Android ADB CLI Utility

Introduction

This Python-based ADB Utility provides a command-line interface to streamline common Android Debug Bridge operations. It offers a user-friendly menu system that allows users to manage USB and Wi-Fi device connections, retrieve detailed device information, perform full or app-specific backups *(with experimental automation features)*, and interact with logcat.

Key Features

Device Connection Management:
- Easily connect and manage both USB and Wi-Fi ADB devices.

Comprehensive Device Information:
- Retrieve device model, manufacturer, detailed system properties, network interfaces, and memory usage statistics.

Backup Operations:
- Perform full device backups or targeted backups of specific applications. Includes experimental features for automated on-device confirmation during backups (requires specific screen coordinate input).

Logcat Interaction:
- Dump the current logcat buffer to a file for analysis and clear the logcat buffer.

Application Management:
- List all installed applications, including their package names and APK paths.

Interactive CLI:
- A menu-driven interface guides users through all available operations.

Forensic Utility

The ADB Utility script offers several functionalities that are valuable in digital forensic analysis of Android devices:

Data Acquisition (Backups)

Full Phone Backup: The ability to perform a full device backup (`adb backup`) is crucial for acquiring a comprehensive image of user data, application data, and device settings. While it creates a `.ab` file (Android Backup) which needs to be parsed, this is a foundational step in many forensic investigations to preserve evidence.

Specific Application Backup: Being able to back up individual application data can be extremely useful for targeted investigations. Many important artifacts (e.g., chats, user activity logs, cached data) reside within specific app data directories. This allows forensicators to isolate and analyze data from applications of interest without necessarily acquiring the entire device.

Device Information Gathering

System Properties (`getprop`): Forensicators can use this to quickly gather critical device metadata such as Android version, build number, manufacturer, device model, security patch level, and various system configurations. This information is vital for understanding the device's environment and for contextualizing other findings.

Network Interfaces (`ip addr`): Information about network configurations, active IP addresses, and MAC addresses can help in reconstructing network activity, identifying connected networks, and tracing communication patterns.

Memory Usage (`procrank`, `dumpsys meminfo`): While primarily for performance analysis, these dumps can sometimes reveal running processes and their memory footprint, which might indicate active malicious processes or processes that were recently active. [cite: 30] For specific applications, `dumpsys meminfo` can help in understanding resource consumption and potential artifacts left in memory.

Activity and Event Logging (Logcat Dumps)

Logcat Analysis: Logcat provides a chronological record of system events, application activities, errors, and debugging messages. This log can contain valuable forensic artifacts, including:

* Application crashes or unusual behavior.
* User interactions (e.g., app launches, screen unlocks).
* Network connection attempts.
* Security-related events or warnings.
* Evidence of malware activity or system tampering.

Clearing Logcat Buffer: The option to clear the logcat buffer before an action could theoretically be used to isolate logs specific to a new activity, although in a strict forensic context, clearing logs is generally avoided unless performing controlled tests.

Application Listing

Installed Applications (`pm list packages`): Getting a complete list of installed applications, along with their package names and APK paths, is fundamental for identifying potentially malicious applications, unauthorized software, or applications relevant to the case. This helps in understanding the attack surface and potential sources of evidence.

Installation and Setup

To use this utility, you need to have ADB configured on your system and enable debugging options on your Android device.

1. Install ADB (Android Debug Bridge):

- Ensure ADB is installed on your system and its executable is accessible via your system's PATH.

- You can download the Android SDK Platform-Tools (which includes ADB) from the official Android developer website.

- Verify installation by opening a terminal/command prompt and typing `adb devices`.

2. Enable Developer Options on your Android Device:

- Go to `Settings` > `About phone` (or `About device`, `System` > `About phone`).

- Locate `Build number`.

- Tap on `Build number` 7 times rapidly until you see a message like "You are now a developer!".

3. Activate USB Debugging:

- Go back to `Settings`. `Developer options` will now be visible (often under `System` or at the bottom of the main `Settings` list).

- Enter `Developer options`.

- Find and enable `USB debugging`. Confirm any prompts.

- Connect your Android device to your computer via a USB cable. On your phone, a "Allow USB debugging?" pop-up will appear. Check "Always allow from this computer" and tap "Allow".

4. Activate Wireless Debugging (for Android 11 and Higher):

- Ensure both your Android device and your computer are connected to the same Wi-Fi network.

- On your Android device, go to `Settings` > `Developer options`.

- Find and enable `Wireless debugging`. Confirm any warnings.

- Tap on the `Wireless debugging` TEXT itself (not just the toggle).

- Tap on `Pair device with pairing code`. A 6-digit Wi-Fi pairing code and an IP address & Port will appear (e.g., `192.168.1.100:41234`). **KEEP THIS SCREEN OPEN** on your phone.

- From the CLI utility, use the "Pair Device Wirelessly (Android 11+)" option and enter the IP:Port and pairing code displayed on your phone. Alternatively, you can do this manually from your PC's terminal:

  - `adb pair <IP_ADDRESS_FROM_PHONE>:<PORT_FROM_PHONE>` (e.g., `adb pair 192.168.1.100:41234`)

  - When prompted, enter the 6-digit pairing code.

- Once paired, use the "Connect to Device via IP and Port" option in the CLI utility, or manually run:

  - `adb connect <IP_ADDRESS_FROM_PHONE>:<PORT_FROM_PHONE>` (e.g., `adb connect 192.168.1.100:41234`)

- You can verify the connection with `adb devices`.

How to Use

1. Follow the Menu: The application will present an interactive menu. Select options by entering the corresponding number and pressing Enter.

- "ADB Device Activation Tutorial": Provides detailed on-screen instructions for enabling ADB on your device.

- "Get Device Information: Allows you to retrieve various details about a connected device and save them to a file.

- "Manage USB Connections": Check USB connection status and lists connected devices.

- "Manage Wi-Fi Connections": Connect to new Wi-Fi devices, pair wirelessly, set devices to TCP/IP mode, and disconnect.

- "Perform Backup Operations": Initiate full phone or specific application backups, with options for partial automation.

2 Project Structure:

* `main_cli.py`: The main command-line interface application, orchestrating user interaction and calling functions from other modules.

* `adb_utils.py`: Contains core utility functions for executing ADB commands, handling command output, and managing serial formatting.

* `adb_connectors.py`: Manages the logic for establishing and checking ADB connections over both USB and Wi-Fi.

* `adb_automator.py`: Provides functions for automating basic on-screen device interactions such as tapping, text input, and key presses, primarily used for experimental backup confirmation.