CASE STUDY 2

Malicious App Entry to a Mobile Device: A Controlled Android APK Sideloading Simulation Case Study

By: Joseph Gonzalez

ZERODAY TECH LABS

Scope	Controlled lab simulation on a virtual Android 8 test device using self-operated infrastructure; educational and analytical purposes only
Environment	Kali Linux, AndroRAT, Android Studio emulator, local Apache delivery service, Chrome, redacted localhost/LAN artifacts
Primary Evidence	APK build screenshot, local server staging screenshot, listener screenshot, APK download screenshot, redacted callback terminal capture
Primary Attack Vector	User-approved APK sideloading through a browser download path, followed by execution and observable callback inside the lab

For educational and analytical purposes only. Local network details shown in figures have been redacted.

Introduction

Mobile devices are attractive initial-access targets because software installation often happens quickly, on a trusted personal device, and under ordinary browsing conditions. Unlike a credential-only phishing page, a malicious application package can shift the risk from a single entered password to persistent device-side exposure once the package is downloaded, approved, and opened.

This case study examines a controlled sideloading scenario. The purpose is to show how the human decision to approve an untrusted app can become the entry point, while keeping the discussion defensive and focused on what a reader should recognize and prevent.

Methodology

A controlled cybersecurity simulation was conducted in an isolated lab using Kali Linux, a local web server, AndroRAT, and an Android Studio emulator configured as a virtual Android test device. The scope was limited to self-controlled infrastructure, self-generated artifacts, and a virtual handset only. No real users, public hosts, third-party phones, or production accounts were involved.

1. Overview

A malicious APK entry path differs from web credential theft because the attacker is attempting to place executable code on the handset itself. In this case, the analytical question was whether a locally delivered APK would progress from download to execution and generate an observable callback inside the lab.

2. Threat Analysis: Vector, Technique, and Human Factor

Attack vector used	Browser-based delivery of a sideloaded APK, followed by user approval of installation from outside the trusted app-store path.
Social engineering techniques	Trojan-style app delivery, pretexted download, tap-through installation, and trust exploitation through a normal-looking file-download workflow.
Human factor exploited	Users often treat app downloads as routine, assume small files are harmless, and approve prompts quickly when the requested action appears to match what they intended to do.
Defensive relevance	Reducing sideloading, using trusted app stores, checking publisher identity, reviewing permissions, keeping Play Protect enabled, and updating the device materially reduce the entry path.

Key reader takeaway: the dangerous moment is not only the download. The real shift happens when the user approves installation and opens the package, turning the phone from a browsing device into the platform for follow-on exposure.

3. Attack Setup

The lab host was used to prepare the test application package and stage it through a local delivery path. Figures 1 and 2 document the package build and local server staging phases. The significance of this phase is that an apparently ordinary application file could be prepared and exposed through a browser-friendly delivery mechanism.

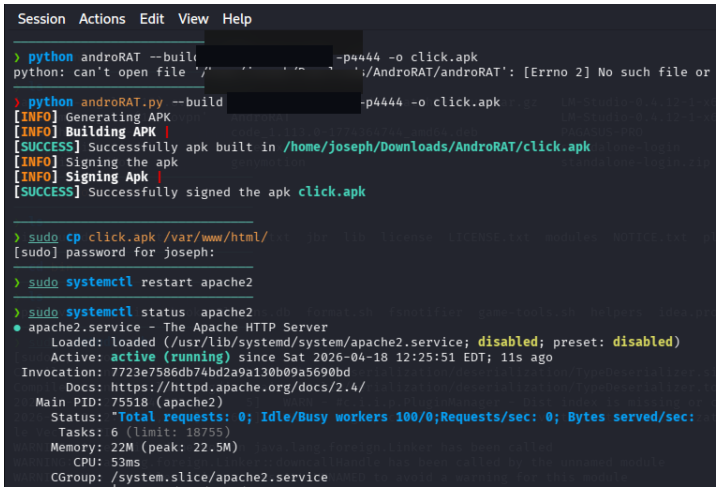

Figure 1. APK generation on the lab host, with local addressing redacted from the terminal capture.

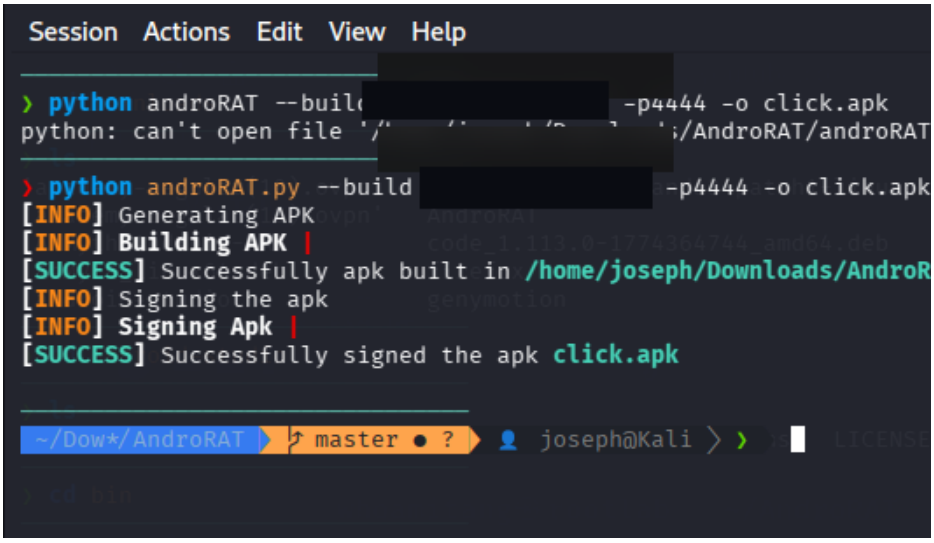

Figure 2. Local server staging and status verification for the delivery path used in the controlled simulation.

4. