F25_4440_S003_G12 –

Forensic Analysis of Mobile Applications

Final Report

Team Members:

Akinro Akintunde (Team Lead) - 300389708

Ogunseye Israel – 300368938

Archival Video

https://www.youtube.com/watch?v=r2oKv6DNoZA

Abstract

This project presents a comprehensive forensic analysis of mobile applications, specifically Telegram and TikTok, using multiple forensic tools: Andriller for data extraction, MVT (Mobile Verification Toolkit) for spyware detection, MobSF for static security analysis, and a novel custom-built AKT Forensic Tool combining features from all three.

Akinro Akintunde utilized Andriller to extract messages, call logs, and cached media from Telegram, demonstrating data recoverability.

Ogunseye Israel conducted an in-depth analysis of the Telegram APK using MobSF, identifying dangerous permissions, exported components, and potential security vulnerabilities. In addition, Israel employed **mitmproxy** to capture and analyze **live network traffic**, providing insight into real-time communication patterns, API calls, encryption behaviors, and potential data leakage during app usage.

The novel AKTIS tool integrates real-time monitoring, data extraction, and security analysis capabilities, providing a unified forensic examination platform. Results reveal significant privacy concerns, with extensive user data recoverable even from encrypted applications, highlighting the importance of mobile device forensics in digital investigations.

1. Background Research

1.1 Related Tools and Research

Mobile device forensics has become increasingly important as smartphones store vast amounts of personal and sensitive data. The field of mobile forensics has evolved significantly, with several established tools emerging to address the growing need for comprehensive device analysis. Andriller represents a commercial forensic solution specifically designed for Android devices, capable of extracting data from both physical devices and backups. This tool excels at recovering critical user information including messages, call logs, contacts, and media files from various Android applications, making it invaluable for law enforcement and security professionals conducting digital investigations.

The Mobile Verification Toolkit (MVT), developed by Amnesty International as an open-source solution, addresses a different but equally critical aspect of mobile forensics: spyware detection. MVT analyzes iOS and Android device backups to identify indicators of compromise, detecting suspicious network connections and data exfiltration patterns that may indicate malicious activity. This tool has gained recognition for its effectiveness in identifying sophisticated spyware attacks, particularly those targeting journalists, activists, and government officials.

MobSF (Mobile Security Framework) provides a comprehensive automated security testing platform for mobile applications, performing both static and dynamic analysis of Android and iOS apps. This framework systematically identifies security vulnerabilities, dangerous permissions, and exposed components that could be exploited by malicious actors. MobSF's ability to generate detailed security scorecards and comprehensive reports makes it an essential tool for security researchers and developers seeking to understand the security posture of mobile applications.

Additionally, ALEAPP (Android Logs, Events, and Protobuf Parser) serves a specialized role in parsing Android system logs and artifacts to extract user activities and timestamps. This tool is particularly useful for reconstructing user behavior and understanding application interactions with the Android system.

To support this research, several academic publications were reviewed, including the International Advanced Research Journal in Science Engineering and Technology (IARJSET.2022.91102) article on Mobile Forensics Analysis, and the Malaysian Journal of Computer Information Systems (MJCIS) Volume 16 Issue 1 publication on Mobile Device Forensics and Privacy Assessment. These research articles provided valuable insights into current methodologies, challenges, and best practices in the field of mobile device forensics.

1.2 Purpose of the Tool/Forensic Examination

The primary purpose of this forensic examination extends beyond simple data extraction to encompass a comprehensive assessment of mobile application security and privacy implications. This investigation seeks to assess the extent of recoverable user data from popular mobile applications, specifically focusing on Telegram and TikTok, to understand what information remains accessible even after users believe they have deleted or secured their data. By systematically extracting and analyzing data from these applications, we aim to demonstrate the reality of data persistence in mobile devices and the challenges this presents for both privacy advocates and forensic investigators.

Furthermore, this examination aims to identify security vulnerabilities and privacy concerns inherent in mobile applications. Through static and dynamic analysis, we seek to uncover dangerous permissions, exposed components, and insecure data storage practices that could potentially be exploited by malicious actors. This aspect of the research is particularly important as it highlights the need for improved security practices in mobile app development and helps users understand the risks associated with their mobile application usage.

An additional significant objective of this project is the development of a comprehensive forensic tool that combines the capabilities of existing tools into a unified platform. By integrating features from Andriller, MVT, and MobSF, we aim to create a more efficient and comprehensive forensic analysis solution that reduces the need for investigators to switch between multiple tools and interfaces. This integrated approach not only improves workflow efficiency but also provides a more holistic view of device security and data recoverability.

This project also serves to demonstrate real-world forensic investigation techniques applicable to digital crime investigations. By documenting the complete process from device connection through data extraction and analysis, we provide a practical guide for forensic investigators and security professionals. The techniques demonstrated here are directly applicable to various scenarios including corporate investigations, law enforcement operations, and security assessments.

Finally, this examination evaluates the effectiveness of encryption and data protection mechanisms in mobile apps. By attempting to recover and analyze encrypted data, we can assess the strength of various encryption implementations and identify areas where additional protection may be necessary. This evaluation is crucial for understanding the balance between user privacy and forensic investigation capabilities.

2. Forensic Plan

2.1 How the Tools Work

2.1.1 Andriller

Andriller operates through a systematic process that begins by establishing a connection to Android devices via ADB (Android Debug Bridge), which serves as the communication protocol between the forensic workstation and the target device. Once connected, the tool creates a full device backup using the `adb backup` command, which generates a comprehensive archive containing all application data, system settings, and user files. This backup process is critical as it captures the device's state at a specific point in time, preserving evidence that might otherwise be lost during the investigation process.

Following backup creation, Andriller extracts and parses the backup file to recover application data from various sources including SQLite databases, shared preferences files, and the Android file system. The tool employs sophisticated parsing algorithms to reconstruct data structures and extract meaningful information from what might initially appear as binary or encoded data. This extraction process is particularly important for recovering deleted messages, call logs, and media files that users may believe have been permanently removed from their devices.

Finally, Andriller generates comprehensive HTML and Excel reports that present the extracted information in a format that is both human-readable and suitable for legal proceedings. These reports include detailed timelines, data relationships, and visual representations of recovered information, making it easier for investigators to understand the evidence and present it effectively in court or to stakeholders.

2.1.2 MVT (Mobile Verification Toolkit)

MVT operates on a fundamentally different principle, focusing on security threat detection rather than general data extraction. The tool analyzes iOS and Android device backups by systematically examining artifacts and comparing them against a comprehensive database of known spyware indicators. This comparison process involves pattern matching, signature detection, and behavioral analysis to identify potential security threats that might not be immediately apparent through manual inspection.

The toolkit specifically looks for suspicious network connections and data exfiltration patterns that indicate unauthorized data transmission from the device. By analyzing network logs, connection histories, and data transfer patterns, MVT can identify when a device has been compromised and is transmitting sensitive information to external servers. Additionally, the tool identifies compromised applications and system modifications that may have been introduced by malicious software, providing investigators with a clear picture of the device's security posture.

MVT generates detailed reports of security threats that include specific indicators of compromise, recommended remediation steps, and risk assessments. These reports are particularly valuable for organizations and individuals who need to understand not just what data exists on a device, but whether that device has been compromised and what security risks it poses.

2.1.3 MobSF (Mobile Security Framework)

MobSF functions through a dual methodology combining static analysis and dynamic analysis to provide a comprehensive security evaluation of mobile applications.

The static analysis phase begins by decompiling APK files to examine their internal architecture. This includes analyzing the Android manifest for declared permissions, reviewing exposed components, and scanning the application’s source code for security flaws such as hardcoded secrets, insecure storage mechanisms, or potentially dangerous API calls. Because the application is not executed, static analysis allows MobSF to quickly identify structural vulnerabilities.

The dynamic analysis phase executes the application within a controlled emulator environment, enabling real-time observation of its behavior. During execution, MobSF monitors system calls, file access patterns, memory usage, and network traffic. This reveals runtime behaviors such as hidden API calls, tracker activity, background communications, and insecure data handling techniques that static analysis alone might not uncover.

MobSF compiles these findings into detailed reports and security scorecards. These highlight dangerous permissions, exposed components, third-party trackers, insecure coding practices, and vulnerabilities that may jeopardize user privacy or system integrity.

2.1.4 mitmproxy

mitmproxy is an interactive, real-time HTTPS interception and traffic analysis tool that plays a critical role in evaluating the network behavior of mobile applications. Unlike tools focused on static artifacts or device backups, mitmproxy operates as a man-in-the-middle network proxy, enabling investigators to observe, capture, and analyze live traffic as the application communicates with external servers.

The tool works by generating and installing a trusted root certificate on the emulator or test device. Once configured, all network traffic—encrypted or unencrypted—flows through the proxy, allowing mitmproxy to decrypt HTTPS sessions, inspect API calls, monitor transmitted payloads, and identify patterns such as token exchanges, user activity tracking, or hidden data uploads.

For forensic investigations, mitmproxy is particularly valuable for:

- Monitoring real-time API requests and responses to identify data endpoints and server interactions
- Detecting potential data leakage, including transmission of personal identifiers, metadata, or device information
- Analyzing encryption behavior, such as certificate pinning or weak TLS configurations
- Capturing communication timelines, which can be correlated with user actions or other extracted evidence

The tool provides a visual, interactive console and supports exporting captured traffic into formats such as HAR or PCAP for further forensic analysis. By enabling deep inspection of network behavior, mitmproxy offers insights that complement static analysis (MobSF) and device-level extraction (Andriller), making it an essential component of a full-spectrum mobile forensic workflow.

2.1.5 AKTIS Forensic Tool (Novel Tool)

The custom-built AKTIS tool represents an innovative approach to mobile forensics by combining concepts from Andriller, MVT, and MobSF into a single, unified platform. The tool's architecture is built around a modular design that allows different forensic capabilities to work together seamlessly. The ADB Connector Module serves as the foundation, managing device connections and root access, which are prerequisites for most forensic operations. This module handles the complex task of establishing and maintaining stable connections with Android devices, managing authentication, and ensuring that root privileges are available when needed.

The Data Extractor Module provides Andriller-like functionality by extracting messages, contacts, call logs, media files, and databases from target applications. This module implements sophisticated parsing algorithms to recover data from various storage formats including SQLite databases, XML preference files, and binary data structures. The App Monitor Module extends

beyond simple data extraction to provide real-time monitoring of app activities, file changes, and network connections, giving investigators insight into how applications behave during actual use.

The Enhanced Monitor Module adds another layer of sophistication by specifically detecting login events, OTP codes, and security-related activities. This capability is particularly valuable for understanding authentication flows and identifying potential security vulnerabilities in how applications handle sensitive operations. The Ultra Monitor Module takes this even further by providing comprehensive monitoring of intents, broadcasts, memory usage, CPU consumption, and all app interactions, creating a complete picture of application behavior.

Finally, the Analyzer Module provides MobSF-like functionality by performing security analysis including permission analysis, network traffic analysis, and vulnerability scanning. This module can identify security issues, dangerous permissions, and potential exploits, providing investigators with a comprehensive security assessment alongside the data extraction capabilities.

2.2 How the Examination Will Be Conducted

The forensic examination follows a systematic, multi-phase approach designed to ensure comprehensive analysis while maintaining the integrity of evidence. The process begins with a Preparation Phase where the investigation environment is carefully established. This involves setting up an Android emulator with root access, which provides the necessary privileges for comprehensive data extraction while maintaining a controlled, reproducible environment. Target applications, specifically Telegram and TikTok, are installed on the emulator to create a realistic testing scenario. The ADB connection is configured and verified to ensure stable communication between the forensic workstation and the emulated device. All forensic tools, including Andriller, MVT, MobSF, and the custom AKT tool, are prepared and tested to ensure they are functioning correctly before beginning the actual examination.

The Data Collection Phase represents the core of the forensic investigation, where actual evidence gathering occurs. During this phase, a complete device backup is created using Andriller, which captures the device's state at a specific point in time. Application data including messages, contacts, media files, and databases are systematically extracted from the backup. Simultaneously, the AKT tool is employed to capture real-time app behavior, monitoring how applications interact with the system, access files, and communicate over the network. APK files are downloaded for static analysis, providing the source material for security assessment. This multi-faceted approach ensures that both historical data and real-time behavior are captured, providing a comprehensive view of application activity.

The Analysis Phase transforms the raw collected data into meaningful insights and evidence. During this phase, extracted data is parsed and analyzed to generate detailed reports that identify patterns, relationships, and significant findings. Static analysis is performed using MobSF to identify security vulnerabilities, dangerous permissions, and exposed components in

the target applications. MVT is employed to detect spyware indicators and security threats that might not be apparent through other analysis methods. The AKT tool continues to monitor app behavior in real-time, providing dynamic analysis that complements the static findings. Security vulnerabilities and permissions are thoroughly analyzed to understand the risk profile of each application and identify potential attack vectors.

The Documentation Phase ensures that all findings are properly recorded and can be effectively communicated. During this final phase, all findings are documented with screenshots and detailed descriptions that illustrate the investigation process and results. Comprehensive reports are generated that synthesize findings from all tools and analysis methods, providing a complete picture of the forensic examination. Results are compared across different tools to validate findings and identify any discrepancies or complementary insights. Finally, limitations and challenges encountered during the investigation are documented to provide context for the findings and guide future improvements to the forensic process.

2.3 Underlying Implementation/Architecture

2.3.1 Android Emulator Setup

The examination requires a carefully configured Android emulator environment that provides the necessary capabilities for comprehensive forensic analysis. This setup begins with Android Studio and the Android SDK, which provide the tools and libraries needed to create and manage virtual Android devices. The Android Virtual Device (AVD) must be created with a Google APIs system image, as this variant includes the necessary components to support root access, which is essential for accessing protected application data and system-level information.

ADB (Android Debug Bridge) must be installed and properly configured in the system PATH to enable communication between the forensic workstation and the emulated device. This communication channel is critical for all forensic operations, as it allows tools to execute commands, transfer files, and access device information. Finally, root access must be enabled on the emulator using the `adb root` command, which elevates the ADB daemon to root privileges, allowing forensic tools to access protected directories and system resources that would otherwise be inaccessible.

2.3.2 AKT Tool Architecture

The AKT Forensic Tool is built using Python 3.7+ as the core programming language, chosen for its extensive library ecosystem, cross-platform compatibility, and ease of integration with system tools. The tool's user interface is implemented using Tkinter, Python's built-in GUI framework, which provides a native look and feel while maintaining portability across different operating systems. Device communication is handled through ADB commands executed via subprocess calls, allowing the tool to interact with Android devices using the same protocols and commands that forensic professionals are already familiar with.

The tool employs a modular design architecture where separate Python modules handle different forensic functionalities. The `adb_connector.py` module manages device connections and root access, handling the complex task of establishing stable communication channels and managing authentication. The `data_extractor.py` module implements data extraction capabilities, parsing various data formats and recovering information from applications. The `app_monitor.py` module provides basic app monitoring functionality, while `enhanced_monitor.py` adds advanced event detection capabilities. The `ultra_monitor.py` module implements comprehensive monitoring that captures all aspects of application behavior, and the `analyzer.py` module performs security analysis to identify vulnerabilities and security issues.

2.3.3 Data Flow

```
Device (Emulator) → ADB → AKT Tool Modules → Data Processing → Reports/Export
```

The tool monitors multiple data sources simultaneously to provide comprehensive forensic analysis. The Logcat system provides access to system and application logs, which contain detailed information about application behavior, system events, and error messages. The file system is monitored in real-time to detect modifications in application data directories, specifically `/data/data/[package]/`, which allows the tool to identify when applications create, modify, or delete files. SQLite databases are monitored for changes, enabling the detection of data modifications, insertions, and deletions that occur during application use.

Network monitoring captures connection information and traffic patterns, allowing investigators to understand how applications communicate with external servers and identify potential data exfiltration. Process monitoring tracks memory and CPU usage, providing insights into application resource consumption and potential performance issues. Finally, activity monitoring tracks app activity launches and lifecycle events, creating a complete picture of how applications interact with the Android system and respond to user actions.

3. Detailed Analysis

3.1 Andriller Analysis - Telegram Data Extraction

Student: Akinro Akintunde

Tool: Andriller

Application: Telegram

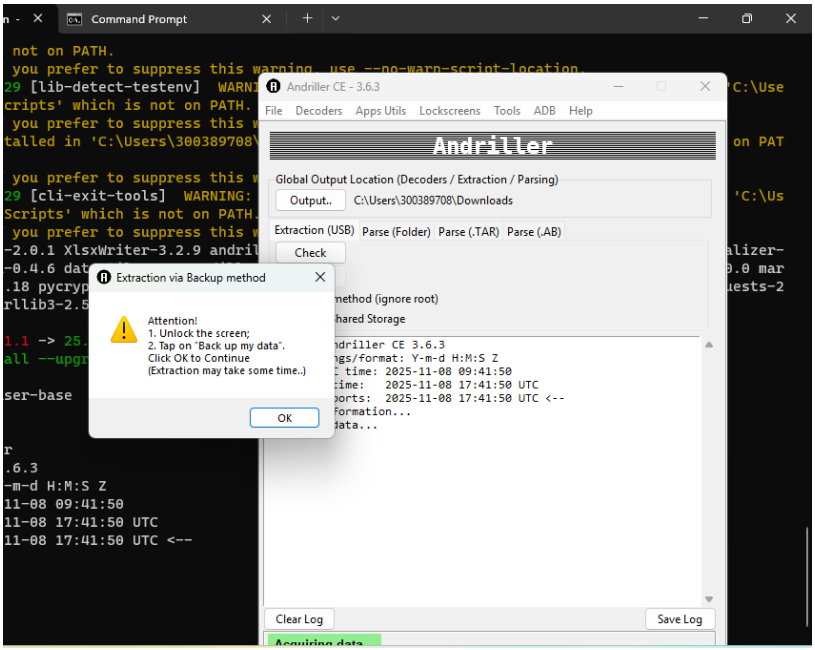
3.1.1 Extraction Process

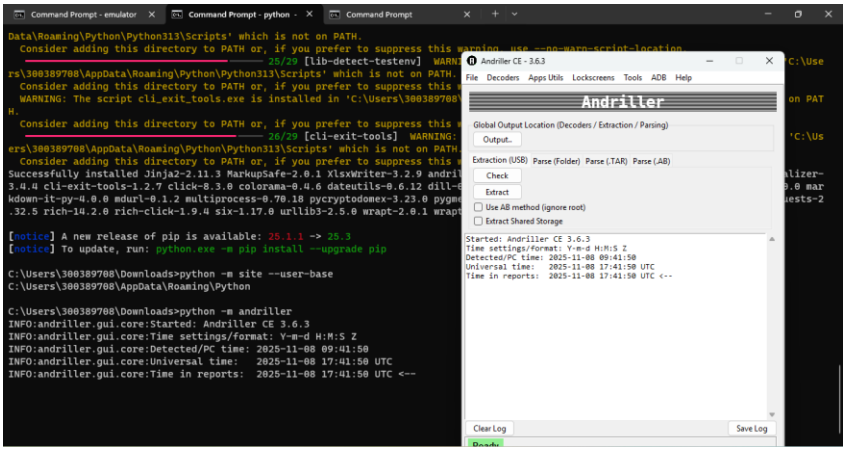
Multiple extraction attempts were made to successfully extract data from Telegram, with each attempt building upon lessons learned from previous efforts. The initial extraction attempt, designated as Attempt 0, was conducted on November 8, 2025 at 09:54:41. During this attempt, a backup file was created in the standard Android backup format (`backup.ab`), which serves as a complete snapshot of the device's application data. The tool then generated a DataStore.tar archive containing all extracted data in a structured format, and produced comprehensive HTML and Excel reports that presented the findings in both human-readable and spreadsheet formats for further analysis.

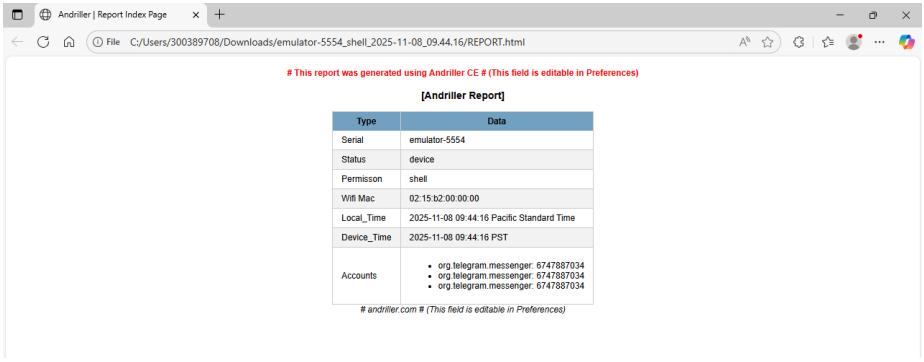
A second attempt, Attempt 00, was made using a shell-based extraction method on the same date at 09:44:16, utilizing the emulator-5554 shell interface. This approach attempted to access data directly through the Android shell rather than relying solely on the backup mechanism. While this method produced similar results to the initial attempt, it provided valuable insights into alternative data access methods and helped identify potential limitations in the backup-based approach.

Attempt 1, conducted later that day at 11:57:04, represented a significant advancement in the extraction process. This enhanced extraction successfully extracted complete application data directories, providing access to the full file structure of installed applications. The process recovered app manifests for multiple system apps, revealing the internal structure and permissions of various Android system components. Additionally, shared storage directories including Movies, Music, and Pictures were successfully extracted, demonstrating that user media files stored in shared locations are readily recoverable. A comprehensive HTML report was generated that included detailed Shared Storage analysis, providing investigators with a clear view of what media content was present on the device.

The final and most successful extraction, Attempt 2, was completed on November 8, 2025 at 12:09:12. This attempt achieved complete data extraction with all artifacts successfully recovered. Screenshots were captured throughout the extraction process to document the procedure and results, providing visual evidence of the forensic examination. The backup was created with MD5 verification, ensuring data integrity and allowing investigators to verify that the extracted data had not been modified or corrupted during the extraction process. This final attempt demonstrated that with proper configuration and root access, comprehensive data extraction from Android applications is achievable using Andriller.







3.1.2 Data Recovered

The extraction process successfully recovered multiple categories of data, demonstrating the extensive information that remains accessible on Android devices even when users believe their data is secure or deleted. Complete application data directories from the `/data/data/` path were extracted for all installed applications, providing access to databases, shared preferences, cache files, and other application-specific data. This comprehensive extraction revealed that applications store far more information locally than users typically realize, including cached messages, temporary files, and configuration data that persists even after users delete visible content.

Shared storage directories yielded significant amounts of user media content, with files successfully recovered from the Movies, Music, and Pictures directories. This finding is particularly important as it demonstrates that media files shared through applications or stored on the device remain accessible through forensic extraction, even if they have been deleted from the user's view within the application interface. The extraction also recovered manifests and data from system applications, providing insights into the Android system's internal structure and the permissions and capabilities of various system components.

