AndroRAT: An Android Remote Access Tool

Group Members:
1. Mahfuzzaman Sizan (1905054)
2. Nur Uddin Ibne Huda (1905056)

March 10, 2024

Contents

1 Introduction

Mobile devices have become integral to modern life, offering convenience and connectivity. However, their widespread usage also attracts malicious actors seeking to exploit vulnerabilities for various purposes, including espionage, data theft, and surveillance. Among the arsenal of tools employed by attackers, AndroRAT stands out as a potent instrument for remote access and control of Android devices. Developed as a client-server application, AndroRAT allows remote manipulation of Android systems, posing significant security concerns.

2 Installation

2.1 Cloning the Repository

To get started with AndroRAT, follow these steps to clone the repository from GitHub:

```
git clone https://github.com/karma9874/AndroRAT.git
```

2.2 Installing Dependencies in Python

After cloning the repository, navigate to the AndroRAT directory and install the required Python dependencies using pip:

```
cd AndroRAT
pip install -r requirements.txt
```

Figure 1: Cloning the AndroRAT repository

2.3 Available Modes

AndroRAT offers various modes for different functionalities:

- **Build Mode**: Used for building the Android APK.
- **Shell Mode**: Used for obtaining an interactive shell of the device.

2.4 Connecting to Mobile and PC

To install AndroRAT on the target Android device and connect it to your PC, follow these steps:

1. **Install the APK**: Transfer the generated APK to the target phone and install it. If security apps prevent installation, force install it.
2. **Open Interpreter from the Attacker PC**: Use the following command to open an interpreter in the attacker PC:

```
    python3 androRAT.py --shell [flags]
```

   If the interpreter doesn't open in Android versions greater than 9, click on the APK file manually.

2.4.1 Build Mode Flags

For the build mode, you can use the following flags:

- **-p, –port**: Attacker port number (optional; default is set to 8000).
- **-o, –output**: Name for the APK file (optional; default is set to "karma.apk").
- **-icon, –icon**: Visible icon after installing APK (optional; by default set to hidden).

Example Usage for Build Mode:

```
python3 androRAT.py --build --ngrok -o evil.apk
```

2.4.2 Shell Mode Flags

For the shell mode, you can use the following flags:

- **-i, –ip**: Listener IP address.
- **-p, –port**: Listener port number.

Example Usage for Shell Mode:

```
python3 androRAT.py --shell -i 0.0.0.0 -p 8000
```

Figure 2: Building the AndroRAT APK

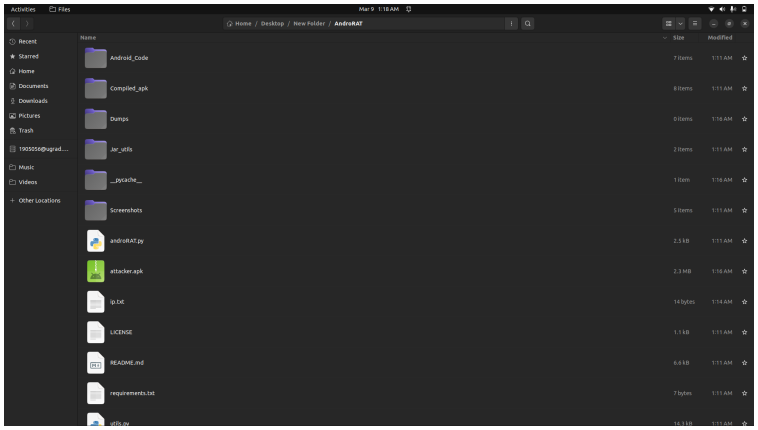

Figure 3: APK file generated

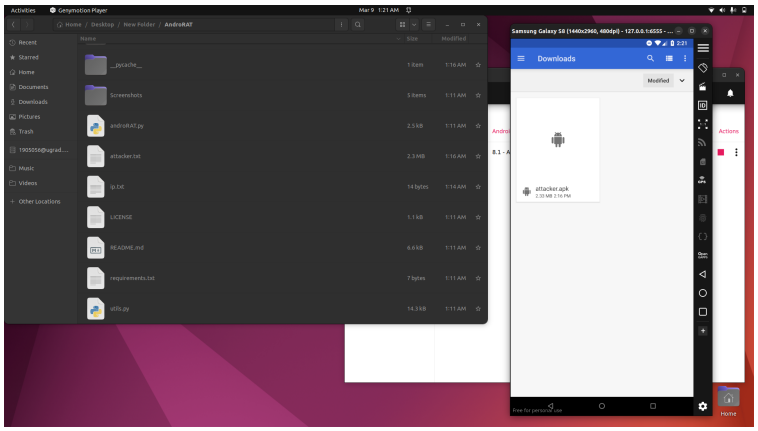

Figure 4: Transferring APK to target device

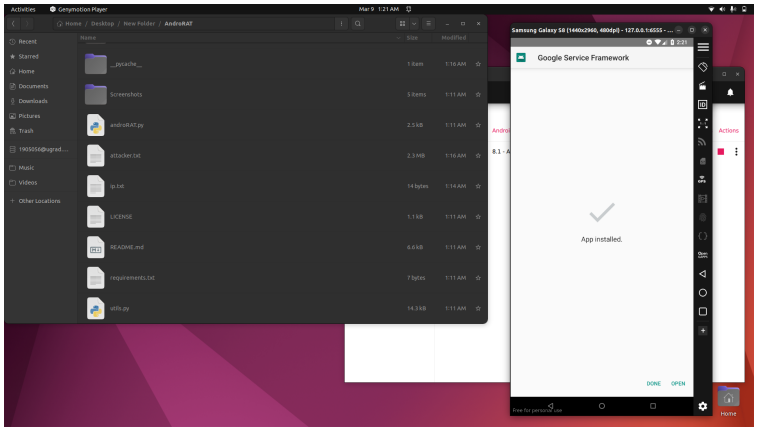

Figure 5: Installing the APK on target device

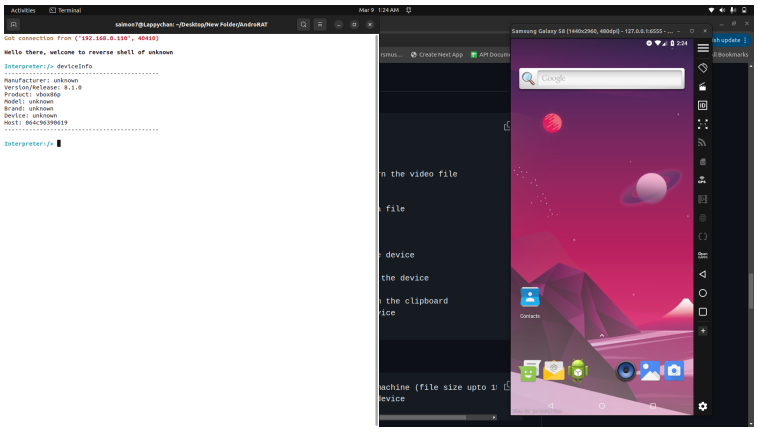

Figure 6: Connected to shell

3 Features of AndroRAT

3.1 Full Persistent Backdoor

AndroRAT provides a highly capable and long-lasting form of unauthorized access to the target device. Here are some key aspects of its full persistent backdoor functionality:

- **Full Access:** The backdoor offers complete access to the compromised system, allowing the attacker to execute various operations such as stealing data, manipulating settings, or executing arbitrary commands.
- **Persistence:** AndroRAT can run in the background indefinitely until detected by highly functional anti-spyware tools or manually stopped by a user. The app's ability to hide its icon reduces the likelihood of detection.
- **Backdoor Entry:** A backdoor serves as a hidden or undocumented means of accessing a computer system, providing unauthorized access without normal authentication procedures.

3.2 Invisible Icon on Install

Upon installation, AndroRAT does not create any visible icon on the target Android phone. However, it's worth noting that icon invisibility only works on Android versions lower than 9. Additional properties include a lightweight APK that runs continuously in the background, and the app automatically starts on boot-up.

