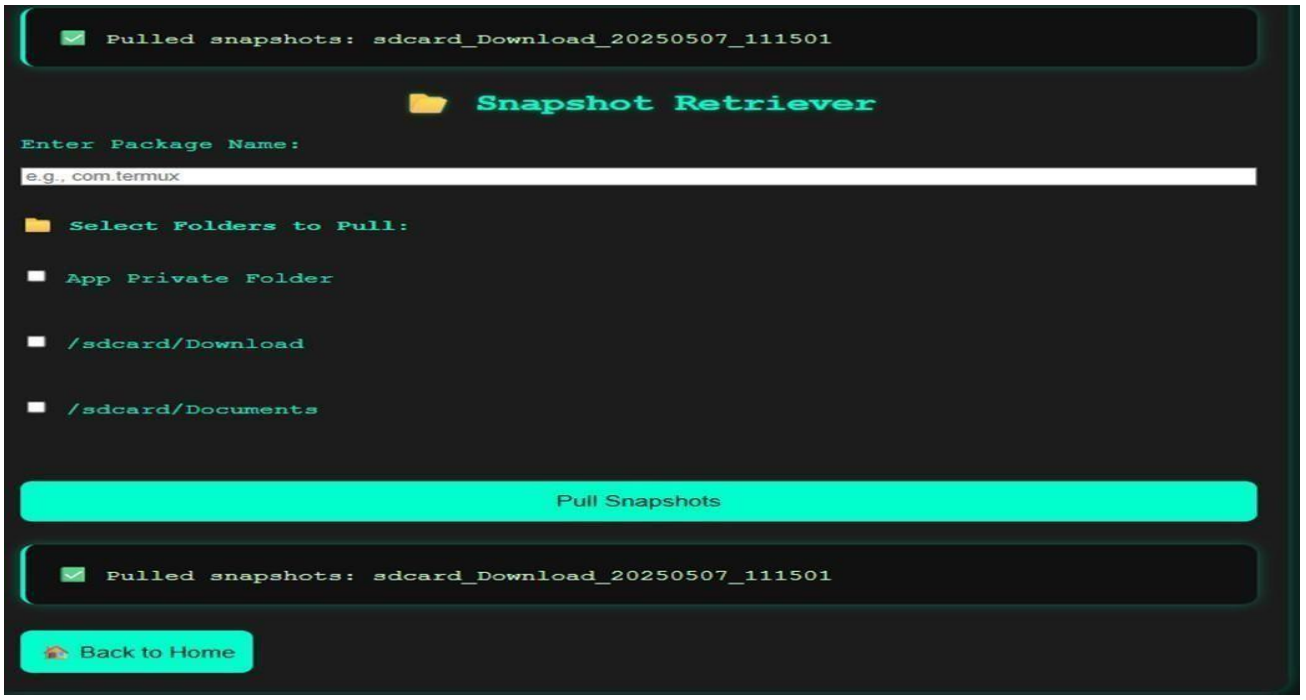

Figure 2.11.2 Retrieved Snapshot

2.12 Overview of the Difference Extractor for APK Analysis (week 12)

The difference extractor component is designed to provide a deep dive into the changes that occur between different states of a system or file after APK installation, modification, or interaction with an Android emulator as shown in figure 2.12.1. In a forensic or security context, this tool becomes invaluable by enabling precise tracking of all system and file-level modifications made by an APK during both static and dynamic analysis phases. Whether it's for examining the aftermath of an APK installation or the result of running an app, the extractor can compare file structures, permissions, and behaviors before and after an event, ensuring no changes go unnoticed.

One of the key features of the difference extractor is its ability to perform file-level comparisons. Using file hashing, it can determine whether any files in the system have been modified or added. For example, it might compare the state of system files in the Android emulator before and after the APK is installed. If any new files are added or if existing files are altered (either by size, content, or metadata), the extractor will flag these discrepancies. This ensures that any malicious or unexpected changes—such as the modification of sensitive system files—are easily detected.