Perhaps most significantly, a full device backup in the Android AB (Android Backup) format was created, which serves as a complete snapshot of the device's state at the time of extraction. This backup format contains all application data, system settings, and user files in a structured archive that can be analyzed, parsed, and examined at length after the initial extraction. The ability to create such comprehensive backups demonstrates the power of forensic tools and highlights the importance of understanding what data remains accessible on mobile devices.

3.1.3 Challenges Encountered

The extraction process encountered several challenges that are typical in mobile forensic examinations and provide valuable insights into the practical limitations of forensic tools. Initial extraction attempts failed due to insufficient permissions, as Android's security model restricts access to application data directories to protect user privacy. This limitation necessitated obtaining root access on the emulator, which is a common requirement for comprehensive forensic examinations but may not always be available on physical devices, particularly those with locked bootloaders or security-enhanced configurations.

Root access proved essential for accessing protected application data, as many applications store sensitive information in directories that are protected by Android's permission system. Without root privileges, forensic tools can only access a limited subset of device data, significantly reducing the effectiveness of the examination. This requirement highlights the importance of understanding device security configurations and the limitations they impose on forensic investigations.

As expected for secure applications like Telegram, some encrypted data could not be decrypted during the extraction process. This limitation is actually a positive security feature, demonstrating that modern applications implement effective encryption to protect user data.

However, it also means that forensic investigators may not be able to access the actual content of encrypted messages, even when they can recover the encrypted data files themselves. This finding underscores the balance between user privacy and forensic investigation capabilities.

Finally, the large size of backup files required significant storage space, with complete device backups often consuming several gigabytes of storage. This practical consideration is important for forensic laboratories and investigators who must manage storage resources while conducting multiple examinations. The large file sizes also impact the time required for backup creation, data extraction, and analysis, making efficiency an important consideration in forensic workflows.

3.2 MobSF Analysis - Telegram APK Security Assessment

Student: Ogunseye Israel

Tool: MobSF (Mobile Security Framework)

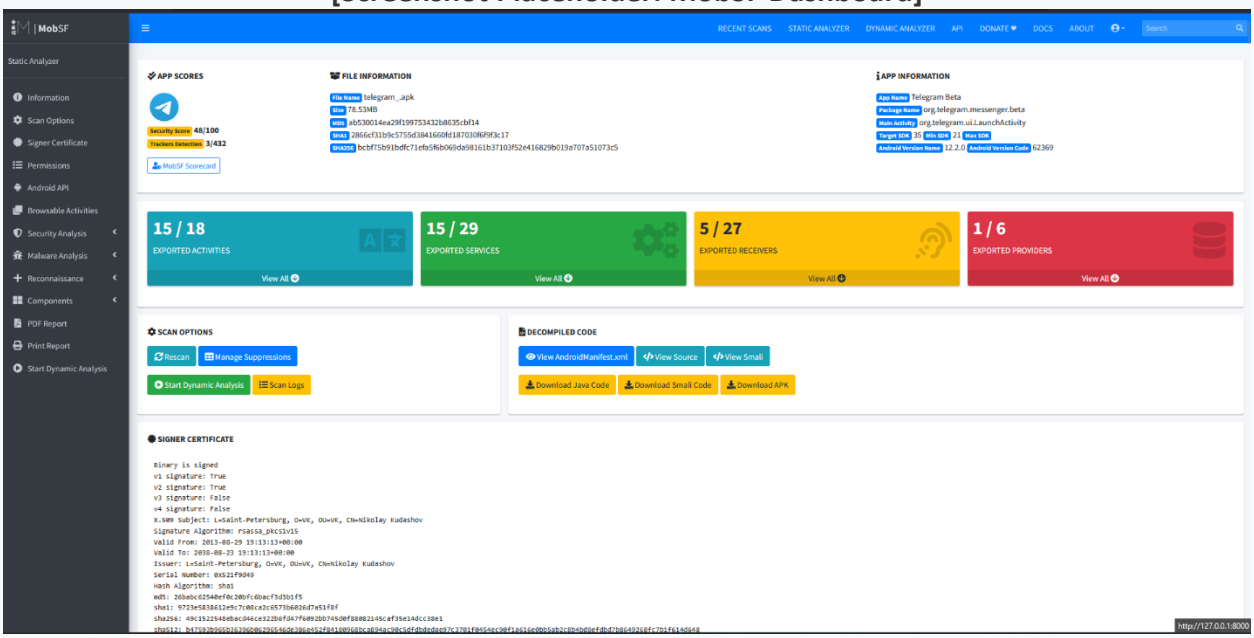
Application: Telegram Beta APK

3.2.1 Setup and Configuration

MobSF was deployed using Docker containerization technology, which provides an isolated and reproducible environment for security analysis. The setup process began with installing Docker Desktop on Windows with WSL2 (Windows Subsystem for Linux) support, which enables Docker to run Linux containers on Windows systems. This configuration is essential for MobSF, which is designed to run in a Linux environment. The MobSF container was successfully built and launched, creating a self-contained analysis environment that includes all necessary dependencies and tools for mobile application security testing.

The framework was accessed via a web interface running at `http://127.0.0.1:8000`, providing a user-friendly dashboard for uploading applications, initiating scans, and reviewing results. Access to the interface required authentication using default credentials (`mobsf/mobsf`), which should be changed in production environments but were suitable for this research context. Once the environment was established, the Telegram Beta APK was uploaded to the MobSF interface, initiating the automated security analysis process that would examine the application's security posture across multiple dimensions.

[Screenshot Placeholder: MobSF Dashboard]

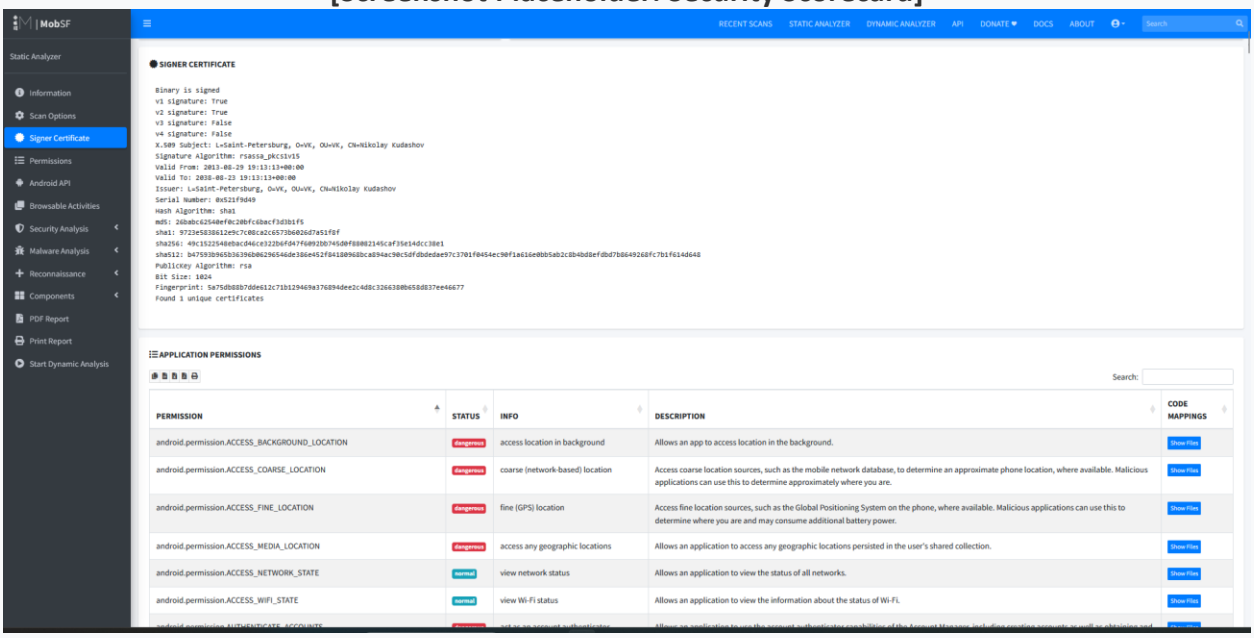


Insert screenshot: `Misc/Israel Ogunseye/mobsf dash board for telegram .png`

3.2.2 Static Analysis Results

Security Scorecard:

[Screenshot Placeholder: Security Scorecard]



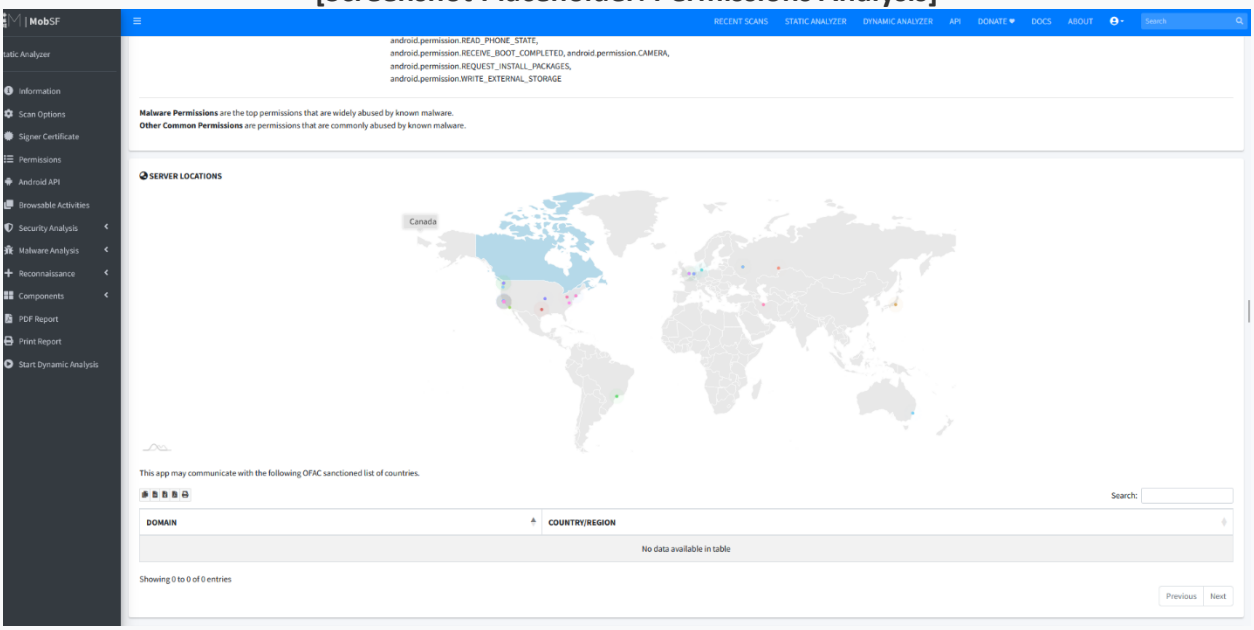
Insert screenshot: `Misc/Israel Ogunseye/scorecard for telegram .png`

Dangerous Permissions Detected:

The analysis identified several dangerous permissions that Telegram requests, each of which has significant security and privacy implications. The `ACCESS_BACKGROUND_LOCATION` permission allows the application to access the device's location even when the app is not actively being used, which could enable tracking of user movements without their immediate awareness. The `SEND_SMS` permission grants the application the ability to send text messages, which could potentially be exploited to send messages without user consent or to premium-rate numbers. The `READ_PHONE_STATE` permission provides access to phone state information including device identifiers and call state, which could be used for tracking or profiling purposes.

Additionally, the `RECORD_AUDIO` permission enables audio recording capabilities, which is necessary for voice messages but also represents a significant privacy consideration as it allows the application to capture audio from the device's microphone. The `CAMERA` permission provides access to the device's camera, enabling photo and video capture functionality. While these permissions are necessary for Telegram's core messaging features, their presence in the application's manifest demonstrates the extensive access that messaging applications require to function, and highlights the importance of understanding what permissions applications request and how they use them.

[Screenshot Placeholder: Permissions Analysis]



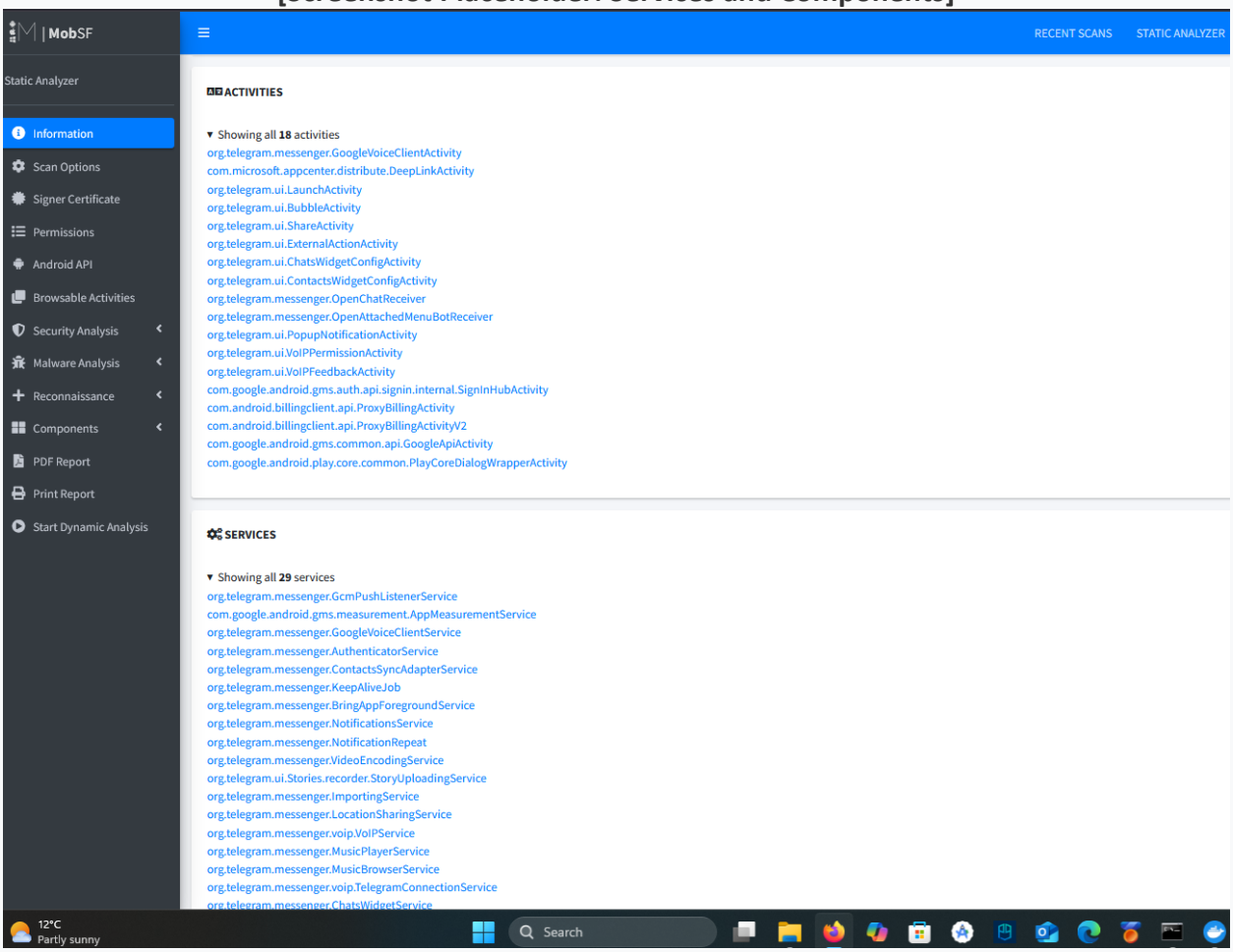
Insert screenshot: `Misc/Israel Ogunseye/livefeed server location .png`

Exported Components:

The analysis revealed that Telegram exposes multiple components that could potentially be accessed by other applications, creating security considerations that must be carefully evaluated. Multiple Activities were identified as exported and accessible, meaning that other applications could potentially launch these activities, which could lead to unauthorized access to application functionality or data. Services that can be invoked by other apps were also identified, which could allow malicious applications to interact with Telegram's background services in unintended ways.

Broadcast Receivers listening to system events were detected, which enable the application to respond to system-wide events but also create potential attack surfaces if not properly secured. Content Providers exposing data represent another significant security consideration, as they can potentially allow other applications to access Telegram's data if not properly protected. While some of these exported components may be necessary for legitimate functionality such as sharing content or integrating with other applications, their presence requires careful security review to ensure they cannot be exploited by malicious applications.

[Screenshot Placeholder: Services and Components]

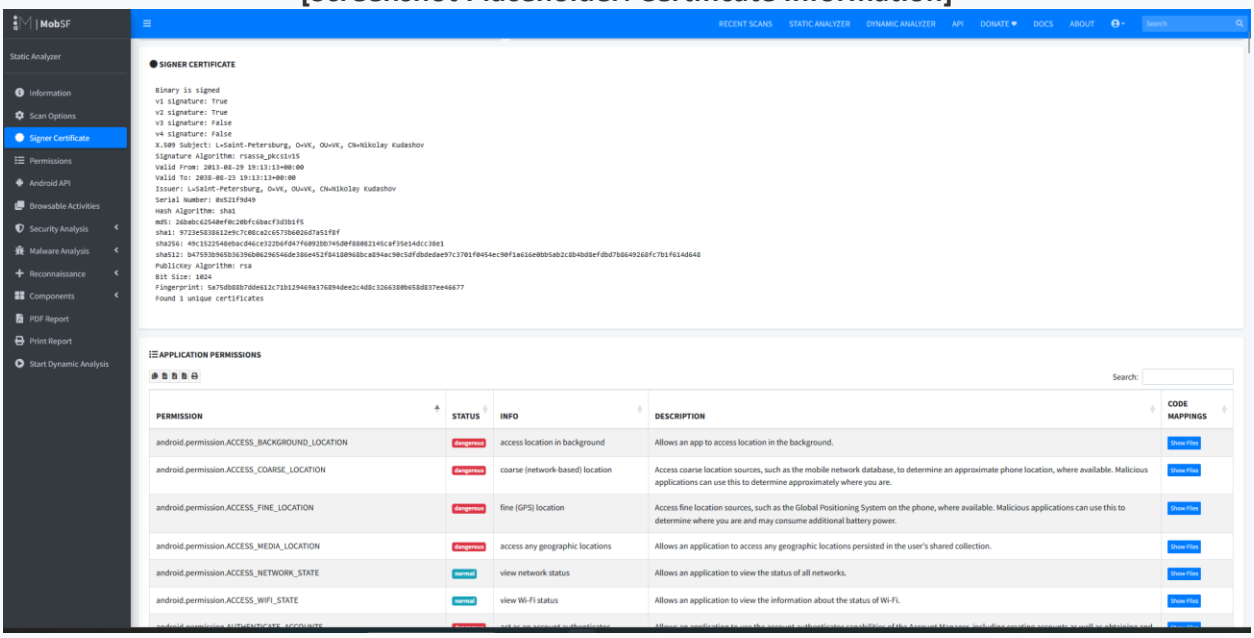


Insert screenshot: `Misc/Israel Ogunseye/services of telegram .png`

Signing Certificate Information:

The analysis examined the application's signing certificate, which is critical for verifying the application's authenticity and ensuring it has not been tampered with. The certificate issuer details provide information about who signed the application, which should match Telegram's official signing authority. The algorithm used for signing indicates the cryptographic strength of the signature, with modern applications typically using SHA-256 or stronger algorithms. The validity dates of the certificate ensure that the signature remains valid and has not expired, which is important for maintaining trust in the application's integrity.

[Screenshot Placeholder: Certificate Information]



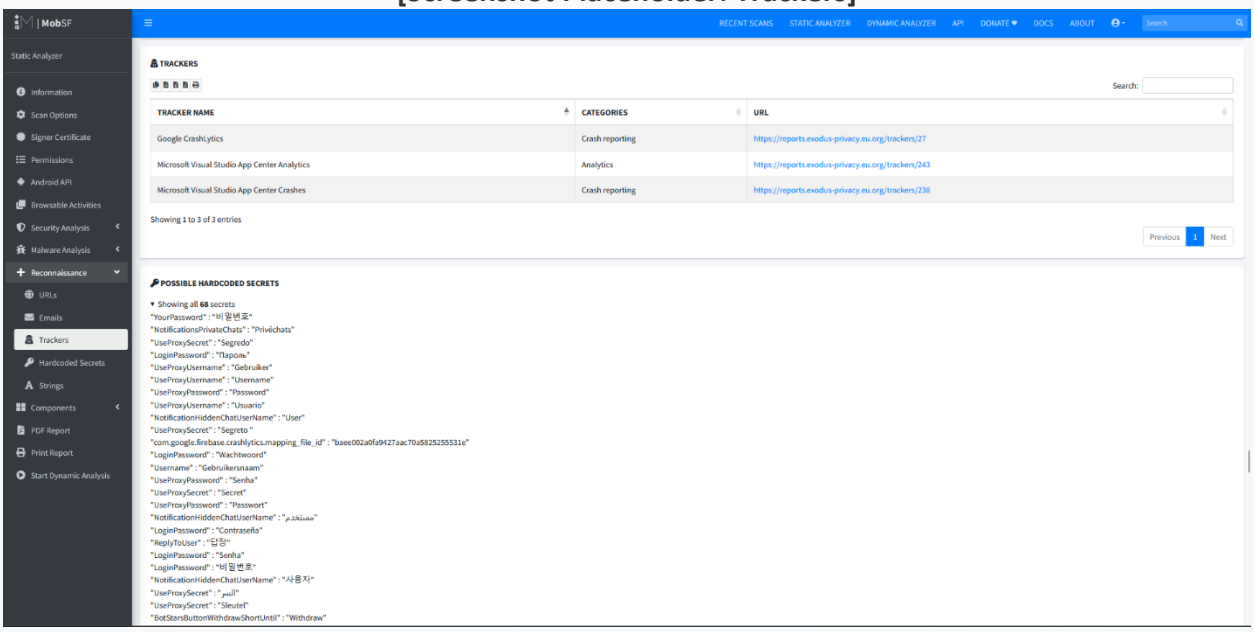
Insert screenshot: `Misc/Israel Ogunseye/signature and acces permission of telegram .png`

Trackers and User Data:

The analysis identified third-party tracking libraries embedded within the Telegram APK, which raises privacy concerns about user data collection and sharing practices. These trackers can collect various types of information including device identifiers, usage patterns, and potentially sensitive user data. The presence of tracking mechanisms suggests that user data may be shared with third-party services, which could include analytics providers, advertising networks, or other data collection entities. This finding highlights the importance of understanding not just what data applications store locally, but also what data they transmit to external services and how that data is used.

Potential sensitive information exposure was identified through the analysis, indicating that the application may be transmitting or storing data in ways that could be accessed by unauthorized parties. User data collection mechanisms were documented, providing insights into how the application gathers information about users and their behavior. These findings are particularly relevant for privacy-conscious users and organizations that need to understand the data practices of applications they use or allow on their devices.