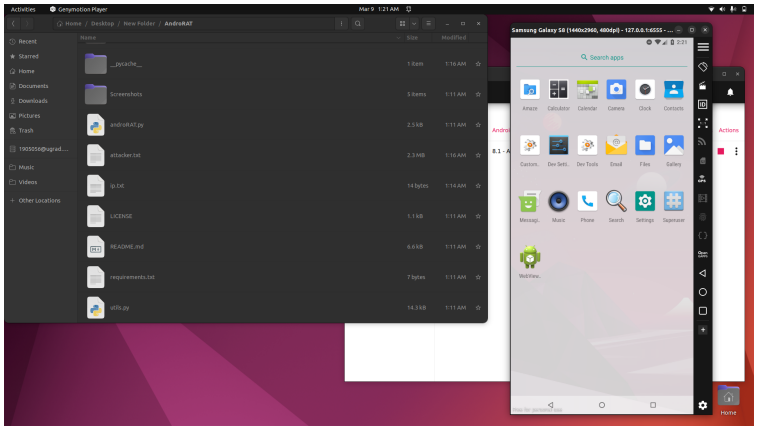

Figure 7: No visible icon on the device

3.3 Audio, Video, and Camera Control

AndroRAT offers extensive control over audio, video, and camera functionalities of the target device:

- **Camera Control:** Users can access both the front and back cameras of the target device.

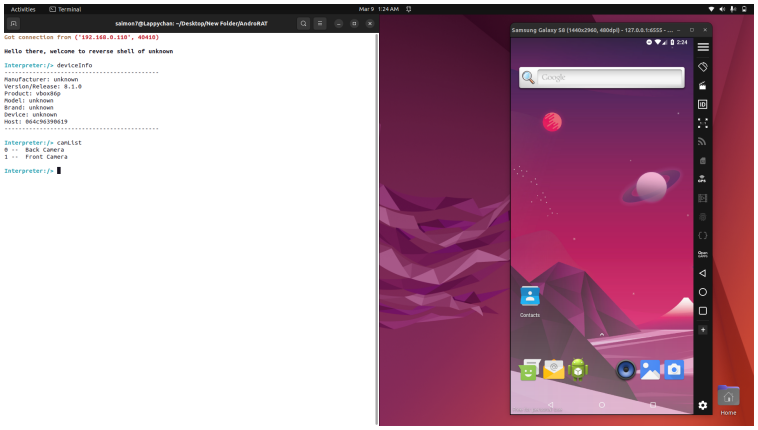

Figure 8: List of available cameras

- **Capturing Images:** AndroRAT can capture images discreetly and save them locally.

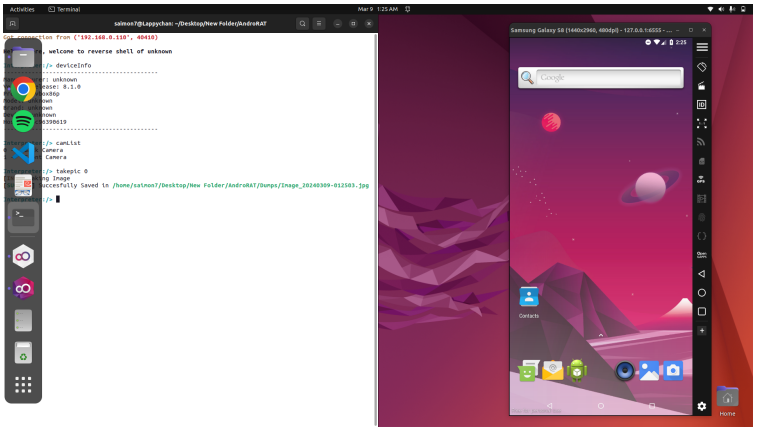

Figure 9: Image captured by AndroRAT

- **Video Controls:** Users can initiate and stop video recording, with the recorded videos saved locally.