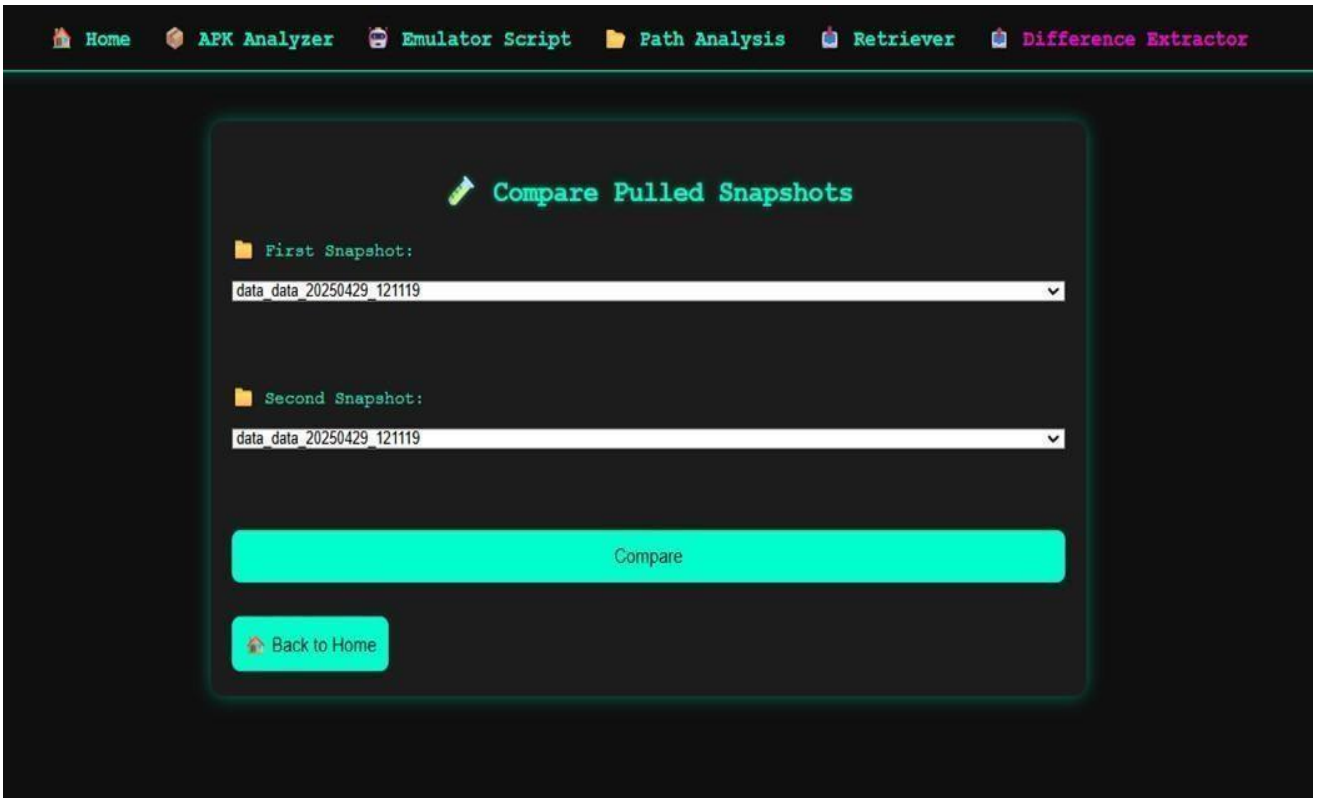

Figure 2.12.1 Comparing of Pulled Snapshots

The permissions comparison feature adds another layer of scrutiny. It's not uncommon for an APK to request additional permissions after installation, or during runtime, especially if the app needs to access sensitive data or system resources. The difference extractor will compare the permissions granted to the APK at different points, such as before installation and after it's running in the emulator. By doing this, it can detect any new permissions requested, and even track the modification of existing permissions, making it easier to identify potential privacy or security risks as shown in figure 2.12.2,For example, if an app initially asks for basic permissions but later seeks access to the camera or microphone during execution, the tool will flag these changes.This makes the extractor a powerful tool for security analysts, forensics experts, and app developers who need to understand the full scope of an APK's impact.

Moreover, the extractor can be used in continuous monitoring or automation scenarios. For example, if the tool is run repeatedly on different versions of an APK or as part of an ongoing investigation, it can highlight incremental changes or detect long-term trends. This can be valuable in cases where an APK evolves over time or when analysing large volumes of apps to spot patterns or anomalous behaviour. Overall, the difference extractor not only provides a detailed view of the APK's impact on the system, but it also helps streamline forensic investigations and improve security analysis. This allows partial or full recovery of data that standard file listings would no longer show in a dynamic environment like an emulator.

Figure 2.12.2 Differences Found

2.13 Structured Presentation of Analysis Results (week 13)

The Report Generation Component is a fundamental part of the AnForA (Android Forensic Automation) framework, designed to systematically compile and present the results of forensic analysis in a structured and professional format. Once the data collection, automated interactions, and difference extraction processes are completed, the component takes over to generate a detailed report that consolidates all findings. The primary purpose of this component is to transform complex and technical data into a clear, readable, and well-organized document that can be used for analysis, review, or as part of a legal evidence presentation.