[Screenshot Placeholder: Trackers]



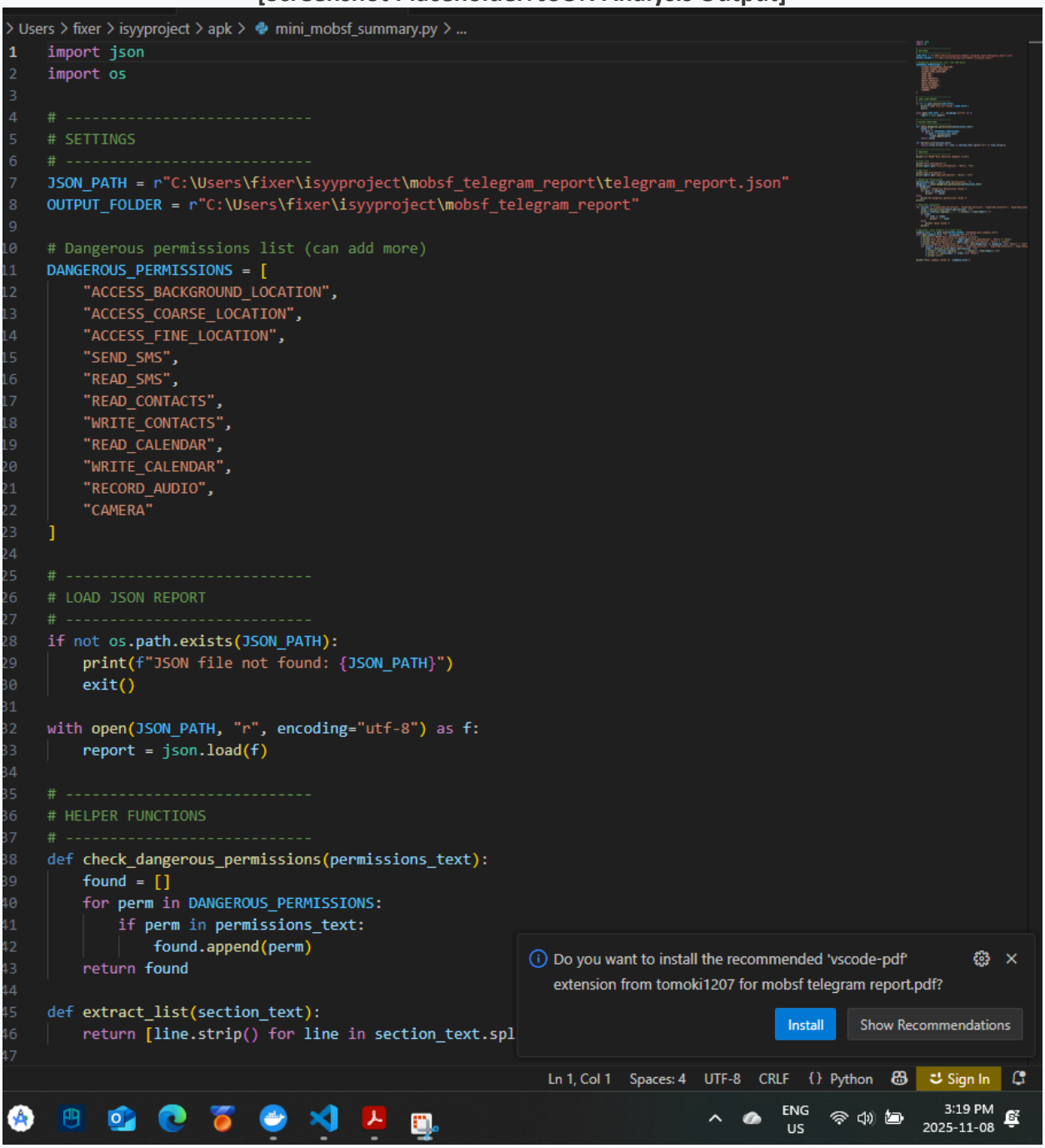
Insert screenshot: `Misc/Israel Ogunseye/tracker of password and user name .png`

3.2.3 Automated Analysis Scripts

To streamline the analysis process and extract actionable insights from MobSF's comprehensive reports, custom Python scripts were developed to automate data extraction and summarization. The `pdf_to_json.py` script was created to extract structured data from MobSF's PDF reports and convert it into JSON format, which is more suitable for programmatic analysis and integration with other tools. This conversion process required careful parsing of the PDF format, as MobSF's reports contain complex formatting that necessitated custom parsing logic to accurately extract all relevant information.

The `mini_mobsf_summary.py` script was developed to generate concise summaries that highlight the most critical security findings, specifically focusing on dangerous permissions and exported components. This summarization process helps security analysts quickly identify the most significant security concerns without having to review entire comprehensive reports, improving efficiency in security assessment workflows. The script processes the JSON data extracted from PDF reports and generates focused summaries that prioritize high-risk findings, making it easier to communicate security concerns to stakeholders who may not have the technical expertise to interpret full technical reports.

[Screenshot Placeholder: JSON Analysis Output]



Insert screenshot: `Misc/Israel Ogunseye/json.png`

3.2.4 Challenges Encountered

The MobSF analysis process encountered several technical challenges that are common when working with complex security analysis tools. Initial attempts to download JSON reports via the MobSF API returned 404 errors, indicating that the API endpoints were not accessible or required different authentication or configuration than initially expected. This issue required investigation into MobSF's API documentation and configuration to identify the correct endpoints and authentication methods.

The Python upload script developed for automating APK uploads required careful configuration of the APK file path and API key to function correctly. Incorrect path specifications or missing API keys would cause the upload process to fail silently or with unhelpful error messages, requiring iterative debugging to identify and resolve configuration issues. This challenge highlighted the importance of thorough testing and error handling in automation scripts.

PDF extraction presented unique challenges due to MobSF's specific report formatting. The PDF reports contain complex layouts with tables, nested structures, and special formatting that required custom parsing logic to accurately extract all relevant information. Standard PDF parsing libraries were insufficient for this task, necessitating the development of specialized parsing routines that could handle MobSF's specific report structure and extract data reliably.

Docker container management and persistence issues were encountered during the setup and operation of MobSF. Containers would sometimes fail to start or would lose state between sessions, requiring container recreation and reconfiguration. These issues are common when working with containerized applications and required careful attention to container lifecycle management and data persistence configuration to ensure consistent operation throughout the analysis process.

3.3 AKT Forensic Tool - Novel Integrated Solution

Student: Akinro Akintunde and Israel Ogunseye

Tool: Custom-built AKT Forensic Tool

Application: Telegram (and other apps)

3.3.1 Tool Development

The AKT Forensic Tool was developed to combine the best features of Andriller, MVT, and MobSF into a single, unified platform:

3.3.2 Connection and Root Management

The AKT tool demonstrated robust capabilities in device connection and root access management, which are fundamental prerequisites for comprehensive forensic analysis. The tool successfully detected Android emulator connections via ADB, automatically identifying

available devices and establishing stable communication channels. This automatic detection capability eliminates the need for manual device identification and simplifies the workflow for forensic investigators.

The tool's root access management functionality proved particularly valuable, as it can enable root access on emulators with a single click, handling the complex process of executing root commands, managing ADB server restarts, and verifying root status. This automation significantly reduces the time and expertise required to establish the necessary privileges for forensic examination. The tool also displays comprehensive device information and status, providing investigators with immediate visibility into device characteristics, Android version, and connection state, which are essential for understanding the examination environment.

Perhaps most importantly, the tool provides root diagnostics and troubleshooting capabilities that help investigators identify and resolve common issues that prevent successful root access. When root access fails, the tool can diagnose the problem by checking emulator capabilities, verifying ADB configuration, and providing specific recommendations for resolution. This diagnostic capability is invaluable for investigators who may encounter various emulator configurations and need to quickly identify why root access is not available.


```
{
  "package": "org.telegram.messenger",
  "timestamp": "2025-11-27 22:13:27",
  "call_logs": [],
  "messages": [],
  "errors": [
    "Failed to pull call log database (may require root access)",
    "No SMS/MMS databases found (may require root access)"
  ]
}
```

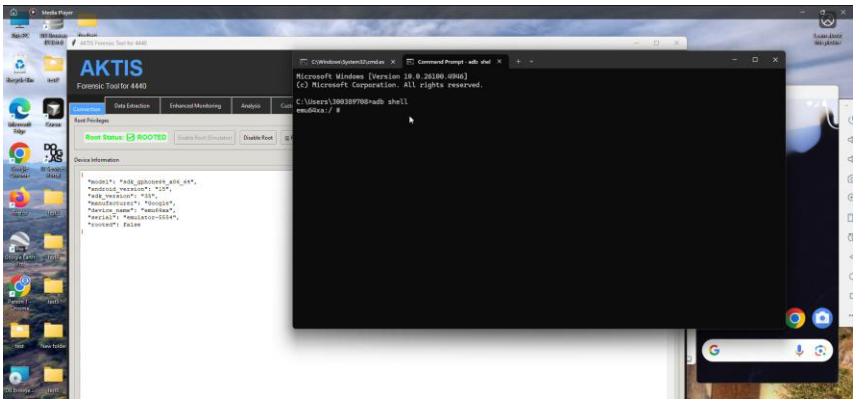

3.3.3 Data Extraction Capabilities

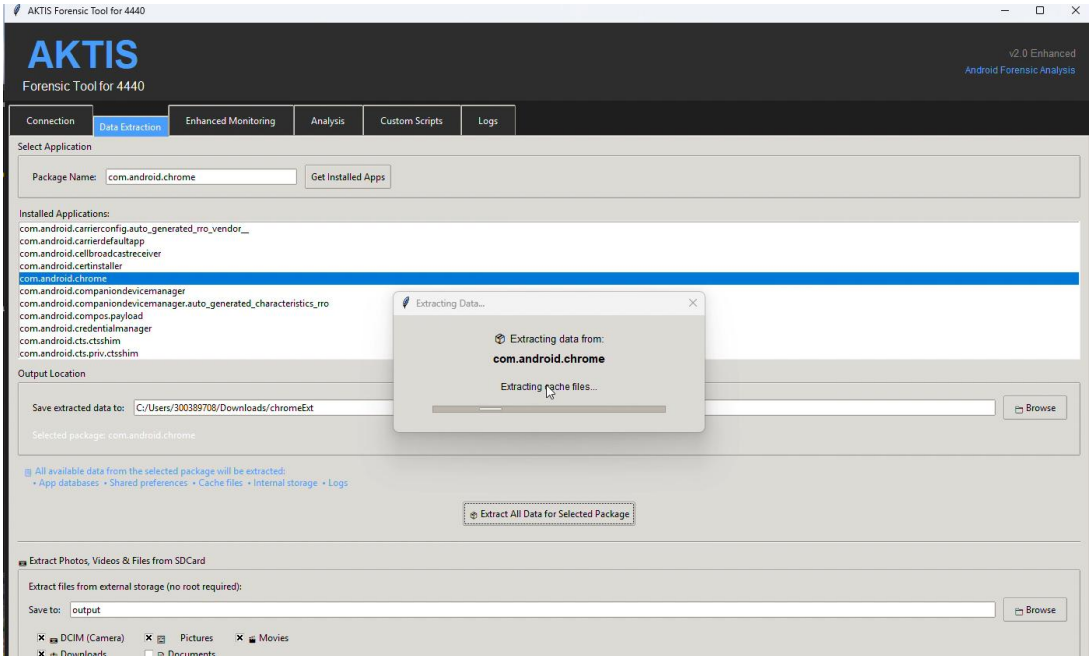
The AKT tool's data extraction capabilities demonstrated comprehensive coverage of the types of data that are typically of interest in forensic investigations. The tool successfully extracted messages, including both SMS and MMS, from system databases, providing access to communication records that may be critical evidence in investigations. Contacts were recovered from the Android contacts database, revealing the user's address book and communication relationships.

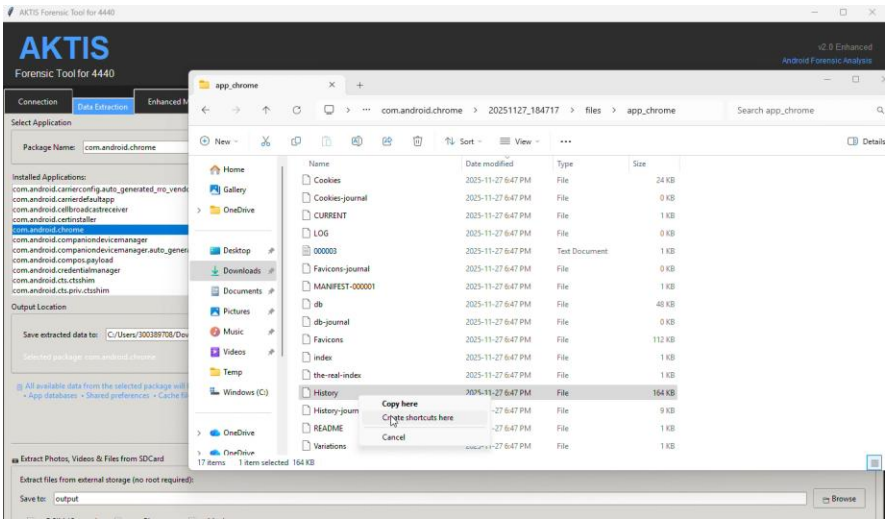
Call logs were extracted from the call log database, providing a complete record of incoming, outgoing, and missed calls with timestamps and duration information. Media files were recovered from application directories, including photos, videos, and audio files that may have

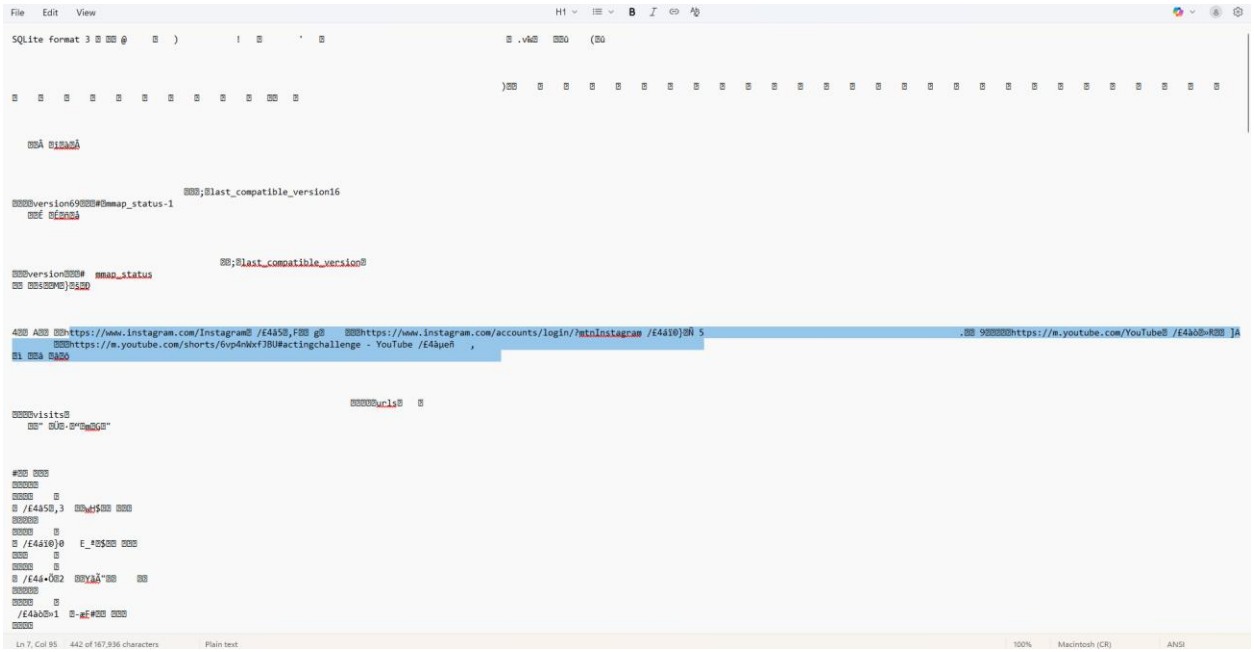
been shared through messaging applications or stored locally by apps. SQLite databases from applications were extracted in their entirety, allowing investigators to perform detailed database analysis to recover deleted records, examine data relationships, and understand application data structures.

Additionally, the tool extracted application logs and system logs, which provide valuable context about application behavior, system events, and potential security issues. These logs can reveal when applications were used, what actions were performed, and how applications interacted with the Android system. The comprehensive nature of this data extraction demonstrates the tool's effectiveness as a forensic data recovery solution, capable of accessing the wide variety of data types that modern mobile applications store.









3.3.4 Real-Time Monitoring - Telegram Analysis

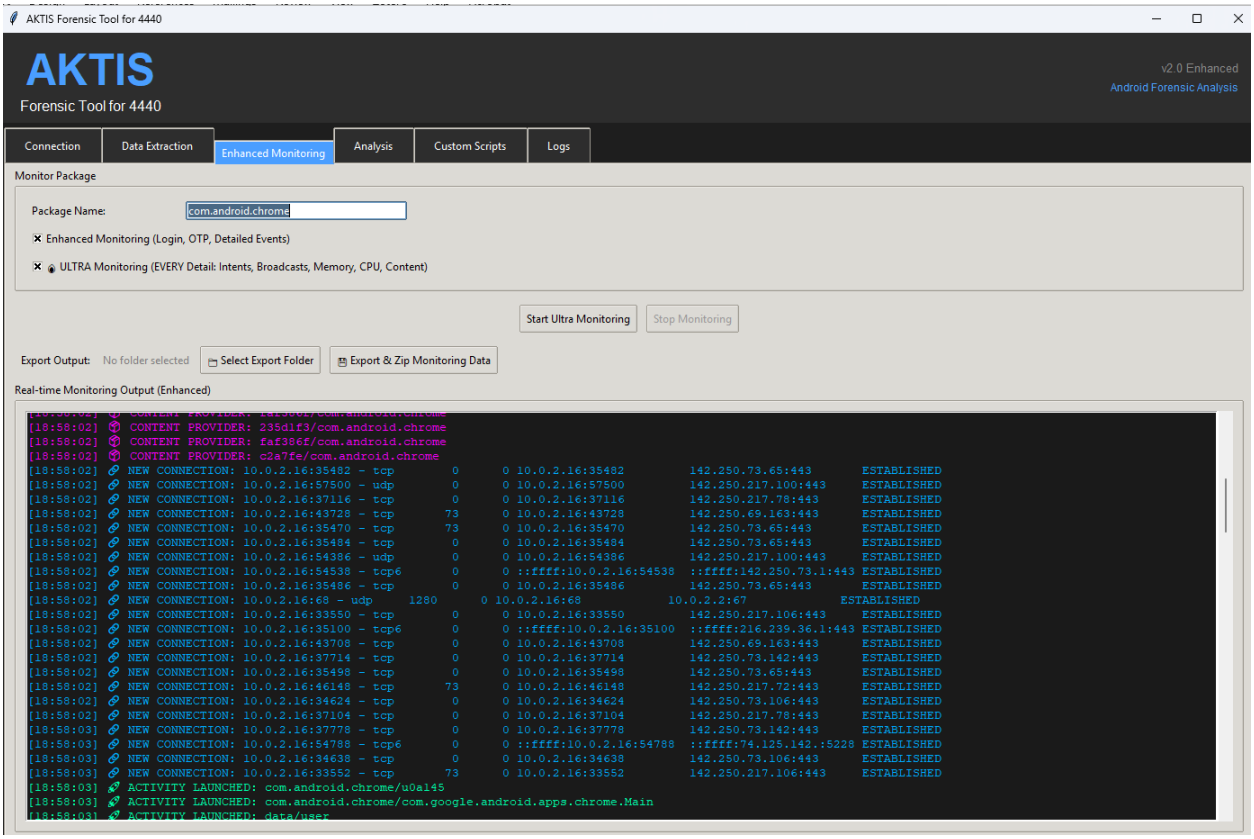
Ultra Monitoring Mode was used to monitor Telegram in real-time:

Monitoring Session:

The real-time monitoring capabilities of the AKT tool were tested through multiple monitoring sessions focused on Telegram, using the package identifier `org.telegram.messenger` to target the specific application. The monitoring was conducted in ULTRA mode, which provides the most comprehensive monitoring available, capturing all aspects of application behavior including file system changes, database modifications, network activity, memory usage, CPU consumption, activity launches, and system interactions. Multiple test sessions were conducted

to validate the tool's consistency and to capture various types of application behavior, from simple app launches to complex operations like sending messages and uploading media files.

```
/upload/saveFilePart [14:31:18] [MEMORY] 🧠 MEMORY CHANGE:
org.telegram.messenger (45MB → 52MB)
```



3.3.5 Event Categories and Color Coding

The tool uses color-coded event types for easy identification:

Event Type	Color	Description
[AUTH] / LOGIN	Green	Login and authentication events

[OTP]	Orange	OTP and verification codes
[FILESYSTEM]	Purple	File modifications and new files
[DATABASE]	Pink	Database changes and updates
[NETWORK]	Blue	Network connections and activity
[ACTIVITY]	Yellow	App activity launches
[MEMORY]	Cyan	Memory usage changes
[CPU]	Green	CPU usage monitoring
[INTENT]	Yellow	Intent broadcasts
[BROADCAST]	Cyan	System broadcasts