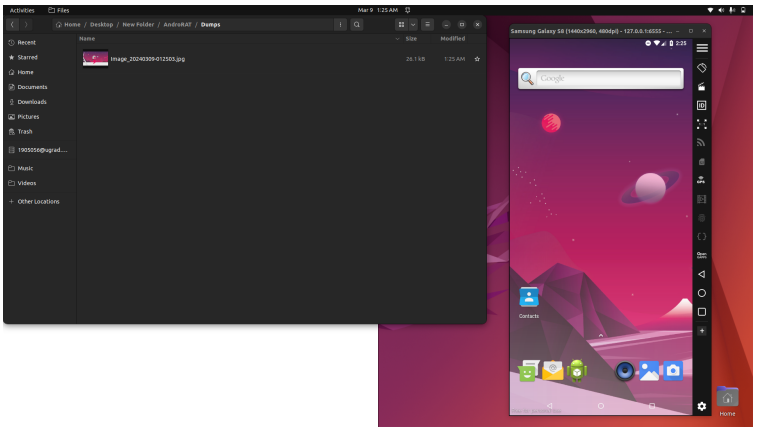

Figure 10: Captured image stored locally

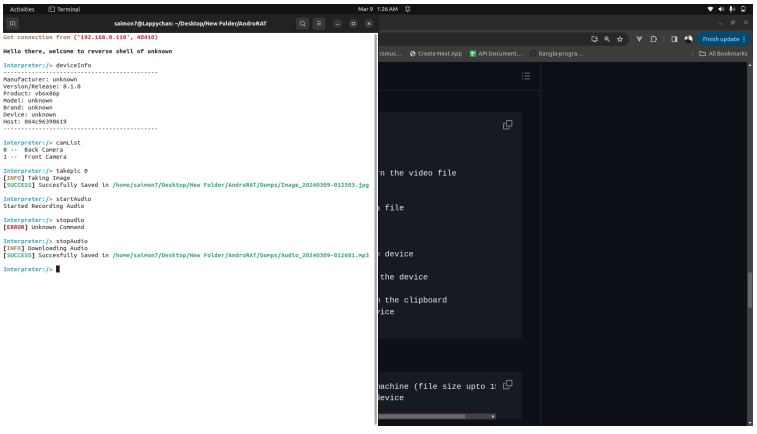

Figure 11: Starting and stopping audio recording

3.4 Call Logs and SMS Logs

AndroRAT enables browsing of call logs and SMS logs from the target device:

- **Call Control:** Users can retrieve call logs from the target device, which are saved locally.

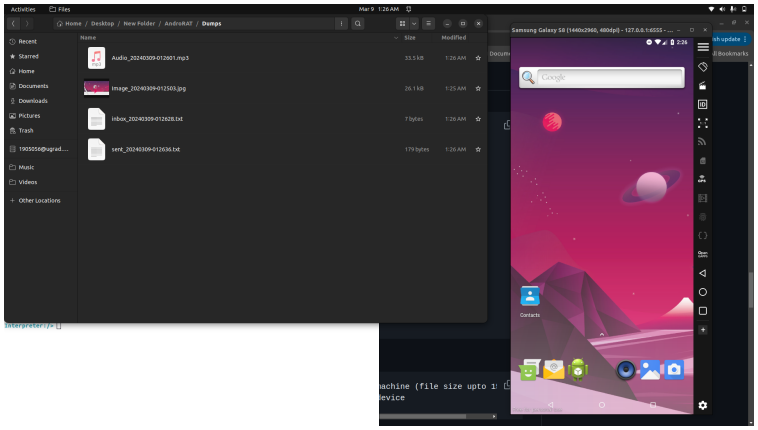

Figure 12: Importing SMS logs

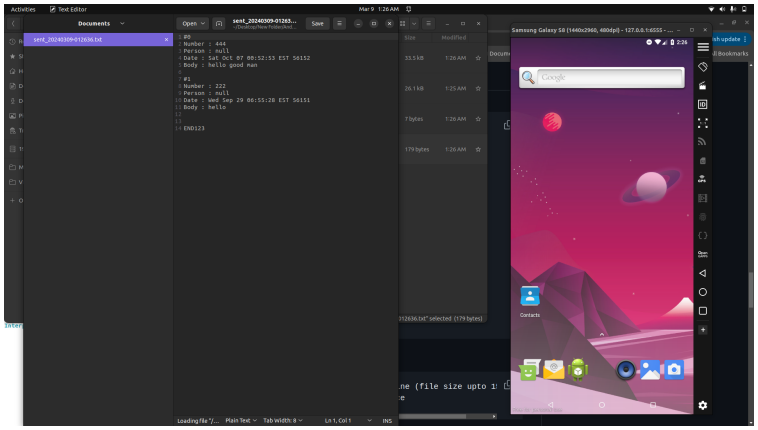

Figure 13: File containing sent SMS logs

- **SMS Control:** AndroRAT allows access to SMS messages, with the option to retrieve both inbox and sent messages.

3.5 Opening Shell

AndroRAT facilitates opening a shell on the attacker's device, providing an interactive command-line interface for further operations.