This component plays a critical role in bridging the gap between raw data extraction and meaningful interpretation. During the forensic investigation, the AnForA framework gathers a vast amount of data, including filesystem changes, logs, and screenshots, resulting from user interactions and application activities. The Report Generation Component meticulously organizes this data into a coherent format. Typically, the report includes an introduction that outlines the scope and objectives of the analysis, followed by a summary of the methods employed and the applications tested. The core sections of the report detail the observed changes, categorized by file modifications, creations, or deletions, and are presented alongside timestamps and file paths for accurate tracking.

2.14 Frontend Styling and User Interface Integration (week 14)

The frontend design of the web application has been crafted with a focus on delivering a professional, cybersecurity-themed user experience as shown in figure 2.14.1, Leveraging a dark-themed background with neon-accented highlights, the interface successfully creates a sleek and visually striking aesthetic. Core interactive elements, such as the navigation bar, input fields, and action buttons, are styled with vibrant turquoise tones that ensure high contrast and excellent visibility against the dark backdrop. This color scheme not only enhances user engagement but also aligns with modern design standards typically adopted in cybersecurity tools.

A modular approach to frontend development has been followed, wherein CSS classes are centrally organized to maintain consistency and scalability across the application. Rounded corners (border-radius) and dynamic glow effects (box-shadow) are strategically applied to buttons, containers, and interactive panels, creating a polished, cohesive appearance. Hover effects and active states provide intuitive feedback during user interactions, improving the responsiveness and usability of the interface. The application of smooth animations and transitions further enhances the overall experience.

The layout architecture extensively utilizes CSS Flexbox and Grid systems to achieve a responsive design framework. This ensures that the application maintains its structural integrity and visual alignment across various devices and screen sizes. Special attention has been given to the navigation bar, which is positioned at the top with symbolic icons accompanying each menu item. The icons not only serve an aesthetic purpose but also facilitate quick module identification, enhancing user navigation without relying solely on textual labels.