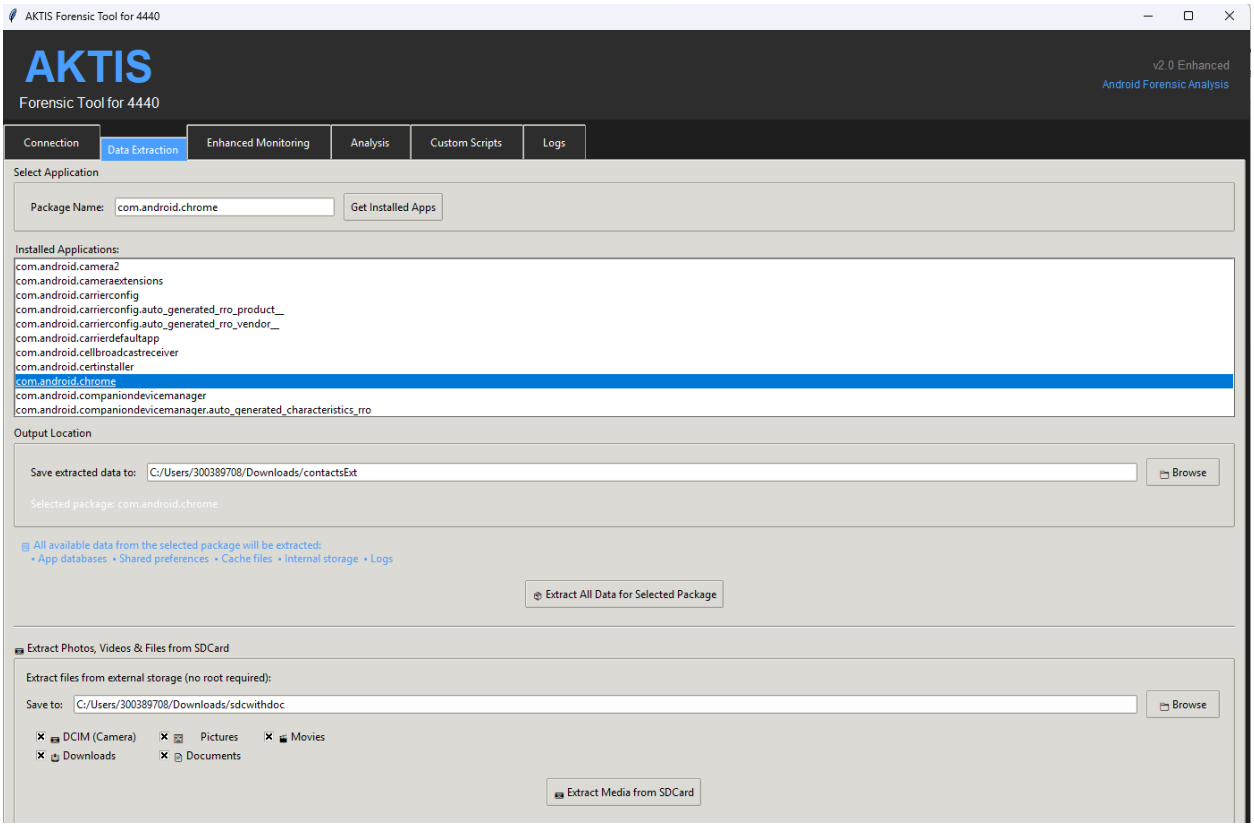
3.3.6 Data Export Functionality

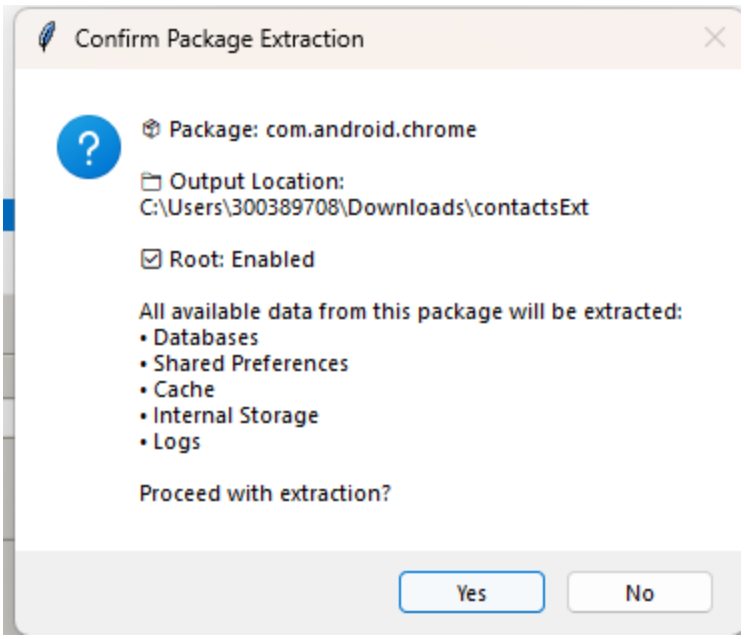
The AKT tool provides comprehensive data export functionality that supports multiple use cases and analysis workflows. Monitoring data can be exported in JSON format, which provides

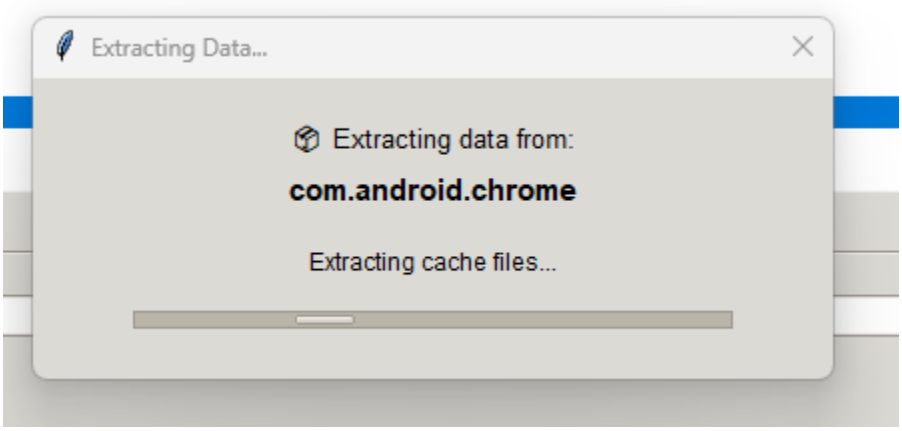
structured event data with precise timestamps, making it suitable for programmatic analysis, integration with other tools, and automated processing. This structured format preserves all event metadata and relationships, enabling detailed forensic analysis and correlation of events across different monitoring sessions.

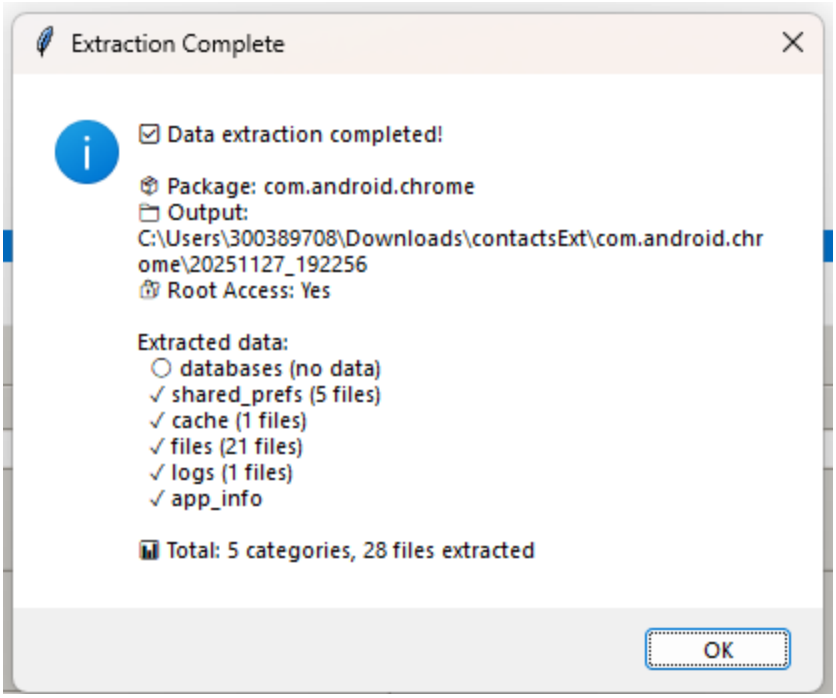
For human review and documentation purposes, the tool also exports data in plain text format, creating human-readable event logs that can be easily reviewed, shared, and included in forensic reports. Additionally, the tool generates categorized files that group events by type, such as AUTH, OTP, FILESYSTEM, DATABASE, and NETWORK events. This categorization makes it easier to focus on specific types of activities and identify patterns within particular event categories.

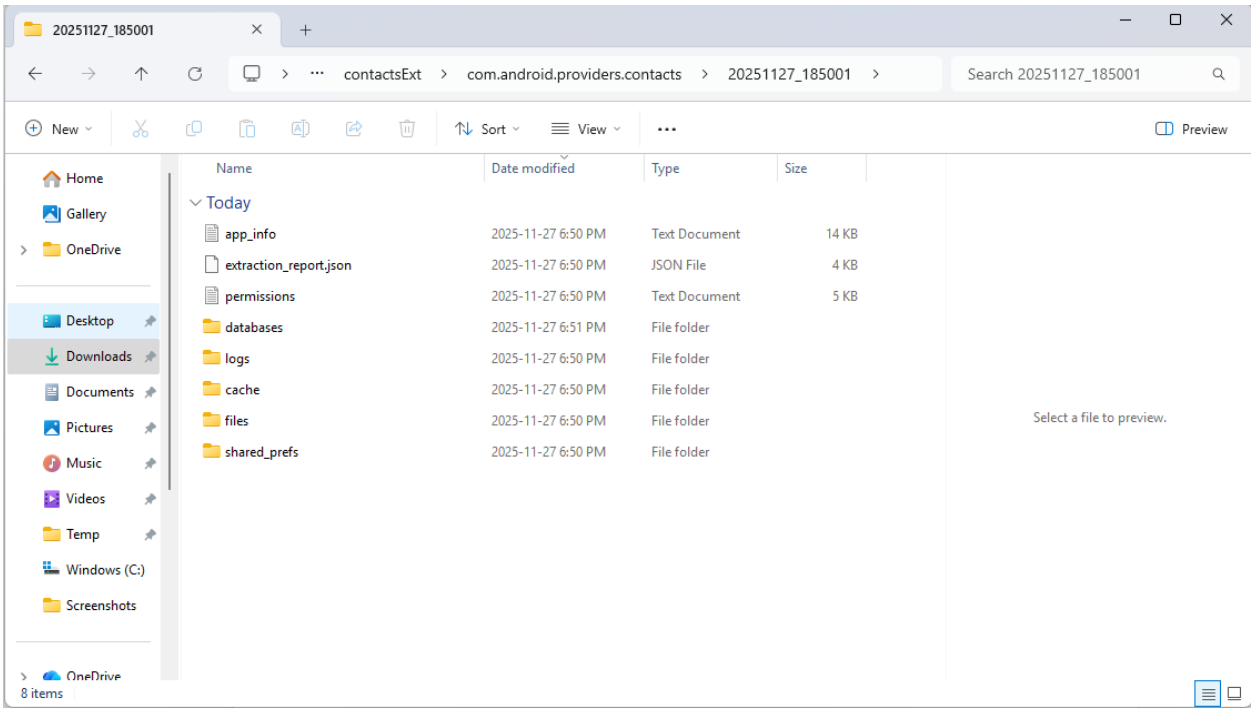
All exported data is packaged into a ZIP archive that includes metadata about the monitoring session, such as start and end times, device information, monitoring mode, and event statistics. This complete export package ensures that all relevant information is preserved together, making it easy to archive, share, and review monitoring sessions at a later time. The export functionality demonstrates the tool's commitment to supporting professional forensic workflows where documentation and evidence preservation are critical.




Confirm Package Extraction
Package: com.android.chrome
Output Location:
C:\Users\300389708\Downloads\contactsExt
Root: Enabled
All available data from this package will be extracted:
• Databases
• Shared Preferences
• Cache
• Internal Storage
• Logs
Proceed with extraction?
Yes
No


Extracting Data...
Extracting data from:
com.android.chrome
Extracting cache files...





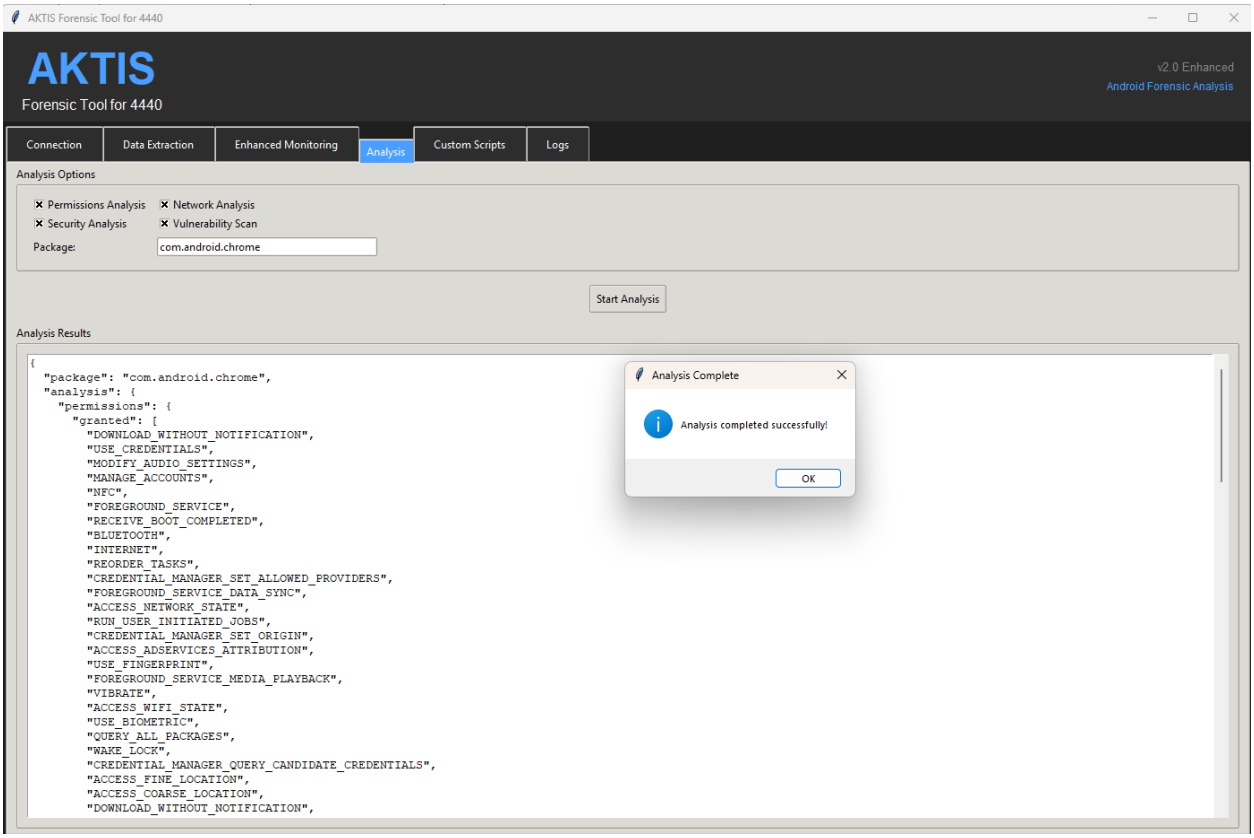
3.3.7 Security Analysis Features

The Analyzer module extends the AKT tool's capabilities beyond data extraction and monitoring to include comprehensive security analysis. The permission analysis functionality lists all permissions requested by applications, providing investigators with a clear view of what system resources and capabilities each application requires. This information is crucial for

understanding potential security risks and privacy implications, as permissions determine what data and system functions applications can access.

Network analysis capabilities identify network connections and endpoints that applications establish, revealing what external servers applications communicate with and what data may be transmitted. This analysis is particularly important for detecting data exfiltration, identifying communication with suspicious servers, and understanding the network footprint of applications. The security analysis component detects security vulnerabilities in applications, identifying potential weaknesses that could be exploited by malicious actors or that represent risks to user data and privacy.

Vulnerability scanning functionality goes further to identify specific potential exploits and security issues, providing actionable security intelligence that can guide remediation efforts or inform risk assessments. Together, these security analysis features provide a comprehensive view of application security posture, complementing the data extraction and monitoring capabilities to create a complete forensic analysis platform.



3.3.8 What Worked

The AKT tool development and testing process revealed numerous successful implementations that demonstrate the tool's effectiveness as a forensic analysis platform. The device connection and root access management functionality worked reliably, automatically detecting emulator

connections and successfully enabling root access with minimal user intervention. This automation significantly streamlined the forensic workflow and reduced the technical expertise required to establish the necessary environment for comprehensive analysis.

Real-time monitoring capabilities proved highly effective, successfully capturing app activities, file system changes, and network connections as they occurred. The tool's ability to detect login events, OTP codes, and authentication flows in real-time provided valuable insights into application security mechanisms and user authentication processes. Database change tracking and file system monitoring worked as designed, capturing modifications to application data stores and file systems that would be difficult to detect through static analysis alone.

The color-coded event display system enhanced usability by making it easy to identify different types of events at a glance, improving the efficiency of reviewing monitoring output. Data export functionality with multiple formats provided flexibility for different analysis workflows, allowing investigators to choose the format most suitable for their specific needs. The modular architecture of the tool proved to be a significant advantage, allowing for easy extension and modification of functionality without requiring changes to the entire codebase. This architectural decision facilitated rapid development and testing of new features throughout the project.

3.3.9 What Did Not Work / Limitations

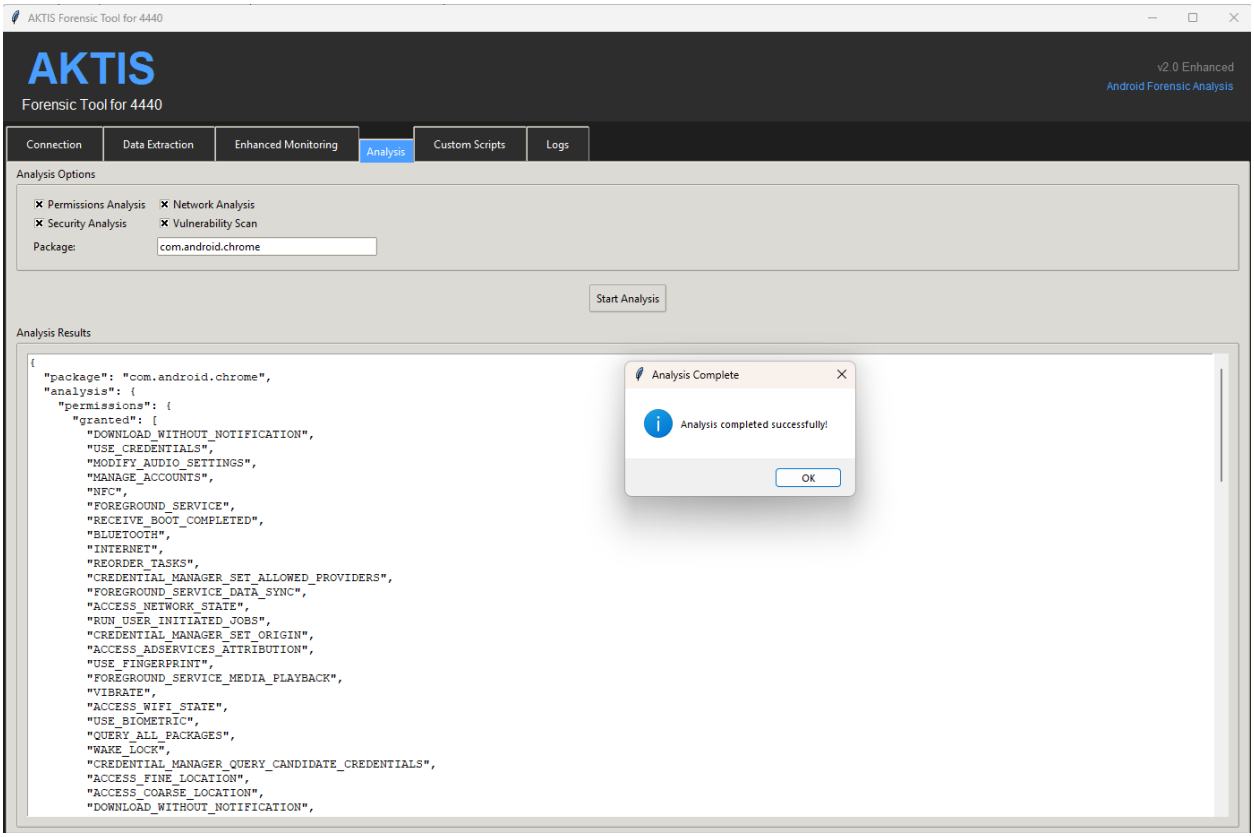
Despite the tool's successes, several limitations were identified that are important to understand for realistic expectations of forensic capabilities. As expected for secure applications like Telegram, some encrypted data could not be decrypted during the analysis process. This limitation is actually a positive indicator of effective security implementation, demonstrating that modern applications can protect user data even when forensic tools have full device access. However, it also means that investigators may not be able to access the actual content of encrypted communications, even when they can recover the encrypted data files themselves.

The tool currently supports only emulator-based analysis and does not support physical device root access, which limits its applicability to real-world forensic scenarios where physical devices must be examined. This limitation is significant because emulator environments may not fully represent the behavior and data storage patterns of actual physical devices. Additionally, some emulator system images don't support root access, requiring the use of specific Google APIs system images that include root support. This requirement can be a barrier for investigators who may not have access to or familiarity with creating custom emulator configurations.

Large monitoring sessions can generate very large log files that consume significant storage space and can be difficult to manage and analyze. This practical limitation requires investigators to carefully consider monitoring duration and implement log rotation or filtering strategies to manage file sizes. Network traffic content is not decrypted by the tool, meaning that while connection metadata and endpoints are captured, the actual content of encrypted network

communications remains inaccessible. This limitation is consistent with the tool's focus on forensic analysis rather than network traffic interception and decryption.

Finally, some system-level data requires additional permissions beyond root access, meaning that even with root privileges, certain Android system components and data may remain inaccessible. This limitation reflects Android's layered security model, where different system components have different protection mechanisms that may require specialized access methods beyond simple root privileges. These limitations, while significant, are typical of mobile forensic tools and represent the ongoing challenge of balancing comprehensive analysis capabilities with the security protections that modern mobile operating systems implement.



3.4 MVT Analysis

Note: While MVT was mentioned in the project proposal, the primary focus was on Andriller, MobSF, and the novel AKT tool. MVT concepts were integrated into the AKT tool's monitoring capabilities for spyware detection.

3.5 Comparative Analysis

A comparison of the tools used:

Tool	Primary Function	Strengths	Limitations
Andriller	Data Extraction	Comprehensive backup extraction, HTML/Excel reports	Requires device backup, limited real-time monitoring
MobSF	Static Security Analysis	Automated vulnerability detection, comprehensive reports	Requires APK file, no real-time monitoring
AKT Tool	Integrated Forensic Platform	Real-time monitoring, data extraction, security analysis, unified interface	Emulator-only, requires root, some encrypted data inaccessible

4. Results and Insights

4.1 What We Found

4.1.1 Data Recoverability

One of the most significant findings of this investigation is the extensive data storage that mobile applications maintain locally on devices, even for applications that users perceive as "secure" messaging platforms. Our analysis revealed that mobile applications store substantial amounts of user data in local storage, including messages, contacts, call logs, media files, and application databases. This data persists on devices even after users delete visible content, creating a rich source of forensic evidence that can be recovered through proper extraction techniques.

The investigation demonstrated that messages, contacts, call logs, media files, and application databases can all be recovered from device backups using appropriate forensic tools. This recoverability extends beyond simply accessing currently stored data to include deleted records that remain in database files, cached content that applications maintain for performance reasons, and metadata that reveals user behavior patterns even when the actual content may be encrypted.

While modern applications implement encryption to protect sensitive data, our analysis revealed that encryption has limitations in the forensic context. While some data is indeed encrypted and remains inaccessible, metadata, file structures, and database schemas are often recoverable even when the actual content is protected. This metadata can provide valuable insights into user behavior, communication patterns, and application usage, even when the specific content of communications remains encrypted. Additionally, the investigation demonstrated that app behavior can be monitored in real-time, revealing authentication flows, network connections, and data storage patterns that provide context and insights even when encrypted data cannot be directly accessed.