3.6 Other Tested Features

Apart from the mentioned functionalities, AndroRAT supports additional features:

- **Vibration Control:** Users can command the target device to vibrate a specified number of times.
- **Location Retrieval:** AndroRAT can retrieve the current location of the target device.
- **SIM Details:** Users can access details of all SIM cards present in the target device.

4 High Level Overview of Source Codes

4.1 MainActivity.java

The `MainActivity.java` class performs the following tasks:

- Logs the IP address and port from the `Config` class.
- Initiates an instance of the `tcpConnection` class to establish a reverse shell connection by executing its `doInBackground` method with the specified IP address and port.
- Hides the app icon from the device's launcher if the `Config` flag is set to true.

4.2 tcpConnection.java

The `tcpConnection.java` class encompasses the following functionalities:

- Defines an `AsyncTask` called `tcpConnection` that runs in the background and handles the reverse shell connection.
- Imports necessary Android libraries and custom payloads for various functions such as camera control, audio recording, SMS retrieval, and more.
- Maintains a persistent connection loop, attempting to connect to a specified IP address and port indefinitely.
- Sets up input and output streams to communicate with the remote server upon connection establishment.
- Listens for commands from the server and performs actions based on the received commands, including taking pictures, running shell commands, retrieving device information, managing SMS messages, etc.
- Handles exceptions and errors during the connection, attempting to re-establish the connection in case of failure.
- Supports background services and job scheduling for different Android versions.

4.3 broadcaseReceiver.java

The `broadcaseReceiver.java` class serves as a BroadcastReceiver designed to monitor the status of the `mainService` and restart it if it's not already running. This ensures that the specified background service is kept alive in the Android application.

4.4 functions.java

The `functions.java` class comprises a collection of utility functions in an Android application, including:

- `deviceInfo()`: Retrieves various device information such as manufacturer, OS version, product, model, brand, device name, and host.
- `readFromClipboard()`: Reads text data from the clipboard, if available, and returns it as a string.
- `jobScheduler()`: Schedules periodic jobs using the Android JobScheduler API to run background tasks at regular intervals.
- `getPhoneNumber()`: Retrieves information about the SIM card(s) installed on the device, including call state, IMEI/MEID numbers, mobile number, serial number, SIM operator name, SIM state, and SIM country ISO code.
- `getScreenUp()`: Wakes up the device and shows the screen by adding specific flags to the activity's window.
- `hideAppIcon()` and `unHideAppIcon()`: Programmatically hide and unhide the application's icon from the device's launcher, respectively.
- `overlayChecker()`: Checks whether the application has overlay permission and provides instructions to enable it if necessary.
- `createNotiChannel()`: Creates a notification channel for the application, primarily for Android versions 8.0 (Oreo) and higher.

5 Conclusion

In conclusion, the analysis of AndroRAT reveals the intricate landscape of mobile device security threats and the sophisticated tools employed by malicious actors. The investigation into AndroRAT's installation process, features, and code structure underscores the urgent need for robust security measures and user awareness.

The installation process of AndroRAT, encompassing both build and shell modes, highlights the ease with which such tools can be deployed on target devices. By leveraging Git cloning and Python dependencies, attackers can quickly set up AndroRAT instances to infiltrate Android devices covertly.

Moreover, the features of AndroRAT, including its full persistent backdoor, invisible icon on install, and extensive surveillance capabilities, pose significant risks to user privacy and data security. From capturing audio, video, and images to browsing call logs and SMS messages, AndroRAT empowers attackers with comprehensive access to sensitive information.

Furthermore, the high-level code overview sheds light on the intricate workings of AndroRAT's Java classes, illustrating how each component contributes to the tool's functionality. From the MainActivity initiating the reverse shell connection to the tcpConnection handling communication with the remote server, the code structure reveals the sophistication of AndroRAT's design.

In light of these findings, it is imperative for mobile users and organizations to implement proactive security measures, including regular software updates, antivirus software, and user education programs. Additionally, developers must prioritize security in the design and implementation of mobile applications to mitigate the risks posed by remote access tools like AndroRAT.

Overall, the analysis of AndroRAT serves as a stark reminder of the evolving threat landscape in the mobile security domain and the critical importance of vigilance and preparedness in safeguarding against emerging threats. By understanding the capabilities and implications of tools like AndroRAT, stakeholders can take proactive steps to protect their devices and data from exploitation by malicious actors.