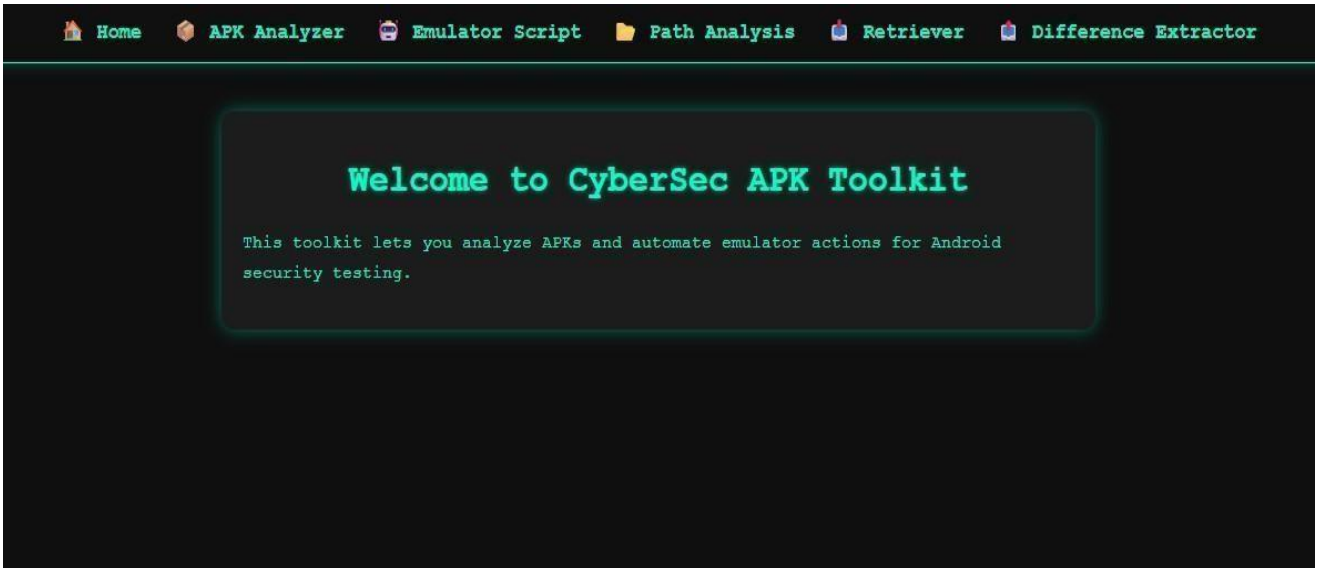

Figure 2.14.1 Home Page

2.15 Overall Presentation and Development of Project (week 15)

During Week 15, n Week 15, the project entered a crucial phase focused on polishing, integrating, and professionally presenting all the work completed throughout earlier weeks. A strong emphasis was placed on delivering a fully functional and visually refined APK Analysis Web Application. The frontend interface was enhanced significantly, adopting a dark-themed cybersecurity aesthetic, with neon highlights, clear typography, rounded card designs, and soft glowing effects. Navigation between different modules such as APK Analyzer, Emulator Script Execution, Path Analysis, Snapshot Retrieval, and Difference Extraction was made seamless and intuitive. Special care was taken to ensure that the user experience was both efficient and visually engaging, employing consistent color schemes, hover animations, and responsive layouts using Flexbox and Grid.

Parallelly, backend modules were integrated seamlessly to support critical functionalities like APK uploading, static analysis, dynamic analysis through emulator scripting, and path analysis retrieval. Special attention was given to ensuring that frontend actions such as "Upload and Analyze APK" were dynamically connected to backend services, with error handling and feedback mechanisms in place. This integration allowed for a smooth and real-time user experience, which is essential for forensic and cybersecurity applications. Concepts such as Application Scanners, Evidence Collection Modules, and Automated Reporting Systems were critically examined. These learnings were incorporated into the project’s architecture to align the tool closer to real-world forensic analysis frameworks.

Additionally, a thorough study of the research paper "AFORAN: An Automated Approach for Android Forensic Analysis" was undertaken. Key components such as the Application Scanner were critically analyzed and documented to understand how automated detection, classification, and analysis methodologies can be applied in a real-world forensic framework. Insights from the research paper directly influenced the module design and functionality prioritization within the developed application.

In preparation for the final presentation, a structured and comprehensive documentation strategy was adopted. Emphasis was placed on conveying both the technical depth and practical relevance of the work. Special highlights included showcasing the integration between frontend-user actions and backend forensic automation, as well as drawing connections between the project design and the methodologies proposed in leading academic research in summary, Week 15 focused on unifying frontend, backend, research learning, and presentation design.

CHAPTER 3

REFLECTION

3.1 Internship Reflection

The internship at NITK provided a valuable opportunity to apply theoretical knowledge in a practical context, specifically focusing on Android forensics and automation. The primary objective was to develop and implement the Anfora (Android Forensic Automation) tool, designed to streamline data extraction from Android emulators. This goal was pursued through a structured approach involving research, development, testing, and continuous refinement. Working within a collaborative environment facilitated the exchange of ideas, fostering innovative solutions to technical challenges encountered throughout the project.

3.2 Achievement of Objectives

The objectives of the internship were progressively achieved by adopting a modular and iterative development approach. Initially, the focus was on understanding Android forensic concepts, particularly how to automate the collection of digital artifacts from Android environments. After establishing a theoretical foundation, we began the practical implementation of AnForA, which integrates several key components: APK installation, user interaction simulation, real-time filesystem monitoring, and automated report generation. Each component was meticulously developed and tested within various Android emulator environments to ensure robustness and accuracy. Frequent testing sessions, feedback loops, and peer reviews contributed to refining the tool's functionalities, making it more efficient and reliable. Collaborative discussions and presentations helped in addressing technical issues and improving the overall design of the tool.

3.3 Skills Acquired

This internship significantly enhanced my technical skills in Android forensics and automation. I gained hands-on experience with tools like ADB (Android Debug Bridge) and UI Automator for automating user interactions, as well as utilizing inotify for real-time filesystem monitoring. Additionally, I developed proficiency in Python scripting to automate data extraction and report generation, integrating these components to work cohesively within the AnForA framework. The project also enhanced my ability to manage complex workflows and troubleshoot issues that arise from diverse Android environments. Besides technical competencies, I improved my soft skills, particularly in documentation, technical writing, and presenting the tool's capabilities to peers.

3.4 Results/Observations/Work Experiences

The internship experience provided practical insights into the complexities of automated forensic data collection from Android devices. One of the most significant observations was that automated user interactions greatly influence the volume and type of forensic artifacts generated. This emphasized the importance of realistic and consistent user emulation in forensic automation. Another key insight was the variability in data output depending on the Android version and device configuration, which required adaptive programming techniques to ensure compatibility. Hands-on tasks included scripting automated APK installations, simulating user actions like clicking buttons or filling forms, and capturing file changes during app execution. The integration of real-time monitoring allowed for efficient tracking of file modifications, ensuring that all relevant data changes were documented. Presenting the tool's functionalities and demonstrating its practical applications not only validated the tool's effectiveness but also showcased the potential of automation in forensic analysis.