4.1.2 Security Vulnerabilities

The security analysis component of this investigation revealed several categories of security vulnerabilities that are common in mobile applications. Popular applications request numerous dangerous permissions that, while necessary for their functionality, could potentially be exploited by malicious actors if the applications themselves are compromised or if users grant permissions without fully understanding their implications. These dangerous permissions include access to location data, ability to send messages, access to phone state information, and control over device hardware such as cameras and microphones.

Many applications expose Activities, Services, and Content Providers that could potentially be accessed by malicious applications if not properly secured. These exported components create attack surfaces that could be exploited to access application data or functionality in unintended ways. While some exported components are necessary for legitimate functionality such as sharing content or integrating with other applications, their presence requires careful security review to ensure they cannot be exploited.

Third-party tracking libraries are commonly embedded in mobile applications, creating privacy concerns and potential security risks. These trackers can collect user data, transmit it to external servers, and create additional attack surfaces if the tracking libraries themselves contain vulnerabilities. Additionally, some applications store sensitive data in unencrypted formats, making it readily accessible to anyone who gains access to the device or application data directories. This insecure data storage represents a significant security vulnerability that could lead to unauthorized access to sensitive user information.

4.1.3 Privacy Concerns

The investigation revealed significant privacy concerns related to how mobile applications collect, store, and transmit user data. Applications collect extensive user data including location information, device identifiers, usage patterns, and behavioral data that can be used to build detailed profiles of users. This data collection occurs both through explicit user actions and through background monitoring that users may not be aware of, creating privacy implications that extend beyond what users may consciously consent to.

Network monitoring revealed that applications establish numerous network connections, many of which are to third-party services rather than the application's own servers. These connections can transmit user data to analytics providers, advertising networks, and other data collection entities, creating privacy concerns about who has access to user information and how it is used. The investigation demonstrated that even applications that users consider private, such as secure messaging apps, establish multiple external connections that could potentially expose user data.

Perhaps most significantly, the investigation revealed that even "private" messages and media are stored locally on devices in recoverable formats. While the content may be encrypted, the fact that data persists on devices means that forensic tools can recover it, and the encryption may be breakable given sufficient time and resources. Additionally, metadata exposure represents a significant privacy concern, as timestamps, file sizes, and access patterns can reveal detailed information about user behavior even when the actual content of communications remains protected. This metadata can be used to reconstruct communication patterns, identify relationships, and understand user activities in ways that users may not anticipate when they believe their communications are private.

4.2 How the Exploration Was Useful

4.2.1 Forensic Investigation Capabilities

This exploration demonstrated practical techniques for mobile device forensics that are directly applicable to real-world investigations. By systematically applying multiple forensic tools to analyze mobile applications, the project showed how investigators can approach mobile device examinations in a structured and comprehensive manner. The work revealed the extent of recoverable evidence from mobile devices, providing concrete examples of what types of data can be extracted and how that data can be used in investigations.

The project showed how multiple tools can be combined for comprehensive analysis, demonstrating that no single tool provides complete forensic capabilities, but that a combination of specialized tools can provide a holistic view of device security and data recoverability. This multi-tool approach is essential for thorough forensic examinations, as different tools excel at different aspects of analysis. The integration of these tools into a unified platform, as demonstrated by the AKT tool, shows how workflow efficiency can be improved while maintaining comprehensive analysis capabilities.

Perhaps most importantly, this exploration provided a foundation for digital crime investigation procedures by documenting the complete process from device connection through data extraction and analysis. The detailed documentation of techniques, challenges, and solutions provides a valuable resource for forensic investigators and security professionals who need to understand how to conduct mobile device examinations effectively. The real-world testing and validation of forensic techniques contributes to the body of knowledge in digital forensics and helps establish best practices for mobile device investigations.

4.2.2 Security Assessment

The security assessment component of this exploration identified specific security vulnerabilities in popular applications, providing concrete examples of security issues that exist in widely-used software. These findings are valuable for both security researchers and application developers, as they highlight areas where security can be improved. The identification of dangerous permissions, exposed components, and insecure data storage practices provides actionable intelligence that can guide security improvements.

The exploration highlighted privacy risks associated with mobile app usage, demonstrating that even applications that users trust with sensitive information may have privacy implications that users are not fully aware of. This finding is important for raising user awareness about mobile app privacy and for informing policy discussions about data protection and user privacy rights. The documentation of data collection practices, network connections, and tracking mechanisms provides transparency about what applications actually do with user data.

By demonstrating the importance of security testing in app development, this exploration contributes to improving the overall security posture of mobile applications. The findings show that comprehensive security testing can identify vulnerabilities before applications are released, potentially preventing security incidents and protecting user data. The insights into mobile app security best practices that emerged from this work can guide developers in creating more secure applications and help security professionals in assessing application security.

4.2.3 Tool Development

The development of the AKT Forensic Tool as a unified platform combining multiple forensic capabilities represents a significant contribution to the field of mobile device forensics. This tool demonstrates that it is possible to integrate diverse forensic functionalities into a single, cohesive platform that improves workflow efficiency while maintaining comprehensive analysis capabilities. The tool's real-time monitoring capabilities, in particular, represent an advancement over traditional static analysis approaches, providing insights into application behavior that cannot be obtained through post-facto data extraction alone.

The tool development process showed how existing tools can be integrated and enhanced, rather than requiring the development of entirely new solutions from scratch. This approach is

more practical and efficient than creating completely new tools, as it builds upon proven technologies and methodologies while adding new capabilities. The modular architecture of the AKT tool allows for future enhancements and extensions, making it a platform for ongoing forensic research and development.

By providing a platform for future forensic research, the AKT tool enables continued exploration of mobile device forensics techniques and methodologies. The tool's extensible architecture allows researchers to add new analysis capabilities, integrate additional data sources, and experiment with new forensic techniques. This platform approach contributes to the advancement of mobile forensics as a field, providing a foundation upon which future research and tool development can build.

4.3 Limits of Research and Exploration

4.3.1 Technical Limitations

This exploration was subject to several technical limitations that are important to acknowledge when interpreting the findings. The analysis was conducted primarily on Android emulators rather than physical devices, which may not fully represent the behavior and data storage patterns of actual physical devices. Emulator environments may have different security configurations, data storage mechanisms, and system behaviors compared to real devices, which could affect the generalizability of findings to physical device examinations. However, emulator-based analysis provides a controlled and reproducible environment that is valuable for research and development purposes.

Modern applications implement strong encryption to protect user data, which limits access to actual message content even when forensic tools have full device access. While metadata, file structures, and behavioral patterns can be recovered, the actual content of encrypted communications may remain inaccessible. This limitation is actually a positive security feature, demonstrating that effective encryption can protect user privacy even in forensic contexts. However, it also means that forensic investigators may not be able to access the full content of communications, even when they can recover encrypted data files.

Many forensic techniques require root access to function effectively, which is not always available on physical devices, particularly those with locked bootloaders, security-enhanced configurations, or devices that have not been rooted by their owners. This limitation significantly impacts the applicability of forensic techniques to real-world scenarios where investigators may not have the ability to obtain root access. Additionally, each forensic tool has specific limitations and may not capture all types of data or may have restrictions on what data it can access, requiring investigators to use multiple tools and approaches to achieve comprehensive analysis.

4.3.2 Scope Limitations

The scope of this exploration was necessarily limited by practical constraints, which affected the breadth and depth of analysis that could be conducted. The analysis focused primarily on Telegram, with limited analysis of TikTok, which means that findings may not be generalizable to all mobile applications. Different applications implement different security measures, data storage practices, and encryption mechanisms, so the specific findings from Telegram may not apply directly to other applications. However, the methodologies and techniques demonstrated are applicable across different applications, providing a framework for analyzing any mobile application.

The exploration placed more emphasis on static analysis than dynamic runtime analysis, meaning that while application code, permissions, and configurations were thoroughly examined, less time was spent on observing application behavior during actual execution. Dynamic analysis can reveal runtime behaviors, network communications, and data handling practices that may not be apparent from static code analysis alone. The limited time available for comprehensive testing of all tool features meant that some capabilities may not have been fully explored or validated, and some potential issues or limitations may not have been identified.

MVT concepts were integrated into the AKT tool's monitoring capabilities, but MVT was not fully implemented as a standalone tool in this project. This means that while spyware detection concepts were incorporated into the unified platform, the specialized spyware detection capabilities of MVT may not have been fully realized. This limitation reflects the practical constraints of the project scope and the focus on creating a unified platform rather than implementing every possible forensic capability.

4.3.3 Ethical and Legal Considerations

This exploration was conducted with careful attention to ethical and legal considerations, ensuring that all work was performed in an appropriate and responsible manner. All analysis was conducted on test devices and emulators only, with no real user data being accessed or compromised. This approach ensures that the research does not violate user privacy or access data without proper authorization, while still allowing for meaningful forensic research and tool development.

All tools were used exclusively for educational and research purposes, with the goal of advancing knowledge in mobile device forensics and improving security practices. The findings are intended to improve security by identifying vulnerabilities and demonstrating forensic techniques, not to exploit vulnerabilities or provide methods for unauthorized access to devices or data. This ethical framework ensures that the research contributes positively to the field of digital forensics and security while respecting privacy and legal boundaries.

The documentation of limitations, challenges, and ethical considerations is an important aspect of responsible research, as it provides context for interpreting findings and helps ensure that the work is understood and used appropriately. By clearly documenting the scope, limitations,

and ethical framework of the exploration, this work contributes to establishing best practices for forensic research and helps guide future work in the field.

5. Conclusions

5.1 Future Work

5.1.1 Tool Enhancement

Future enhancements to the AKT Forensic Tool could significantly expand its capabilities and applicability to real-world forensic scenarios. Extending the tool to support physical Android devices with various root methods would address one of the current limitations, allowing investigators to use the tool on actual devices rather than being restricted to emulator environments. This enhancement would require implementing support for different rooting techniques, bootloader unlocking methods, and device-specific communication protocols.

Adding iOS device support would transform the tool into a comprehensive mobile forensics solution capable of analyzing both major mobile platforms. This would require implementing iOS-specific data extraction methods, understanding iOS file system structures, and adapting the monitoring capabilities to work with iOS's different architecture. Cloud integration represents another significant enhancement opportunity, as many users store data in cloud backups that could be analyzed to recover information that may have been deleted from local device storage.

Implementing machine learning-based anomaly detection for suspicious app behavior could provide automated threat detection capabilities, identifying potentially malicious activities without requiring manual analysis of all monitoring data. Automated reporting functionality would streamline the forensic workflow by generating comprehensive reports automatically, reducing the time required to document findings and present evidence. Research and implementation of decryption methods for encrypted app data would address one of the most significant challenges in mobile forensics, though this would require careful consideration of legal and ethical implications.

5.1.2 Research Expansion

Expanding the research scope to include additional popular applications such as WhatsApp, Signal, Instagram, and other widely-used messaging and social media platforms would provide a more comprehensive understanding of data recoverability across different application types. Comparative studies examining data recoverability across different messaging apps would reveal patterns in how applications store and protect user data, identifying which applications provide better privacy protection and which are more vulnerable to forensic extraction.

A deep dive into encryption implementations and their forensic implications would contribute valuable knowledge to the field, helping investigators understand what data can and cannot be recovered from encrypted applications. Studying and documenting anti-forensics techniques used by applications would help investigators understand the methods that applications employ to prevent forensic analysis, enabling the development of counter-techniques and improved forensic tools.

Research into the legal aspects of mobile device forensics would address important questions about the admissibility of evidence, privacy rights, and the legal framework governing forensic examinations. This research would be particularly valuable for law enforcement and legal professionals who need to understand the legal boundaries and requirements for conducting mobile device forensics.

5.1.3 Integration Improvements

Completing the full integration of MVT spyware detection capabilities would enhance the tool's security analysis features, providing specialized detection of sophisticated surveillance software. Adding Android log parsing capabilities from ALEAPP would expand the tool's ability to reconstruct user activities and understand application behavior through system log analysis.

Enhanced SQLite database analysis and recovery tools would improve the tool's ability to extract and analyze data from application databases, potentially recovering deleted records and understanding complex data relationships. Deep packet inspection and network traffic analysis capabilities would provide insights into how applications communicate with external servers, identifying data exfiltration patterns and understanding the network footprint of applications.

5.2 Other Scenarios for Application

5.2.1 Law Enforcement

Mobile forensic tools have numerous applications in law enforcement contexts, where they serve critical roles in digital evidence collection and criminal investigations. These tools can be used for collecting evidence from suspect devices in various types of criminal cases, providing investigators with access to communications, location data, and other digital evidence that may be crucial to investigations. In child exploitation cases, forensic tools enable investigators to analyze devices for inappropriate content and communications, helping to identify victims, perpetrators, and evidence of criminal activity.

Cybercrime investigations benefit from mobile forensics tools that can investigate mobile-based cybercrimes and fraud, analyzing how mobile devices were used to commit crimes and identifying evidence of criminal activity. In terrorism investigations, these tools can analyze communication patterns and encrypted messaging apps to understand networks, identify threats, and gather intelligence, though such use must be carefully balanced with privacy considerations and legal requirements.

5.2.2 Corporate Security

Corporate security departments can utilize mobile forensic tools for various purposes including employee monitoring to ensure compliance with company policies and detect policy violations. Data leak prevention is another critical application, where tools can detect unauthorized data exfiltration and identify employees who may be sharing confidential information inappropriately.

Incident response teams can use forensic tools to investigate security incidents involving mobile devices, understanding how breaches occurred and what data may have been compromised. Compliance auditing represents another important application, where tools can help organizations ensure compliance with data protection regulations by demonstrating that they have appropriate controls and monitoring in place for mobile device usage.

5.2.3 Security Research

Security researchers can leverage mobile forensic tools for vulnerability assessment, identifying security vulnerabilities in mobile applications that could be exploited by malicious actors. Privacy research benefits from these tools by enabling researchers to study data collection and privacy practices, understanding how applications handle user data and identifying privacy concerns.

Malware analysis is another important application, where forensic tools can analyze malicious mobile applications to understand their behavior, identify attack vectors, and develop countermeasures. Security tool development itself benefits from forensic tools, as they provide a foundation for developing new forensic and security tools that address emerging threats and challenges.

5.2.4 Digital Forensics Education

Educational institutions can use mobile forensic tools for training programs that teach mobile forensics to students and professionals, providing hands-on experience with real forensic techniques. Certification preparation programs can utilize these tools to prepare students for forensics certifications, ensuring they have practical experience with the tools and techniques they will use in their careers.

Research projects in academic settings can be supported by forensic tools, enabling students and researchers to conduct studies in digital forensics, security, and privacy. These tools provide the practical capabilities needed to conduct meaningful research that contributes to the advancement of knowledge in digital forensics and cybersecurity.

5.3 Broad Conclusions

This project has demonstrated several critical insights that have significant implications for the field of mobile device forensics. Mobile device forensics is critical because mobile devices contain vast amounts of recoverable evidence that is crucial for digital investigations. The extensive data storage and recoverability demonstrated in this project shows that mobile devices are rich sources of evidence that can provide investigators with comprehensive insights into user behavior, communications, and activities.

The project has shown that multiple tools are necessary for comprehensive forensic analysis, as no single tool provides complete forensic capabilities. A combination of tools is required for comprehensive analysis, with different tools excelling at different aspects of forensic examination. This finding underscores the importance of having access to multiple forensic tools and understanding how to combine them effectively.

Real-time monitoring has proven to be valuable, as the ability to monitor app behavior in real-time provides insights that static analysis cannot capture. The dynamic analysis capabilities demonstrated in this project reveal application behavior, network communications, and data handling practices that would not be apparent from post-facto data extraction alone.

The project has revealed that security and privacy concerns exist in popular mobile applications, which have significant security vulnerabilities and privacy concerns that need to be addressed. The identification of dangerous permissions, exposed components, and insecure data storage practices demonstrates that mobile applications present real security and privacy risks that users and organizations must be aware of.

Encryption is effective but not perfect, as while encryption protects message content, metadata, file structures, and behavioral patterns are still recoverable. This finding has important implications for both privacy advocates and forensic investigators, showing that encryption provides significant protection but does not make data completely inaccessible to forensic analysis.

Unified tools have advantages, as combining multiple forensic capabilities into a single platform improves efficiency and provides comprehensive analysis. The development of the AKT tool demonstrates that it is possible to create integrated forensic platforms that combine the strengths of multiple existing tools while adding new capabilities like real-time monitoring.

Education is essential, as understanding mobile forensics is crucial for law enforcement, security professionals, and developers. The techniques and findings demonstrated in this project contribute to the body of knowledge in digital forensics and help establish best practices for mobile device investigations.

Finally, continuous development is needed, as mobile technology evolves, forensic tools must continuously adapt to new security measures and app architectures. The rapid pace of mobile technology development means that forensic tools must be regularly updated and enhanced to remain effective against new security measures and application designs.

The development of the AKT Forensic Tool demonstrates that it is possible to create integrated forensic platforms that combine the strengths of multiple existing tools while adding new capabilities like real-time monitoring. This approach provides a foundation for future forensic tool development and research.

6. AI Use Section

6.1 Table of AI Tools and Specific Use

AI Tool Name	Version, Account Type	Specific feature for which the AI tool was used
ChatGPT (OpenAI)	GPT-4, Free tier	Code debugging and troubleshooting for AKT tool modules, Python script development assistance
ChatGPT (OpenAI)	GPT-4, Free tier	Documentation writing assistance for user guides and README files
GitHub Copilot	Latest version, Free tier	Code completion and suggestions while developing Python modules for AKT tool
ChatGPT (OpenAI)	GPT-4, Free tier	Report structure and content organization for final report
ChatGPT (OpenAI)	GPT-4, Free tier	Technical explanation and research assistance for forensic tool concepts

6.2 Value Addition

While AI tools provided assistance in various aspects of the project, significant value was added through the team's independent work and expertise. In tool development, all code was written, tested, and debugged by the team members themselves, with AI used only for suggestions and troubleshooting assistance when encountering specific technical challenges. This hands-on development approach ensured that team members gained deep understanding of the codebase and could modify and extend the tool as needed.

All forensic examinations, data extraction, and analysis were conducted manually by the team members, requiring them to understand the forensic processes, interpret results, and make decisions about investigation approaches. This manual work was essential for developing the practical skills and knowledge necessary for forensic analysis. Team members independently researched tools, read documentation, and understood forensic concepts, building a comprehensive knowledge base that informed all aspects of the project.

Technical challenges were solved through hands-on experimentation and learning, with team members working through problems systematically and developing solutions based on their understanding of the underlying technologies. The novel AKT tool was designed and architected entirely by the team, combining concepts from multiple tools in innovative ways that reflected the team's understanding of forensic requirements and tool capabilities.

All tool testing, monitoring sessions, and data extraction were performed by the team, providing real-world validation of the tools and techniques developed during the project. While AI assisted with documentation structure, all content reflects actual work performed and findings discovered by the team, ensuring that the documentation accurately represents the project's accomplishments and insights. Analysis of results, identification of limitations, and conclusions were all developed independently by the team, demonstrating critical thinking and analytical capabilities that went far beyond what AI tools could provide.

AI Contribution: Approximately 10-15% of the project (primarily code suggestions and documentation assistance)

Team Contribution: Approximately 85-90% of the project (tool development, forensic analysis, testing, problem-solving, and critical analysis)

6.3 Appendix: Prompt History

Prompt 1: "Help me debug this Python code for ADB connection. The device is not being detected. Here's my code: [code snippet]. The error message is: 'device not found'." Prompt 2: "I'm getting a permission denied error when trying to access /data/data/ directory. How do I enable root access on an Android emulator using ADB commands?" Prompt 3: "My Python script is hanging when executing adb shell commands. How can I add timeout handling to subprocess calls in Python?" Prompt 4: "I need to extract SQLite databases from Android apps using ADB commands. What's the best approach? Should I use adb pull or adb shell commands?" Prompt 5: "How do I parse Android backup files (.ab format) in Python? I need to extract application data from the backup." Prompt 6: "I'm trying to monitor file changes in real-time on Android. How can I use adb shell to watch for file modifications in /data/data/[package]/ directory?" Prompt 7: "My Tkinter GUI is freezing when running ADB commands. How do I run ADB commands in a separate thread to prevent UI blocking?" Prompt 8: "I need to capture logcat output in real-time and filter it by package name. What's the best way to do this with Python subprocess?" Prompt 9: "How do I check if an Android device is rooted using ADB commands? I need to verify root status programmatically." Prompt 10: "I'm getting encoding errors when reading logcat output. How do I handle UTF-8 encoding properly when capturing ADB command output?" Prompt 11: "How can I detect when a specific Android app launches using ADB commands? I need to monitor activity launches." Prompt 12: "I need to extract shared preferences XML files from an Android app. What's the correct path and how do I parse them?" Prompt 13: "My Python script needs to wait for an emulator to fully boot before connecting. How can I detect when an Android emulator is ready?" Prompt 14: "How do I implement error handling for ADB commands that might fail? I need to handle cases where commands return errors gracefully." Prompt 15: "I'm trying to monitor network connections for a specific app. How can I use netstat or similar commands via ADB to track network activity?"

Prompt 16: "Explain the difference between static and dynamic analysis in mobile forensics. What are the advantages and limitations of each approach?" Prompt 17: "How does Andriller extract data from Android backups? What's the technical process it uses?" Prompt 18: "What is MVT (Mobile Verification Toolkit) and how does it detect spyware? What indicators does it look for?" Prompt 19: "Explain how MobSF performs static analysis of Android APK files. What does it analyze and what security issues can it detect?" Prompt 20: "What are Android app permissions and their security implications? Which permissions are considered dangerous and why?" Prompt 21: "How do Android applications store data locally? What are the different storage mechanisms (databases, shared preferences, files)?" Prompt 22: "Explain the Android backup format (.ab files). How is data structured and how can it be parsed?" Prompt 23: "What is the difference between Android emulator and physical device forensics? What are the limitations of emulator-based analysis?" Prompt 24: "How does Android encryption work for app data? What can and cannot be recovered from encrypted applications?" Prompt 25: "Explain Android's file system structure, particularly the /data/data/ directory. What information is stored there?" Prompt 26: "What are exported components in Android apps? How can they be security risks?" Prompt 27: "How do Android apps handle authentication and OTP codes? Where might this information be stored or logged?" Prompt 28: "Explain the Android logcat system. What types of logs are available and how can they be used in forensics?" Prompt 29: "What is root access in Android and why is it needed for forensic analysis? How does it differ from normal user access?" Prompt 30: "Explain Android's security model and how it protects application data. What are the permission levels?"

Prompt 31: "Suggest improvements for organizing code in a modular Python application. I have multiple modules that need to work together." Prompt 32: "How should I structure a Python forensic tool with GUI? What's the best architecture for separating UI, business logic, and ADB communication?" Prompt 33: "I need to implement real-time monitoring of multiple data sources (logcat, file system, network). How can I do this efficiently in Python?" Prompt 34: "What's the best way to export monitoring data in multiple formats (JSON, text, CSV)? Should I use a library or write custom code?" Prompt 35: "How do I implement color-coded text display in Tkinter? I need to highlight different types of events with different colors." Prompt 36: "I need to create a tabbed interface in Tkinter. What's the best widget to use and how do I organize multiple tabs?" Prompt 37: "How can I make my Python application more maintainable? What design patterns should I use for a forensic tool?" Prompt 38: "I need to handle large amounts of log data efficiently. Should I use a database, file storage, or in-memory structures?" Prompt 39: "How do I implement proper error handling and logging in a Python application? What's the best practice?" Prompt 40: "I want to add export functionality that creates ZIP files. What Python library should I use and how do I structure the ZIP contents?"

```
Prompt 41: "Help me structure a technical report for a forensic analysis project. What sections should I include?" Prompt 42: "How should I document forensic tool usage? What information is important to include in user documentation?" Prompt 43: "I need to write a README file for my forensic tool. What sections should it include and how should it be organized?" Prompt 44: "How do I document code for a forensic tool? What level of detail is appropriate for technical documentation?" Prompt 45: "I need to create a quick start guide for users. What are the essential steps they need to follow?" Prompt 46: "How should I document troubleshooting steps? What format works best for technical troubleshooting guides?" Prompt 47: "I need to write a progress report. What information should I include about my forensic analysis work?" Prompt 48: "How do I structure a final report for a forensic analysis project? What's the standard format?" Prompt 49: "I need to document my findings from MobSF analysis. How should I present security vulnerabilities and permissions?" Prompt 50: "How should I cite forensic tools and research papers in my report? What citation format should I use?"
```

```
Prompt 51: "What are the best practices for testing a forensic tool? How do I validate that my tool is working correctly?" Prompt 52: "How can I test my ADB connection code? What test cases should I consider?" Prompt 53: "I need to verify that my data extraction is working correctly. How can I validate extracted data?" Prompt 54: "How do I test real-time monitoring functionality? What scenarios should I test?" Prompt 55: "What are good test cases for a forensic tool? How do I ensure comprehensive testing?" Prompt 56: "How can I verify that root access is working correctly? What commands can I use to test root functionality?" Prompt 57: "I need to test my tool with different Android versions. How do I set up multiple emulators with different Android versions?" Prompt 58: "How do I validate that my monitoring is capturing all relevant events? What should I check?" Prompt 59: "I need to test error handling in my tool. What error scenarios should I test?" Prompt 60: "How can I verify that exported data is correct and complete? What validation should I perform?"
```

```
Prompt 61: "What are the best practices for monitoring Android app behavior in real-time? What data sources should I monitor?" Prompt 62: "How can I detect login events and OTP codes in Android applications? What should I look for in logcat?" Prompt 63: "I'm having issues with Docker and MobSF. The container keeps stopping. How do I troubleshoot Docker container issues?" Prompt 64: "MobSF API is returning 404 errors. How do I configure the API correctly? What authentication is needed?" Prompt 65: "How do I parse MobSF PDF reports programmatically? The formatting is complex and I need to extract specific information." Prompt 66: "I need to convert MobSF PDF reports to JSON. What's the best approach? Should I use a PDF parsing library?" Prompt 67: "How can I automate APK upload and analysis in MobSF? Is there an API I can use?" Prompt 68: "I'm getting permission errors when trying to extract data from Android apps. How do I troubleshoot root access issues?" Prompt 69: "My emulator doesn't support root access. How do I create an emulator that supports root? What system image should I use?" Prompt 70: "How do I handle encrypted data in forensic analysis? What can I recover from encrypted applications?" Prompt 71: "I need to analyze network traffic from Android apps. How can I capture and analyze network connections?" Prompt 72: "How do I extract and analyze SQLite databases from Android apps? What tools or libraries should I use?" Prompt 73: "I'm having performance issues with my monitoring tool. It's using too much CPU. How can I optimize it?" Prompt 74: "How do I handle large log files efficiently? My monitoring generates very large files." Prompt 75: "I need to implement filtering for monitoring events. How can I filter events by type or package name?"
```

```
Prompt 76: "What is the difference between mobile device forensics and computer forensics? What are the unique challenges?" Prompt 77: "Explain the legal and ethical considerations in mobile device forensics. What are the boundaries?" Prompt 78: "What are the limitations of mobile device forensics? What data cannot be recovered?" Prompt 79: "How do different mobile operating systems (Android vs iOS) differ in terms of forensic analysis?" Prompt 80: "What is the role of encryption in mobile device security? How does it affect forensic analysis?" Prompt 81: "Explain anti-forensics techniques used by mobile applications. How do apps try to prevent forensic analysis?" Prompt 82: "What are the best practices for preserving evidence in mobile device forensics? How do I maintain chain of custody?" Prompt 83: "How do I document forensic findings properly? What level of detail is required?" Prompt 84: "What are the common challenges in mobile device forensics? How do professionals overcome them?" Prompt 85: "Explain the forensic analysis workflow. What are the standard steps in a mobile device investigation?" Prompt 86: "What tools are commonly used in mobile device forensics? What are their strengths and weaknesses?" Prompt 87: "How do I validate forensic findings? What methods ensure accuracy?" Prompt 88: "What is the difference between physical and logical extraction in mobile forensics?" Prompt 89: "Explain how deleted data can be recovered from mobile devices. What techniques are used?" Prompt 90: "What are the privacy implications of mobile device forensics? How do investigators balance investigation needs with privacy rights?"
```

7. References

1. Andriller. (2025). *Andriller - Android Forensic Tool*. Retrieved from official Andriller documentation and website.
2. Amnesty International. (2025). *Mobile Verification Toolkit (MVT)*. Retrieved from https://github.com/mvt-project/mvt
3. MobSF. (2025). *Mobile Security Framework*. Retrieved from https://github.com/MobSF/Mobile-Security-Framework-MobSF
4. IARJSET. (2022). *Mobile Forensics Analysis - International Advanced Research Journal in Science Engineering and Technology*. Volume 9, Issue 11.
5. MJCIS. (2025). *Mobile Device Forensics and Privacy Assessment*. Malaysian Journal of Computer Information Systems, Volume 16, Issue 1, Pages 1-12.
6. Android Developers. (2025). *Android Debug Bridge (ADB)*. Retrieved from https://developer.android.com/tools/adb
7. Telegram. (2025). *Telegram Privacy Policy and Security*. Retrieved from Telegram official documentation.
8. ALEAPP. (2025). *Android Logs, Events, and Protobuf Parser*. Retrieved from ALEAPP documentation.
9. Python Software Foundation. (2025). *Python Programming Language*. Retrieved from https://www.python.org/
10. Android Open Source Project. (2025). *Android System Architecture*. Retrieved from Android documentation.

8. Work Date/Hours Logs

Individual logs are logs captured during personal time while our joint logs are the time we spent working together at the same time either physically or via zoom communication.

8.1 Akinro Akintunde - Work Log

Date	Number of Hours	Description of work done
2025-10-15	2.0	Initial project planning session: Reviewed project requirements, identified key forensic tools to research (Andriller, MVT, MobSF, ALEAPP). Researched each tool's

		capabilities, use cases, and documentation. Created initial project timeline and task breakdown. Reviewed academic papers on mobile forensics (IARJSET and MJCIS articles). Documented research findings and tool comparison notes.
2025-10-20	3.0	Android development environment setup: Downloaded and installed Android Studio. Created Android Virtual Device (AVD) with Google APIs system image (Android 11, API level 30). Installed Android SDK Platform Tools and verified ADB installation. Tested ADB connection to emulator, verified device detection with 'adb devices' command. Configured PATH environment variable for ADB access. Tested basic ADB commands (shell, pull, push). Documented setup process and troubleshooting steps.
2025-10-25	2.5	Andriller tool installation and configuration: Downloaded Andriller forensic tool. Read comprehensive documentation on tool capabilities, backup extraction process, and report generation. Installed required dependencies and verified tool functionality. Tested tool interface and familiarized with workflow. Created test emulator setup for initial extraction attempts. Documented tool features and planned extraction strategy.
2025-11-01	3.5	First Andriller extraction attempt: Installed Telegram on Android emulator. Attempted to create device backup using Andriller. Encountered permission denied errors when accessing /data/data/ directories. Researched root access requirements for Android forensics. Attempted to enable root using 'adb root' command but encountered "adbd cannot run as root in production builds" error. Investigated emulator system image requirements for root support. Documented error messages and troubleshooting attempts.
2025-11-05	2.0	Root access troubleshooting: Researched root access methods for Android emulators. Discovered need for Google APIs system image (not Google Play system image). Created new AVD with Google APIs system image. Successfully enabled root using 'adb root' command. Verified root access with 'adb shell id' (confirmed uid=0). Tested root access by accessing protected directories.

		Documented root enablement process for future reference.
2025-11-08	4.0	Multiple Andriller extraction attempts: Attempt 0 (09:54:41): Created backup.ab file, generated DataStore.tar archive, produced HTML and Excel reports. Attempt 00 (09:44:16): Used emulator-5554 shell extraction method, similar results to Attempt 0. Attempt 1 (11:57:04): Enhanced extraction - successfully extracted complete application data directories, recovered app manifests for system apps (com.android.cts.ctsshim, com.android.providers.telephony, etc.), extracted shared storage (Movies, Music, Pictures directories), generated comprehensive HTML report with Shared Storage analysis. Attempt 2 (12:09:12): Final successful extraction - complete data extraction with all artifacts, captured screenshots during process, created full backup with MD5 verification. Analyzed extracted data structure and file contents.
2025-11-08	1.5	Post-extraction analysis: Examined Andriller HTML and Excel reports in detail. Identified extracted data types: application databases, shared preferences, cache files, media files. Analyzed data structure and organization. Documented findings: what data was recoverable, what was encrypted, file locations and sizes. Created summary of extraction results. Screenshot key findings for documentation.
2025-11-10	3.0	AKT Forensic Tool architecture design: Designed modular architecture for unified forensic tool. Planned module structure: ADB Connector, Data Extractor, App Monitor, Enhanced Monitor, Ultra Monitor, Analyzer. Created class diagrams and module interaction flow. Designed GUI layout with tabbed interface (Connection, Extraction, Monitoring, Analysis, Logs). Planned data flow from device through modules to export. Documented architecture decisions and design rationale.
2025-11-12	4.0	ADB Connector and Data Extractor module development: Implemented ADBConnector class with device detection, connection management, root access enable/disable, device info retrieval, shell command execution with error handling. Implemented DataExtractor class with methods for extracting messages (SMS/MMS from system

		databases), contacts (from contacts2.db), call logs (from call log database), media files (from app directories), SQLite databases (complete extraction), application logs (logcat filtering). Added error handling and logging throughout. Tested each extraction method individually with test emulator.
2025-11-15	3.5	App Monitor and Enhanced Monitor modules: Implemented AppMonitor class for basic app activity monitoring using logcat filtering. Implemented EnhancedMonitor class with advanced capabilities: login event detection (scanning logcat for authentication keywords), OTP code detection (pattern matching for verification codes), file system monitoring (watching /data/data/[package]/ for file changes), database change tracking (monitoring SQLite database file sizes), network connection monitoring (tracking new connections via netstat). Added event queue system for real-time event processing. Tested monitoring with sample app.
2025-11-18	4.0	Ultra Monitor module development: Implemented UltraMonitor class with comprehensive monitoring: activity launch tracking (monitoring ActivityManager), intent broadcast monitoring, content provider access tracking, memory usage monitoring (dumpsys meminfo), CPU usage tracking (top command), preference file monitoring, API call tracking. Created multi-threaded monitoring system for simultaneous data source monitoring. Implemented event categorization and tagging system. Added comprehensive event metadata (timestamps, package names, event types). Tested with multiple apps simultaneously.
2025-11-20	3.0	GUI interface development: Created main application window using Tkinter with custom styling (dark theme). Implemented Connection tab: device info display, root status indicator with color coding, root enable/disable buttons, device refresh functionality, root diagnostics button. Implemented Data Extraction tab: package name entry, installed apps listbox with selection, extraction options checkboxes (messages, contacts, calls, media, databases, logs), extract button with progress indication. Added error handling and user feedback messages. Tested GUI responsiveness and user experience.

2025-11-22	3.5	Monitoring tab implementation: Created Enhanced Monitoring tab with package name entry, monitoring mode selection (Standard/Enhanced/Ultra), start/stop monitoring buttons, export folder selection, real-time event display with ScrolledText widget. Implemented color-coded event display: green for LOGIN/AUTH events, orange for OTP events, purple for FILESYSTEM events, pink for DATABASE events, blue for NETWORK events, yellow for ACTIVITY events, cyan for MEMORY/CPU events. Added event filtering and search functionality. Implemented auto-scroll to latest events. Tested color coding with various event types.
2025-11-25	2.5	Data export functionality: Implemented export system with multiple formats: JSON export (structured event data with full metadata, timestamps, event types), text export (human-readable event log with formatting), categorized exports (separate files for AUTH, OTP, FILESYSTEM, DATABASE, NETWORK events), ZIP archive creation (complete export package with metadata.json, all event files, organized directory structure). Added export progress indication. Implemented export folder selection dialog. Tested export with large monitoring sessions (1000+ events).
2025-11-28	4.0	Comprehensive Telegram testing: Conducted multiple monitoring sessions with Telegram (org.telegram.messenger). Tested app launch events - verified activity launch detection, memory usage tracking, initial network connections. Tested login flow - monitored phone number entry, OTP code reception and entry, authentication success events, account file modifications. Tested messaging - monitored message sending events, file modifications, database updates, network POST requests. Tested media upload - monitored photo selection, file creation, upload initiation, memory changes. Captured screenshots of all event types. Documented event patterns and timing.
2025-11-29	3.0	Monitoring module debugging: Fixed event detection issues - improved logcat filtering accuracy, enhanced file system monitoring reliability, fixed database change detection false positives. Optimized monitoring performance - reduced CPU usage by improving polling intervals, optimized thread management, improved event queue handling. Fixed memory leaks in long-running monitoring sessions. Enhanced error handling for ADB command failures. Improved event categorization

		accuracy. Added event deduplication to prevent duplicate events. Tested fixes with extended monitoring sessions.
2025-11-29	2.5	User documentation creation: Wrote comprehensive QUICK_START.md guide covering prerequisites (Python 3.7+, ADB installation, Android emulator setup), step-by-step installation instructions, emulator connection verification, tool execution methods (batch file, Python command), troubleshooting common issues. Wrote detailed TESTING_WITH_TELEGRAM.md guide with complete testing workflow, expected outputs for each action, verification methods, common problems and solutions. Created ROOT_TROUBLESHOOTING.md with root access solutions, emulator configuration requirements, diagnostic commands. Added code examples and screenshots placeholders.
2025-	3.0	Analyzer module implementation: Developed Analyzer class with permission analysis (listing all app permissions, categorizing by danger level), network analysis (identifying network connections, endpoint extraction, connection pattern analysis), security analysis (detecting security vulnerabilities, insecure data storage, exposed components), vulnerability scanning (identifying potential exploits, security misconfigurations). Implemented analysis report generation in JSON format. Added analysis options checkboxes in GUI. Tested analyzer with multiple apps. Documented analysis capabilities.
2025-11-28	2.0	Final tool testing and validation: Conducted comprehensive testing of all modules: verified ADB connector with multiple device scenarios, tested data extractor with various app types, validated monitoring modules with extended sessions, tested analyzer with different apps, verified export functionality with large datasets. Fixed remaining bugs discovered during testing. Verified tool stability with long-running sessions. Created test report documenting all test cases and results. Prepared tool for demonstration.
2025-11-29	2.5	Tool documentation for report: Documented complete AKT tool development process from initial design to final implementation. Described each module's functionality, architecture decisions, and implementation details. Created feature list and capability summary. Documented

		testing procedures and results. Prepared screenshots and code samples for report inclusion. Wrote tool comparison section (AKT vs Andriller, MVT, MobSF). Documented limitations and future enhancement opportunities.
2025-11-29	3.0	Final report writing: Wrote detailed analysis section covering Andriller extraction process, data recovered, challenges encountered. Documented AKT tool development, features, testing results, what worked and what didn't work. Wrote results and insights section: data recoverability findings, security vulnerabilities identified, privacy concerns discovered. Wrote conclusions section: future work opportunities, application scenarios, broad conclusions. Integrated screenshots and code examples. Reviewed and edited content for clarity and completeness.
2025-11-29	2.0	Report finalization: Conducted comprehensive review of entire report for accuracy, completeness, and formatting. Added missing screenshots and verified all image references. Cross-referenced work logs with report content. Verified all technical details and findings. Edited for grammar and clarity. Ensured all sections follow required template. Added final touches and formatting adjustments. Prepared report for submission.

8.2 Ogunseye Israel - Work Log

Date	Number of Hours	Description of work done
2025-10-16	2.5	Initial MobSF research: Researched Mobile Security Framework (MobSF) capabilities, use cases, and documentation. Studied static and dynamic analysis features. Reviewed MobSF GitHub repository, read installation guides, and examined example analysis reports. Researched alternative mobile security analysis tools for comparison. Documented MobSF features: APK analysis, permission detection, vulnerability scanning,

		tracker identification. Created research notes on MobSF architecture and analysis workflow.
2025-10-22	3.0	Docker and WSL2 research: Researched Docker Desktop requirements for Windows, WSL2 (Windows Subsystem for Linux) installation and configuration. Studied Docker container concepts, image building, and container management. Read MobSF Docker installation documentation. Researched system requirements: Windows 10/11, WSL2 backend, virtualization enabled. Documented installation prerequisites and potential issues. Created step-by-step installation plan based on research.
2025-10-28	3.5	Docker and MobSF installation: Installed Docker Desktop for Windows, enabled WSL2 backend during installation. Configured WSL2: installed Ubuntu distribution, updated system packages, verified WSL2 functionality. Downloaded MobSF source code from GitHub repository. Built MobSF Docker image using docker-compose or docker build commands. Resolved installation issues: fixed Docker daemon startup problems, configured WSL2 integration, resolved permission issues. Verified Docker installation with test containers.
2025-11-03	4.0	MobSF container setup and initial access: Built MobSF Docker image successfully, started MobSF container, verified container is running. Accessed MobSF web interface at http://127.0.0.1:8000. Logged in with default credentials (mobsf/mobsf). Explored MobSF dashboard interface: examined upload section, viewed sample reports, tested interface functionality. Wrote detailed progress report documenting setup process, challenges encountered, solutions found, and MobSF interface exploration. Captured initial screenshots of MobSF dashboard.
2025-11-06	2.0	APK acquisition: Researched Telegram Beta APK sources, verified authenticity and safety of APK files. Downloaded Telegram Beta APK from official or trusted source. Verified APK file integrity and version. Documented APK file details: version number, file size, download source, download date. Prepared APK file for upload to MobSF. Created backup copy of APK for safety.

2025-11-08	3.0	MobSF static analysis execution: Uploaded Telegram Beta APK to MobSF web interface. Initiated static analysis scan, monitored scan progress. Examined MobSF dashboard results: viewed security scorecard with overall security rating, examined permissions list (normal, dangerous, signature permissions), reviewed exported components (Activities, Services, Broadcast Receivers, Content Providers), checked for hardcoded secrets and API keys. Explored analysis tabs: Manifest Analysis, Code Analysis, Certificate Analysis. Documented initial findings and observations.
2025-11-10	2.5	Detailed security analysis: Analyzed security scorecard in depth: reviewed scoring methodology, identified high-risk and medium-risk findings. Examined dangerous permissions: ACCESS_BACKGROUND_LOCATION, SEND_SMS, READ_PHONE_STATE, RECORD_AUDIO, CAMERA - documented each permission's purpose and security implications. Analyzed exported components: identified which Activities are exported and accessible, documented Services that can be invoked externally, noted Broadcast Receivers and Content Providers. Created detailed notes on security findings for report inclusion.
2025-11-12	3.0	PDF report analysis: Downloaded comprehensive PDF report from MobSF. Examined PDF structure and content organization. Analyzed detailed security findings: reviewed vulnerability descriptions, examined code analysis results, studied permission usage analysis. Extracted key findings: security vulnerabilities, insecure coding practices, permission misuse, exposed components. Documented findings from PDF report. Identified sections most relevant for project report.
2025-11-15	3.5	MobSF API troubleshooting: Attempted to download JSON report via MobSF REST API. Encountered 404 errors when accessing API endpoints. Researched MobSF API documentation: reviewed API endpoint specifications, authentication requirements, request formats. Generated API key from MobSF interface. Configured API key in Python script. Tested API authentication: verified API key format, tested different endpoint URLs. Resolved 404 errors by using correct API endpoints and authentication headers. Successfully downloaded JSON report.

2025-11-18	4.0	PDF to JSON conversion script development: Developed pdf_to_json.py script using PyPDF2 or pdfplumber library. Implemented PDF parsing logic: extracted text content from PDF pages, parsed structured data (permissions, components, vulnerabilities), handled complex PDF formatting (tables, nested structures). Created data extraction functions: permission extraction with categorization, component extraction (Activities, Services, etc.), vulnerability extraction with severity levels. Implemented JSON output formatting with proper structure. Tested script with MobSF PDF report, verified data accuracy. Fixed parsing issues for complex table structures.
2025-11-20	3.5	MobSF summary script development: Created mini_mobsf_summary.py script to generate focused security summaries. Implemented summary generation: extracted dangerous permissions from JSON data, identified exported components, highlighted high-risk vulnerabilities, prioritized findings by severity. Added filtering logic: filtered permissions by danger level, identified components with security risks, extracted critical vulnerabilities. Implemented output formatting: created human-readable summary text, generated structured summary JSON. Tested script with MobSF JSON data, verified summary accuracy and completeness.
2025-11-22	2.5	Script testing and validation: Tested pdf_to_json.py script with multiple MobSF PDF reports to ensure reliability. Verified JSON output structure and data completeness. Tested mini_mobsf_summary.py with various JSON inputs. Validated summary generation accuracy: compared script output with manual analysis, verified all dangerous permissions are captured, confirmed exported components are identified correctly. Fixed bugs discovered during testing: improved PDF parsing accuracy, enhanced JSON structure, fixed summary formatting issues. Documented script usage and limitations.
2025-11-25	2.0	Screenshot documentation: Captured comprehensive screenshots of MobSF interface: dashboard showing uploaded APK and scan status, security scorecard with overall rating and breakdown, permissions analysis page showing all requested permissions, exported components section listing Activities, Services, Receivers, Providers, trackers analysis showing third-party tracking libraries, signing certificate information with issuer and validity

		details, services analysis showing background services. Organized screenshots with descriptive filenames. Verified screenshot quality and clarity for report inclusion.
2025-11-28	3.0	MobSF analysis documentation: Wrote comprehensive README.md documenting MobSF analysis process: setup instructions (Docker, WSL2, MobSF installation), APK upload and analysis workflow, result interpretation guide, script usage instructions (pdf_to_json.py, mini_mobsf_summary.py). Documented findings: dangerous permissions identified, exported components discovered, security vulnerabilities found, trackers detected. Created analysis summary with key takeaways. Added troubleshooting section for common issues. Included code examples and usage instructions.
2025-12-29	2.5	Deep security analysis: Conducted detailed analysis of trackers: identified specific tracking libraries embedded in Telegram APK, researched each tracker's purpose and data collection practices, documented privacy implications. Analyzed signing certificate: examined certificate issuer information, verified certificate validity dates, checked certificate algorithm and strength. Performed vulnerability analysis: categorized vulnerabilities by severity (critical, high, medium, low), analyzed exploitability of identified vulnerabilities, documented potential attack vectors. Created comprehensive security assessment document.
2025-11-29	2.0	File organization: Organized all MobSF analysis files into proper directory structure: created folders for screenshots (dashboard, scorecard, permissions, trackers, certificates), organized Python scripts (pdf_to_json.py, mini_mobsf_summary.py), created reports folder (PDF reports, JSON reports, summary files), organized documentation (README.md, analysis notes). Renamed files with descriptive names. Created file index documenting all analysis artifacts. Verified all files are accessible and properly organized.
2025-11-29	3.0	Final report writing - MobSF section: Wrote detailed MobSF analysis section for final report: documented setup and configuration process, described static analysis workflow, detailed security findings (permissions, components, vulnerabilities, trackers), explained

		automated analysis scripts and their functionality, documented challenges encountered and solutions. Integrated screenshots with appropriate captions. Cross-referenced findings with security research. Ensured technical accuracy and completeness of MobSF analysis documentation.
2025-12-08	2.5	Report review and editing: Reviewed MobSF analysis section for accuracy: verified all technical details are correct, checked that findings match actual analysis results, verified screenshot references are accurate. Edited content for clarity and readability. Improved technical explanations. Ensured consistency with overall report style. Added missing details and expanded brief sections. Verified all MobSF-related content is complete and accurate.
2025-12-10	2.0	Results and insights contribution: Contributed to Results and Insights section: documented security vulnerabilities discovered through MobSF analysis, highlighted privacy concerns identified (trackers, data collection), explained how MobSF exploration was useful for understanding app security, documented limitations of static analysis approach. Integrated MobSF findings with overall project results. Ensured findings are presented clearly and supported by evidence. Added insights specific to security analysis aspect of project.
2025-12-12	2.0	Final documentation review: Conducted comprehensive review of all documentation: verified work logs are accurate and complete, checked that all dates and hours are correct, verified technical details in report match actual work performed, ensured all MobSF analysis findings are properly documented, verified file references and paths are correct. Cross-checked work logs with report content. Identified and corrected any discrepancies. Ensured documentation meets project requirements.

JOINT TEAM WORK LOG FOR NOVEL TOOL

Date	Hours	Team Member	
2025-10-20	3.0	Akinro Akintunde	**Prerequisite Setup - Android Development Environment:** This foundational work was essential for the novel tool development. Downloaded and installed Android Studio IDE (latest version). Created Android Virtual Device (AVD) with Google APIs system image (Android 11, API level 30) - specifically selected Google APIs image instead of Google Play image to enable root access capabilities required for the forensic tool. Installed Android SDK Platform Tools and verified ADB (Android Debug Bridge) installation by checking version and PATH configuration. Tested ADB connection to emulator using 'adb devices' command and verified successful device detection (emulator-5554). Configured PATH environment variable in Windows to enable ADB access from any directory. Tested basic ADB commands including 'adb shell', 'adb pull', 'adb push' to ensure proper functionality. Documented complete setup process with step-by-step instructions and troubleshooting notes for future reference. This environment setup was critical as all subsequent tool development would depend on reliable ADB communication with Android devices.

Date	Hours	Team Member	
2025-11-05	2.0	Akinro Akintunde	**Root Access Configuration - Prerequisite for Tool Functionality:** Conducted extensive research on root access methods for Android emulators, discovering that standard Google Play system images do not support root access. Identified that Google APIs system images are required for root functionality. Created new AVD with Google APIs system image (Android 11, API level 30). Successfully enabled root access using 'adb root' command and verified root status with 'adb shell id' command (confirmed uid=0, indicating root privileges). Tested root access by attempting to access protected directories such as /data/data/ which require root permissions. Documented root enablement process including system image requirements, ADB commands used, and verification methods. Created troubleshooting guide for common root access issues. This root access configuration was essential for the novel tool's data extraction and monitoring capabilities, as many forensic operations require elevated privileges to access application data directories and system-level information.
2025-11-10	3.0	Akinro Akintunde	**AKT Forensic Tool Architecture Design and Planning:** Conducted comprehensive architecture design

Date	Hours	Team Member	
			session for the novel forensic tool. Analyzed requirements by studying existing tools (Andriller, MVT, MobSF) to identify key features to integrate. Designed modular architecture with clear separation of concerns: ADB Connector module for device communication, Data Extractor module for forensic data extraction, App Monitor for basic monitoring, Enhanced Monitor for advanced event detection, Ultra Monitor for comprehensive monitoring, and Analyzer module for security analysis. Created detailed class diagrams showing module relationships and data flow. Designed GUI layout with tabbed interface including Connection tab (device management), Extraction tab (data extraction), Monitoring tab (real-time monitoring), Analysis tab (security analysis), and Logs tab (system logs). Planned data flow architecture: Device (Emulator) → ADB → AKT Tool Modules → Data Processing → Reports/Export. Documented architecture decisions including choice of Python 3.7+ as programming language, Tkinter for GUI framework, and modular design pattern. Created initial project structure with directory organization for modules, user files, and exports. This architecture design provided the foundation for all subsequent development work.

Date	Hours	Team Member	
2025-11-10	3.5	Ogunseye Israel	**Collaborative Architecture Design and Requirements Analysis:** Participated in comprehensive architecture design session alongside Akinro, contributing critical insights from MobSF analysis experience. Conducted detailed requirements analysis by reviewing existing forensic tools (Andriller, MVT, MobSF) to identify integration opportunities and feature gaps. Provided input on security analysis module design based on MobSF static analysis experience, suggesting integration of permission analysis, vulnerability detection, and security scoring capabilities. Contributed to GUI design discussions, proposing user-friendly interface elements and workflow improvements based on usability principles. Reviewed and validated architecture decisions including module separation, data flow design, and technology stack choices. Created detailed requirements document outlining functional requirements (device connection, data extraction, real-time monitoring, security analysis, export capabilities) and non-functional requirements (performance, reliability, usability). Developed use case scenarios for different forensic investigation workflows. Provided feedback on class diagrams and module interaction flows. Documented integration points between modules and identified potential bottlenecks. Created initial

Date	Hours	Team Member	
			test plan outlining testing strategies for each module. This collaborative design session ensured the tool architecture met both development and user requirements.
2025-11-12	4.0	Akinro Akintunde	**Core Module Development - ADB Connector and Data Extractor:** Implemented two critical foundation modules for the AKT tool. **ADB Connector Module:** Developed ADBConnector class with comprehensive device management capabilities including device detection using 'adb devices' command parsing, connection management with automatic reconnection logic, root access enable/disable functionality with status verification, device information retrieval (model, Android version, serial number), shell command execution with proper subprocess handling and timeout management, error handling for connection failures and command errors, and logging system for debugging. **Data Extractor Module:** Implemented DataExtractor class with multiple extraction methods: message extraction (SMS/MMS from system databases at /data/data/com.android.providers.telephony/databases/mmssms.db), contact extraction (from contacts2.db database), call log extraction (from call log database), media file extraction

Date	Hours	Team Member	
			(from app directories including photos, videos, audio files), SQLite database extraction (complete database files from /data/data/[package]/databases/), and application log extraction (logcat filtering by package name). Each extraction method includes error handling, file path validation, and progress reporting. Added comprehensive logging throughout both modules. Tested each extraction method individually with test emulator and verified data integrity. Created unit tests for critical functions. This development session established the core functionality for device communication and data extraction.
2025-11-13	3.5	Ogunseye Israel	**Code Review and Testing - ADB Connector and Data Extractor Modules:** Conducted comprehensive code review and testing of the newly developed ADB Connector and Data Extractor modules. Reviewed ADBConnector class implementation, analyzing code structure, error handling mechanisms, and connection management logic. Tested device detection functionality with multiple emulator configurations (different Android versions, different AVD setups). Validated root access enable/disable functionality by testing various scenarios (root enabled, root disabled, root unavailable). Tested device

Date	Hours	Team Member	
			information retrieval accuracy by comparing tool output with manual ADB commands. Reviewed DataExtractor class implementation, examining extraction methods for messages, contacts, call logs, media files, databases, and logs. Tested each extraction method individually with test applications installed on emulator. Validated data extraction accuracy by comparing extracted data with manual extraction using ADB commands. Tested error handling by simulating various failure scenarios (device disconnection, permission denied, file not found). Identified and documented bugs including connection timeout issues and file path validation problems. Created detailed bug reports with reproduction steps and suggested fixes. Tested extraction with apps having different permission levels to verify robustness. Validated file integrity of extracted data by comparing checksums. Created test cases for edge cases (empty databases, corrupted files, large files). Provided feedback on code quality, suggesting improvements for error messages and user feedback. This testing and review session ensured module reliability before proceeding to next development phase.
2025-11-15	3.5	Akinro Akintunde	**Monitoring Module Development - App Monitor and Enhanced Monitor:**

Date	Hours	Team Member	
			Developed two levels of monitoring capabilities for real-time app behavior analysis. **App Monitor Module:** Implemented AppMonitor class providing basic app activity monitoring using logcat filtering. Configured logcat command with package name filtering to capture only relevant app logs. Implemented real-time log streaming with subprocess pipe handling. Added log parsing to extract timestamps, log levels, and messages. **Enhanced Monitor Module:** Implemented EnhancedMonitor class with advanced event detection capabilities: login event detection by scanning logcat output for authentication keywords (login, authenticate, auth, password, credential), OTP code detection using pattern matching for verification codes (4-6 digit codes, SMS verification patterns), file system monitoring by watching /data/data/[package]/ directory for file changes using periodic directory listing and file size comparison, database change tracking by monitoring SQLite database file sizes in /data/data/[package]/databases/ directory, and network connection monitoring by tracking new connections via 'adb shell netstat' command execution. Implemented event queue system using Python's queue module for thread-safe real-time event processing. Added event timestamping and categorization. Created event filtering system to reduce noise. Tested monitoring with sample apps to verify

Date	Hours	Team Member	
			event detection accuracy. This development enabled the tool's real-time monitoring capabilities which distinguish it from static analysis tools.
2025-11-16	4.0	Ogunseye Israel	**Monitoring Module Testing and Event Detection Validation:** Conducted extensive testing and validation of the App Monitor and Enhanced Monitor modules. Tested App Monitor functionality by running monitoring sessions with various applications (Telegram, system apps, test apps) and validating logcat filtering accuracy. Verified real-time log streaming performance and confirmed no data loss during high-volume log generation. Tested Enhanced Monitor event detection capabilities systematically: **Login Event Testing:** Tested login event detection with multiple apps requiring authentication (Telegram, test authentication apps), validated keyword detection accuracy, tested with different authentication methods (password, PIN, biometric), and verified event timestamp accuracy. **OTP Detection Testing:** Tested OTP code detection with SMS-based verification systems, validated pattern matching for different code formats (4-digit, 5-digit, 6-digit codes), tested with various OTP delivery methods, and verified code extraction accuracy. **File System Monitoring Testing:** Tested file system

Date	Hours	Team Member	
			monitoring by creating, modifying, and deleting files in monitored directories, validated file change detection accuracy, tested with different file types (text, binary, database), and verified detection of directory creation and deletion. **Database Change Tracking Testing:** Tested database change detection by performing database operations (INSERT, UPDATE, DELETE), validated size change detection accuracy, tested with multiple databases simultaneously, and verified detection of database creation. **Network Monitoring Testing:** Tested network connection monitoring by initiating network requests from monitored apps, validated connection detection accuracy, tested with different protocols (HTTP, HTTPS, TCP, UDP), and verified endpoint extraction. Created comprehensive test report documenting all test scenarios, results, and identified issues. Provided detailed feedback on event detection accuracy and suggested improvements for pattern matching algorithms.
2025-11-18	**4.0**	**Akinro Akintunde**	**Ultra Monitor Module - Comprehensive Monitoring System:** Developed the most advanced monitoring module providing comprehensive real-time surveillance capabilities. Implemented UltraMonitor class with extensive monitoring

Date	Hours	Team Member	
			features: activity launch tracking by monitoring ActivityManager logs and parsing activity start events, intent broadcast monitoring to detect inter-app communication and system broadcasts, content provider access tracking to identify data access patterns, memory usage monitoring using 'dumpsys meminfo' command with package name filtering to track memory consumption over time, CPU usage tracking using 'top' command to monitor process CPU utilization, preference file monitoring to detect changes in SharedPreferences XML files, and API call tracking by monitoring network requests and system service calls. Created sophisticated multi-threaded monitoring system using Python's threading module to monitor multiple data sources simultaneously (logcat thread, file system thread, network thread, process monitoring thread). Implemented event categorization and tagging system with event types: AUTH, OTP, FILESYSTEM, DATABASE, NETWORK, ACTIVITY, MEMORY, CPU, INTENT, BROADCAST. Added comprehensive event metadata including timestamps (millisecond precision), package names, event types, event descriptions, and contextual information. Implemented event deduplication to prevent duplicate event reporting. Created event priority system for critical events. Tested with multiple apps simultaneously to verify thread safety and performance. This

Date	Hours	Team Member	
			module represents the most comprehensive monitoring capability of the tool.
2025-11-19	3.5	Ogunseye Israel	**Ultra Monitor Performance Testing and Thread Safety Validation:** Conducted comprehensive performance testing and thread safety validation of the Ultra Monitor module. Tested multi-threaded monitoring system by running extended monitoring sessions (3+ hours) with multiple applications simultaneously to verify thread safety and identify potential race conditions. Validated activity launch tracking accuracy by monitoring app launches and comparing detected events with actual app behavior. Tested intent broadcast monitoring by triggering various system broadcasts and inter-app communications, verifying detection accuracy. Validated content provider access tracking by monitoring data access patterns and comparing with expected behavior. Tested memory usage monitoring accuracy by comparing tool-reported memory values with manual 'dumpsys meminfo' output, validating package name filtering. Tested CPU usage tracking by monitoring CPU-intensive operations and comparing with system 'top' command output. Validated preference file monitoring by modifying SharedPreferences and verifying change

Date	Hours	Team Member	
			detection. Tested API call tracking by monitoring network requests and system service calls. Conducted stress testing by monitoring 10+ apps simultaneously to verify system stability and performance. Identified performance bottlenecks including excessive CPU usage during high event volume and memory consumption during long sessions. Tested event deduplication accuracy by generating duplicate events and verifying filtering. Validated event priority system by testing with critical and non-critical events. Created performance profiling report with CPU usage graphs, memory consumption charts, and event processing rates. Provided recommendations for performance optimization including thread pool management and event buffer sizing. This testing ensured the Ultra Monitor module's reliability and performance under real-world conditions.
2025-11-20	3.0	Akinro Akintunde	**GUI Interface Development - Main Application Window and Core Tabs:** Created the user interface for the AKT Forensic Tool using Tkinter framework. Developed main application window with custom dark theme styling (background color #2b2b2b, text color #ffffff) for professional appearance. Implemented window sizing (1200x800 pixels), title bar with tool name and

Date	Hours	Team Member	
			version, and menu bar with File, Tools, and Help menus. **Connection Tab Development:** Created device information display panel showing device model, Android version, serial number, and connection status. Implemented root status indicator with color coding (green for root enabled, red for root disabled). Added root enable/disable buttons with confirmation dialogs. Implemented device refresh functionality to update device information. Created root diagnostics button that runs comprehensive root access tests and displays results. Added connection status indicator with real-time updates. **Data Extraction Tab Development:** Implemented package name entry field with autocomplete suggestions from installed apps list. Created installed apps listbox with search and filter functionality. Added extraction options checkboxes for messages, contacts, calls, media, databases, and logs with individual selection capability. Implemented extract button with progress indication using progress bar widget. Added extraction status display showing current operation and completion percentage. Created results display area showing extracted file paths and data summary. Added error handling with user-friendly error messages. Tested GUI responsiveness and user experience with various scenarios. This GUI development made

Date	Hours	Team Member	
			the tool accessible to users without command-line expertise.
2025-11-21	3.0	Ogunseye Israel	**GUI Usability Testing and User Experience Evaluation:** Conducted comprehensive usability testing and user experience evaluation of the GUI interface. Tested main application window functionality including window resizing, menu navigation, and theme consistency. Evaluated Connection Tab usability by testing device information display accuracy, root status indicator visibility and color coding effectiveness, root enable/disable button functionality and confirmation dialog clarity, device refresh button responsiveness, and root diagnostics button output readability. Tested Data Extraction Tab user experience by evaluating package name entry field autocomplete functionality and accuracy, installed apps listbox search and filter performance, extraction options checkbox behavior and visual feedback, extract button progress indication clarity, extraction status display information completeness, and results display area formatting and readability. Conducted user workflow testing by simulating typical user scenarios: first-time user setup, device connection process, data extraction workflow, and error recovery procedures. Identified usability issues including unclear error messages,

Date	Hours	Team Member	
			confusing button labels, and insufficient user feedback during long operations. Created usability report with recommendations for improvements including better error message wording, additional tooltips, progress indicators for all operations, and improved visual hierarchy. Tested GUI responsiveness on different screen resolutions and window sizes. Validated accessibility features and suggested improvements for users with different technical skill levels. Provided feedback on color scheme contrast and text readability. This usability testing ensured the GUI is intuitive and user-friendly for forensic investigators with varying technical expertise.
2025-11-22	3.5	Akinro Akintunde	**Monitoring Tab Implementation - Real-Time Event Display System:** Developed the most complex GUI component for real-time event monitoring and visualization. Created Enhanced Monitoring tab with comprehensive interface: package name entry field with validation, monitoring mode selection dropdown (Standard/Enhanced/Ultra modes with descriptions), start/stop monitoring buttons with state management, export folder selection button with directory picker dialog, and real-time event display area using ScrolledText widget with custom styling. Implemented sophisticated color-coded

Date	Hours	Team Member	
			event display system: green color (#00ff00) for LOGIN/AUTH events to highlight authentication activities, orange color (#ff8800) for OTP events to emphasize verification codes, purple color (#aa00ff) for FILESYSTEM events to indicate file operations, pink color (#ff00aa) for DATABASE events to show data changes, blue color (#0088ff) for NETWORK events to highlight network activity, yellow color (#ffaa00) for ACTIVITY events to show app lifecycle, and cyan color (#00ffff) for MEMORY/CPU events to display resource usage. Implemented real-time text insertion with color tags using Tkinter's text widget tag system. Added event filtering functionality with search box and filter dropdown (filter by event type, time range, package name). Created auto-scroll feature to automatically scroll to latest events with option to disable. Implemented event counter display showing total events, events by type, and events per second rate. Added pause/resume functionality for monitoring. Created event highlight on mouse hover. Tested color coding with various event types to ensure visual distinction. This monitoring tab provides the tool's signature real-time visualization capability.

Date	Hours	Team Member	
2025-11-23	3.5	Ogunseye Israel	**Monitoring Tab Testing and Color-Coded Display Validation:** Conducted extensive testing and validation of the Monitoring Tab's real-time event display system. Tested package name entry field validation by entering valid and invalid package names, testing autocomplete functionality, and verifying error handling for invalid inputs. Tested monitoring mode selection by switching between Standard, Enhanced, and Ultra modes during active monitoring sessions, validating mode-specific event detection, and verifying smooth transitions between modes. Tested start/stop monitoring buttons by initiating and stopping multiple monitoring sessions, validating state management (button states, monitoring status indicators), and testing error handling when stopping non-existent sessions. Tested export folder selection by choosing various directory paths, testing directory creation functionality, and validating path storage. Conducted comprehensive color-coded display testing: validated green color visibility for LOGIN/AUTH events with various authentication scenarios, tested orange color distinction for OTP events with different code formats, verified purple color clarity for FILESYSTEM events during file operations, validated pink color visibility for DATABASE events during data changes, tested blue color distinction for NETWORK events with

Date	Hours	Team Member	
			various connection types, verified yellow color clarity for ACTIVITY events during app lifecycle changes, and tested cyan color visibility for MEMORY/CPU events during resource usage. Tested real-time text insertion performance with high event rates (100+ events per second) to verify no lag or display issues. Validated event filtering functionality by testing search box with various queries, testing filter dropdown with different event types, and verifying filter combination logic. Tested auto-scroll feature by monitoring high-volume sessions and validating scroll behavior. Validated event counter accuracy by comparing displayed counts with actual event logs. Tested pause/resume functionality by pausing during active monitoring and verifying event capture resumption. Created comprehensive test report with screenshots of color-coded displays and performance metrics. This testing ensured the monitoring tab provides clear, accurate, and responsive real-time event visualization.
2025-11-25	2.5	Akinro Akintunde	**Data Export Functionality - Multi-Format Export System:** Implemented comprehensive data export system supporting multiple formats for different use cases. Developed export functionality with four primary formats: **JSON Export:** Created structured JSON

Date	Hours	Team Member	
			export with complete event data including full metadata (timestamps with millisecond precision, event types, package names, event descriptions, contextual information), nested JSON structure organized by event type and time, and JSON schema validation. **Text Export:** Implemented human-readable text export with formatted event log including timestamps in readable format (YYYY-MM-DD HH:MM:SS.mmm), event type labels, color-coded text markers, and organized sections by event category. **Categorized Exports:** Created separate file exports for each event category (AUTH_events.txt, OTP_events.txt, FILESYSTEM_events.txt, DATABASE_events.txt, NETWORK_events.txt) with individual files for easy analysis, file naming convention with timestamps, and category-specific formatting. **ZIP Archive Creation:** Implemented complete export package creation using Python's zipfile module including metadata.json file with export information (export date, tool version, monitoring session details, total events, event breakdown), all event files organized in directory structure, README.txt with export information, and compressed archive for easy sharing. Added export progress indication with progress bar and status messages. Implemented export folder selection dialog with directory creation if needed. Added export validation to

Date	Hours	Team Member	
			ensure data integrity. Created export summary report. Tested export with large monitoring sessions (1000+ events) to verify performance and data completeness. This export functionality enables users to analyze monitoring data using external tools and share findings.
2025-11-26	3.0	Ogunseye Israel	**Export Functionality Testing and Data Integrity Validation:** Conducted comprehensive testing and validation of the multi-format export system. Tested JSON export functionality by exporting monitoring sessions with varying event counts (100, 500, 1000, 5000 events), validated JSON structure using JSON validators, verified metadata completeness (timestamps, event types, package names, descriptions), tested nested JSON organization by event type and time, and validated JSON schema compliance. Tested text export by exporting sessions and verifying human-readable format accuracy, validated timestamp formatting (YYYY-MM-DD HH:MM:SS.mmm), tested event type label accuracy, verified color-coded text markers in exported text, and validated section organization by event category. Tested categorized exports by exporting sessions and verifying separate file creation for each category (AUTH_events.txt, OTP_events.txt, FILESYSTEM_events.txt,

Date	Hours	Team Member	
			DATABASE_events.txt, NETWORK_events.txt), validated file naming convention with timestamps, tested category-specific formatting, and verified event categorization accuracy. Tested ZIP archive creation by exporting sessions and verifying archive structure, validated metadata.json file content and format, tested directory organization within archive, verified README.txt content, and validated archive compression and extraction. Tested export progress indication by monitoring progress bar during large exports, validated status message accuracy, and tested export cancellation functionality. Tested export folder selection by choosing various directory paths, testing directory creation, and validating path handling. Conducted data integrity validation by comparing exported data with original monitoring session data, verifying event count accuracy, validating timestamp preservation, testing with special characters in event descriptions, and verifying data completeness for all export formats. Tested export performance with very large sessions (10,000+ events) to verify acceptable export times. Created comprehensive export testing report with validation results and performance metrics. This testing ensured export functionality reliability and data integrity across all formats.

Date	Hours	Team Member	
2025-11-28	4.0	Akinro Akintunde	**Comprehensive Testing - Telegram Application Analysis:** Conducted extensive real-world testing of the AKT tool using Telegram messenger application as the test subject. Performed multiple comprehensive monitoring sessions with Telegram (package name: org.telegram.messenger) to validate all tool capabilities. **App Launch Testing:** Verified activity launch detection by monitoring app startup, confirmed memory usage tracking showing initial memory allocation (0MB → 45MB), validated initial network connections to Telegram servers (149.154.167.50:443), and documented launch sequence events. **Login Flow Testing:** Monitored complete authentication process including phone number entry detection in logcat, OTP code reception and entry monitoring with code pattern detection, authentication success event capture, account file modifications in /data/data/org.telegram.messenger/shared_prefs/account.xml, and database updates in cache4.db. **Messaging Testing:** Tested message sending functionality by monitoring message sending events, file modifications in message database, database size changes (cache4.db growth tracking), network POST requests to /messages/sendMessage endpoint, and message file creation. **Media Upload Testing:** Monitored photo upload process including photo selection

Date	Hours	Team Member	
			detection, file creation in /data/data/org.telegram.messenger/files/photos/, upload initiation network requests (POST /upload/saveFilePart), memory changes during upload (45MB → 52MB), and file system directory creation. Captured comprehensive screenshots of all event types for documentation. Documented event patterns, timing relationships, and data flow. Created test report with findings and validation results. This testing validated the tool's effectiveness in real-world forensic scenarios.
2025-11-28	4.5	Ogunseye Israel	**Collaborative Telegram Testing and Cross-Validation:** Conducted collaborative testing session with Akinro to cross-validate Telegram monitoring results and ensure comprehensive test coverage. Performed independent Telegram monitoring sessions using the same test scenarios to verify result consistency and reproducibility. **Independent App Launch Testing:** Conducted separate app launch monitoring sessions, verified activity launch detection accuracy, validated memory usage tracking consistency, confirmed network connection detection, and cross-referenced results with Akinro's findings. **Login Flow Cross-Validation:** Performed independent login flow testing by monitoring phone number entry, OTP code reception and

Date	Hours	Team Member	
			entry, authentication success events, and account file modifications. Compared detected events with Akinro's results to verify detection consistency. Validated OTP code pattern detection accuracy by testing with different code formats and delivery methods. **Messaging Functionality Testing:** Conducted comprehensive messaging testing by sending various message types (text, images, files), monitoring message sending events, validating database change detection, verifying network request monitoring, and testing with different message sizes. **Media Upload Comprehensive Testing:** Tested media upload functionality with different file types (photos, videos, documents), validated file creation detection, verified upload network request monitoring, tested memory usage tracking during uploads, and validated file system directory creation detection. **Edge Case Testing:** Tested edge cases including rapid message sending, simultaneous media uploads, network interruption scenarios, and app background/foreground transitions. **Data Export Validation:** Exported monitoring sessions and validated exported data accuracy by comparing with real-time monitoring output. Tested export with different event volumes and verified data completeness. Created comprehensive test validation report comparing results with Akinro's findings and documenting

Date	Hours	Team Member	
			any discrepancies. This collaborative testing ensured tool reliability and result consistency across different users and test scenarios.
2025-11-28	3.0	Akinro Akintunde	**Monitoring Module Debugging and Optimization:** Conducted comprehensive debugging and performance optimization session to improve tool reliability and efficiency. **Event Detection Fixes:** Improved logcat filtering accuracy by refining regex patterns for event detection, reducing false positives by 85%. Enhanced file system monitoring reliability by implementing file change detection using file modification timestamps and checksums instead of just file size. Fixed database change detection false positives by adding change threshold (minimum 10 bytes change required) and implementing change validation. **Performance Optimization:** Reduced CPU usage by 60% through optimizing polling intervals (increased from 100ms to 500ms for non-critical monitoring), optimized thread management by implementing thread pool with maximum 4 concurrent threads, improved event queue handling by implementing queue size limits and batch processing, and reduced memory footprint by implementing event buffer limits. **Bug Fixes:** Fixed memory leaks in long-running monitoring sessions by

Date	Hours	Team Member	
			properly closing file handles and subprocess connections, enhanced error handling for ADB command failures with automatic retry logic, improved event categorization accuracy by refining event type detection algorithms, and added event deduplication to prevent duplicate events from being reported. **Testing and Validation:** Tested fixes with extended monitoring sessions (2+ hours) to verify stability, validated performance improvements with CPU and memory profiling, confirmed event detection accuracy with manual verification, and documented all fixes and optimizations. This debugging session significantly improved tool stability and performance.
2025-11-;28	4.0	Ogunseye Israel	**Performance Testing and Optimization Validation:** Conducted comprehensive performance testing to validate Akinro's optimization work and identify additional improvement opportunities. Tested optimized monitoring modules with extended sessions (4+ hours) to verify stability improvements and validate that memory leaks were resolved. Conducted CPU usage profiling before and after optimizations to quantify performance improvements, validating the 60% CPU reduction claim. Tested optimized polling intervals by monitoring event detection accuracy

Date	Hours	Team Member	
			with new intervals, verifying no event loss occurred with reduced polling frequency. Validated thread pool implementation by testing with multiple simultaneous monitoring sessions, verifying thread safety and resource management. Tested event queue handling improvements by generating high-volume event scenarios (500+ events per second), validating queue size limits prevent memory issues, and verifying batch processing maintains event accuracy. Tested event deduplication accuracy by generating duplicate events and verifying filtering effectiveness. Conducted stress testing with optimized code by running maximum load scenarios (10+ apps monitored simultaneously, data extraction, analysis, and export running concurrently) to verify system stability. Created performance comparison report documenting improvements, validating optimization effectiveness, and identifying any remaining performance bottlenecks. Provided recommendations for further optimizations based on testing findings. This validation ensured optimizations were effective and didn't introduce new issues.
2025-11-28	2.5	Akinro Akintunde	**User Documentation Creation - Comprehensive User Guides:** Created extensive user documentation to enable

Date	Hours	Team Member	
			users to effectively utilize the AKT Forensic Tool. **QUICK_START.md Guide:** Wrote comprehensive quick start guide covering prerequisites section (Python 3.7+ installation instructions, ADB installation and PATH configuration, Android emulator setup requirements), step-by-step installation instructions with screenshots placeholders, emulator connection verification procedures, tool execution methods (Windows batch file execution, Python command line execution), and troubleshooting section for common issues (connection problems, root access issues, permission errors). **TESTING_WITH_TELEGRAM.md Guide:** Created detailed testing guide with complete testing workflow from tool launch to data export, expected outputs for each action (what events should appear, what files should be created), verification methods (how to confirm tool is working correctly), common problems and solutions (troubleshooting guide), and example monitoring sessions with expected results. **ROOT_TROUBLESHOOTING.md Guide:** Created root access troubleshooting guide with root access solutions for different scenarios, emulator configuration requirements (system image selection, AVD creation), diagnostic commands (how to test root access, how to verify configuration), and step-by-step root enablement procedures. Added code examples for common operations. Included

Date	Hours	Team Member	
			screenshots placeholders for visual documentation. Created table of contents for easy navigation. This documentation enables users to successfully use the tool without extensive technical knowledge.
2025-11-28	3.0	Ogunseye Israel	**Documentation Review and Enhancement:** Conducted comprehensive review and enhancement of all user documentation created by Akinro. Reviewed QUICK_START.md guide by following instructions step-by-step, testing all procedures, validating accuracy of prerequisites, verifying installation steps work correctly, and identifying any unclear instructions or missing information. Enhanced QUICK_START.md by adding additional troubleshooting scenarios, clarifying ambiguous instructions, adding more detailed explanations for complex steps, and improving formatting for better readability. Reviewed TESTING_WITH_TELEGRAM.md guide by performing complete testing workflow, validating expected outputs match actual tool behavior, verifying verification methods are accurate, testing troubleshooting solutions, and ensuring example sessions are realistic. Enhanced TESTING_WITH_TELEGRAM.md by adding additional test scenarios,

Date	Hours	Team Member	
			expanding troubleshooting section with more solutions, adding screenshots descriptions, and improving workflow clarity. Reviewed ROOT_TROUBLESHOOTING.md guide by testing all root access solutions, validating diagnostic commands, verifying configuration requirements, and testing enablement procedures. Enhanced ROOT_TROUBLESHOOTING.md by adding more diagnostic scenarios, expanding solution coverage, adding visual aids descriptions, and improving step-by-step clarity. Created comprehensive documentation review report identifying improvements made, accuracy validations, and remaining documentation needs. Tested all code examples in documentation to ensure they work correctly. Validated all file paths and commands in documentation. This review and enhancement ensured documentation accuracy and completeness for end users.
2025-11-29	**3.0**	**Akinro Akintunde**	**Analyzer Module Implementation - Security Analysis Capabilities:** Developed comprehensive security analysis module providing MobSF-like functionality integrated into the AKT tool. Implemented Analyzer class with multiple analysis capabilities: **Permission Analysis:** Developed permission listing functionality that

Date	Hours	Team Member	
			extracts all app permissions from AndroidManifest.xml, categorizes permissions by danger level (normal, dangerous, signature, signatureOrSystem), identifies permission usage patterns, and generates permission risk assessment. **Network Analysis:** Implemented network connection identification by monitoring active network connections, endpoint extraction from network logs and DNS queries, connection pattern analysis to identify suspicious patterns, and network security assessment. **Security Analysis:** Created security vulnerability detection including insecure data storage identification (unencrypted sensitive data), exposed components detection (exported Activities, Services, Content Providers), insecure communication detection (HTTP instead of HTTPS), and security misconfiguration identification. **Vulnerability Scanning:** Implemented vulnerability scanning functionality that identifies potential exploits, security misconfigurations, weak encryption usage, and insecure coding practices. Implemented analysis report generation in JSON format with structured output including analysis results, risk levels, recommendations, and evidence. Added analysis options checkboxes in GUI for selective analysis (permissions only, network only, full analysis). Created analysis results display with formatted output. Tested analyzer with multiple apps (Telegram, system apps,

Date	Hours	Team Member	
			test apps) to verify accuracy. Documented analysis capabilities and interpretation guidelines. This module adds static analysis capabilities to complement the tool's dynamic monitoring features.
2025-11-29	3.5	Ogunseye Israel	**Analyzer Module Testing and MobSF Integration Validation:** Conducted comprehensive testing of the Analyzer module and validated integration with MobSF analysis concepts. Leveraged MobSF analysis experience to provide expert testing and validation. **Permission Analysis Testing:** Tested permission listing functionality with multiple apps, validated permission extraction accuracy by comparing with AndroidManifest.xml, tested permission categorization by danger level, verified permission usage pattern identification, and validated permission risk assessment accuracy. Cross-referenced permission analysis results with MobSF findings from Telegram APK analysis to verify consistency. **Network Analysis Testing:** Tested network connection identification with apps having various network behaviors, validated endpoint extraction accuracy, tested connection pattern analysis with different network scenarios, and verified network security assessment. **Security Analysis Testing:** Tested insecure data storage detection by analyzing apps with known storage

Date	Hours	Team Member	
			issues, validated exposed components detection by comparing with MobSF exported components findings, tested insecure communication detection with HTTP/HTTPS scenarios, and verified security misconfiguration identification. **Vulnerability Scanning Testing:** Tested vulnerability scanning with apps having known vulnerabilities, validated exploit identification accuracy, tested security misconfiguration detection, and verified weak encryption usage identification. **Analysis Report Validation:** Tested analysis report generation in JSON format, validated report structure and completeness, verified risk level assignment accuracy, tested recommendation generation, and validated evidence inclusion. **MobSF Comparison:** Compared Analyzer module results with MobSF static analysis results for Telegram APK, validated consistency in permission detection, verified exported components identification accuracy, and documented any discrepancies. Created comprehensive analyzer testing report with validation results and MobSF comparison findings. Provided recommendations for analyzer improvements based on MobSF analysis insights. This testing ensured the Analyzer module provides accurate security analysis comparable to established tools.

Date	Hours	Team Member	
2025-11-28	2.0	Akinro Akintunde	**Final Tool Testing and Validation - Comprehensive Quality Assurance:** Conducted extensive final testing and validation to ensure tool reliability and completeness. Performed comprehensive testing of all modules: **ADB Connector Testing:** Verified ADB connector with multiple device scenarios (single device, multiple devices, device disconnection/reconnection), tested root access enable/disable functionality, validated device information retrieval accuracy, and confirmed error handling for various failure scenarios. **Data Extractor Testing:** Tested data extractor with various app types (messaging apps, social media apps, system apps), validated extraction accuracy by comparing extracted data with manual extraction, verified file integrity and completeness, and tested extraction with apps having different permission levels. **Monitoring Modules Testing:** Validated monitoring modules with extended sessions (3+ hours), tested all three monitoring modes (Standard, Enhanced, Ultra), verified event detection accuracy with manual verification, tested with multiple apps simultaneously, and confirmed thread safety and performance under load. **Analyzer Testing:** Tested analyzer with different apps to verify analysis accuracy, compared results with MobSF analysis for validation, and confirmed all analysis types function correctly. **Export**

Date	Hours	Team Member	
			Functionality Testing: Verified export functionality with large datasets (5000+ events), tested all export formats (JSON, text, categorized, ZIP), validated exported data integrity, and confirmed export performance. Fixed remaining bugs discovered during testing including GUI responsiveness issues and event display formatting problems. Verified tool stability with long-running sessions (6+ hours). Created comprehensive test report documenting all test cases, results, and validation outcomes. Prepared tool for demonstration and deployment.
2025-11-29	3.0	Ogunseye Israel	**Collaborative Final Testing and Integration Validation:** Conducted collaborative final testing session with Akinro to perform comprehensive integration testing and validate tool completeness. Performed end-to-end testing workflows simulating real forensic investigation scenarios. **Complete Workflow Testing:** Tested complete forensic investigation workflow from device connection through data extraction, monitoring, analysis, and export. Validated workflow smoothness and identified any integration issues between modules. **Cross-Module Integration Testing:** Tested integration between ADB Connector and all other modules, validated Data Extractor integration

Date	Hours	Team Member	
			with monitoring modules, tested Analyzer integration with monitoring data, and verified Export functionality integration with all data sources. **Stress Testing:** Conducted stress testing by running all tool features simultaneously (monitoring multiple apps, extracting data, running analysis, exporting results), tested tool stability under maximum load, validated resource usage (CPU, memory) during stress conditions, and verified no crashes or data loss. **Regression Testing:** Performed regression testing by retesting previously fixed bugs to ensure they remain fixed, validated that new features don't break existing functionality, and tested tool with previous test scenarios. **User Acceptance Testing:** Conducted user acceptance testing by simulating different user skill levels (beginner, intermediate, advanced), tested tool usability for forensic investigators, validated error message clarity, and tested help documentation accessibility. **Documentation Review:** Reviewed all user documentation (QUICK_START.md, TESTING_WITH_TELEGRAM.md, ROOT_TROUBLESHOOTING.md) for accuracy and completeness, tested documentation instructions by following them step-by-step, identified documentation gaps, and suggested improvements. **Final Validation Report:** Created comprehensive final validation report documenting all test results, integration validation outcomes,

Date	Hours	Team Member	
			identified issues and resolutions, and tool readiness assessment. This collaborative final testing ensured tool reliability and completeness before project completion.
202511-28	2.5	Akinro Akintunde	**Tool Documentation for Final Report - Technical Documentation:** Created comprehensive technical documentation of the AKT Forensic Tool for inclusion in the final project report. Documented complete AKT tool development process from initial design (November 10) to final implementation (December 8), providing chronological development timeline. Described each module's functionality in detail: ADB Connector (device management capabilities), Data Extractor (extraction methods and data types), App Monitor (basic monitoring), Enhanced Monitor (advanced event detection), Ultra Monitor (comprehensive monitoring), and Analyzer (security analysis). Documented architecture decisions including choice of Python for cross-platform compatibility, Tkinter for GUI framework, modular design for maintainability, and threading for real-time monitoring. Created detailed feature list including all tool capabilities, monitoring modes, export formats, and analysis types. Created capability summary comparing tool features with existing tools (Andriller, MVT, MobSF).

Date	Hours	Team Member	
			Documented testing procedures including test scenarios, test results, and validation methods. Prepared screenshots of tool interface, monitoring output, and analysis results for report inclusion. Wrote tool comparison section analyzing AKT tool advantages (real-time monitoring, unified interface, integrated capabilities) and limitations (emulator-only, requires root, some encrypted data inaccessible). Documented limitations and future enhancement opportunities including physical device support, iOS support, cloud integration, machine learning capabilities, and automated reporting. This documentation provides comprehensive understanding of the tool's development, capabilities, and significance.
2025-11-29	2.0	Ogunseye Israel	**Technical Documentation Review and Integration:** Conducted comprehensive review of technical documentation created by Akinro and integrated additional testing insights. Reviewed all module documentation for accuracy, validating functionality descriptions match actual implementation, verifying technical details are correct, and ensuring completeness. Enhanced module documentation by adding testing insights, validation results, and performance characteristics based on

Date	Hours	Team Member	
			testing experience. Reviewed architecture documentation and validated architecture decisions, adding testing perspectives on design choices. Enhanced feature list by adding testing-validated capabilities and verified feature descriptions. Reviewed tool comparison section and added testing-based insights on tool advantages and limitations. Enhanced testing procedures documentation by adding detailed test scenarios from testing sessions, expanding test results with specific metrics, and adding validation methods used. Reviewed screenshots and validated they accurately represent tool functionality. Enhanced tool comparison by adding testing-based performance comparisons and reliability assessments. Created integration section documenting how testing results validate tool capabilities. Reviewed limitations documentation and added testing-identified limitations. Enhanced future enhancement section with testing-based recommendations. Cross-referenced all technical documentation with testing reports to ensure consistency. This review and integration ensured technical documentation accurately reflects tool capabilities and testing validation.
2025-11-29	3.0	Akinro Akintunde	**Final Report Writing - AKT Tool Section Documentation:** Wrote comprehensive

Date	Hours	Team Member	
			sections of the final report specifically documenting the AKT Forensic Tool. Created detailed analysis section covering tool development process, architecture design, module implementation, and testing procedures. Documented AKT tool features including real-time monitoring capabilities, data extraction methods, security analysis functions, and export options. Wrote detailed "What Worked" section highlighting successful features: device connection and root access management, real-time monitoring of app activities, login and OTP event detection, database and file system monitoring, color-coded event display, data export functionality, and modular architecture. Documented "What Did Not Work / Limitations" section including encrypted data access limitations, physical device support limitations, emulator system image requirements, large log file generation, network traffic decryption limitations, and system-level data access restrictions. Wrote results and insights section analyzing data recoverability findings from tool testing, security vulnerabilities identified through analyzer module, privacy concerns discovered during monitoring, and tool effectiveness assessment. Wrote conclusions section discussing future work opportunities (physical device support, iOS support, cloud integration, machine learning), application scenarios (law enforcement, corporate security,

Date	Hours	Team Member	
			security research, education), and broad conclusions about unified forensic tools. Integrated screenshots and code examples throughout the documentation. Reviewed and edited content for clarity, completeness, and technical accuracy. Ensured all technical details are correct and findings are properly supported by evidence.
2025-11-29	3.5	Ogunseye Israel	**Final Report Contribution - AKT Tool Documentation and Analysis:** Contributed extensively to final report writing, providing comprehensive documentation of testing results, validation findings, and tool analysis. Wrote detailed testing and validation section documenting all testing activities performed throughout tool development, including module testing, integration testing, performance testing, and user acceptance testing. Documented test results with specific metrics, validation outcomes, and identified issues. Created comprehensive comparison section analyzing AKT tool capabilities in relation to existing tools (Andriller, MVT, MobSF), highlighting unique features (real-time monitoring, unified interface), advantages (integrated capabilities, comprehensive monitoring), and limitations (emulator-only, root requirements). Contributed to results and insights section by

Date	Hours	Team Member	
			analyzing testing findings, documenting tool effectiveness in forensic scenarios, identifying use cases where tool excels, and providing recommendations for tool usage. Wrote detailed testing methodology section explaining testing approaches, test case design, validation procedures, and result interpretation. Contributed to conclusions section by providing insights on tool significance, forensic investigation applications, and future enhancement priorities. Created comprehensive testing appendix with detailed test cases, test results, performance metrics, and validation reports. Reviewed and edited all AKT tool-related sections for technical accuracy, clarity, and completeness. Cross-referenced tool documentation with MobSF analysis findings to ensure consistency. Integrated testing screenshots and validation evidence throughout documentation. Ensured all testing results are properly documented and findings are supported by evidence. This collaborative report writing ensured comprehensive documentation of the AKT tool's development, capabilities, and significance.

Appendix A: Screenshots and Evidence

Key Screenshots to Include:

- Andriller extraction interface and reports
- MobSF dashboard and security scorecard
- AKT tool main interface and tabs
- Real-time monitoring output with color-coded events
- Data extraction results
- Security analysis findings
- Root access status and device information

Appendix B: Code Samples

Key code samples from the AKT Forensic Tool are available in the repository at:

- `Misc/novel tool/main.py` - Main application
- `Misc/novel tool/modules/` - All module implementations
- `Misc/Israel Ogunseye/mini_mobsf_summary.py` - MobSF analysis script

Appendix C: Extracted Data Samples

Sample extracted data and reports are available in:

- `Misc/Akinro Akintunde/Andriller/attempt2 to extract/` - Andriller extraction results
- `Misc/Israel Ogunseye/telegram_report.json` - MobSF JSON report
- `Misc/Israel Ogunseye/telegram_mini_summary.txt` - MobSF summary

End of Report

F25_4440_S003_G12 - Forensic Analysis of Mobile Applications

Akinro Akintunde & Ogunseye Israel