3.5 Challenges Faced

One of the primary challenges during the internship was ensuring compatibility between the AnForA tool and various Android emulator versions. Differences in UI layouts and permissions management across Android versions required creating adaptable automation scripts. Another challenge involved detecting subtle filesystem changes that occurred during rapid app interactions. Efficiently capturing and categorizing these changes without overwhelming the monitoring system required careful optimization. Additionally, generating accurate reports from dynamically changing data sets posed a challenge, necessitating the use of robust data parsing and formatting techniques. Managing concurrent tasks, such as user emulation and file monitoring, also required synchronization to maintain data integrity. Overcoming these challenges involved a combination of rigorous testing, iterative debugging, and incorporating feedback from mentors and colleagues.

CHAPTER 4

CONCLUSION

The internship at NITK provided an invaluable opportunity to bridge theoretical knowledge with practical implementation in the field of Android forensics. Developing the AnForA (Android Forensic Automation) tool was a challenging yet rewarding experience that significantly enhanced both technical and analytical skills. The project involved integrating various components such as APK installation, user emulation, filesystem monitoring, and automated report generation, all of which contributed to creating a robust and efficient forensic automation tool.

Throughout the internship, I faced several challenges, including handling diverse Android environments and ensuring the accuracy of data extraction. However, by leveraging problem-solving techniques and actively collaborating with mentors and peers, I was able to overcome these obstacles and successfully achieve the project objectives. The hands-on experience not only deepened my understanding of Android forensic practices but also reinforced the importance of automation and adaptability in forensic investigations.

In conclusion, the internship experience significantly contributed to my professional growth, equipping me with practical skills in digital forensics, automation, and technical problem-solving. Working on AnForA has instilled a strong foundation for future projects in the field and has prepared me to address real-world challenges in digital investigation with confidence and technical competence.

REFERENCES

[1] Dey, A., Chakraborty, S., & Sen, S. (2023). AFORAN: *An Automated Framework for Android File System Monitoring and Forensic Analysis*. In Proceedings of the 2023 IEEE 13th International Conference on Communication Systems and Network Technologies (CSNT) (pp. 190–195). IEEE. https://doi.org/10.1109/CSNT57996.2023.10157632

[2] Bhutani, A., Ranjan, A., & Girdhar, A. (2020). *Automated Tool for Extracting and Analysing Deleted WhatsApp Artifacts from Android Phones*. Procedia Computer Science, 171, 1467–1476. https://doi.org/10.1016/j.procs.2020.04.157

[3] Li, C., & Li, D. (2011). *Mobile Phone Forensics Analysis System Based on Android Platform*. Procedia Engineering, 15, 3027–3031. https://doi.org/10.1016/j.proeng.2011.08.566

[4] Grover, J., & Sharma, M. (2015). *A Comparative Study of Mobile Device Forensics Tools*. International Journal of Computer Applications, 109(2), 7–10. https://doi.org/10.5120/19124-0792

[5] Lin, X., Chen, T., Zhu, T., Yang, K., & Wei, F. (2018). Automated forensic analysis of mobile applications on Andriod devices.*26*, S59–S66. https://doi.org/10.1016/j.diin.2018.04.012

[6] Umar, R., Riadi, I., & Zamroni, G. M. (2017). *A Comparative Study of Forensic Tools for WhatsApp Analysis using NIST Measurements*. *International Journal of Advanced Computer Science and Applications*, 8(12). https:// doi: 10.1016/j.diin.2016.04.001.

[7] Roy, N. R., Khanna, A. K., & Aneja, L. (2016). *Android Phone Forensic: Tools and Techniques*. International Conference on Computing, Communication and Automation (ICCCA). https://doi: 10.1145/1052883.1052895

[8] Alldredge, J. (2015). *The 'CSI Effect' and Its Potential Impact on Juror Decisions*. *Themis: Research Journal of Justice Studies and Forensic Science*, https://doi: 10.1145/2184751.2184827

[9] Burkhard, M. (2014). *Handbook of Forensic Medicine*. Sussex: Wiley Blackwell, p. 10. ISBN: 9780470979990. https://doi: 10.1007/978- 81- 322- 2268- 2

[10] Braithwaite, J. (2000). *The New Regulatory State and the Transformation of Criminology*. *British Journal of Criminology*, 40(2), pp. 222–238. https:// doi: 10.1145/3213846.32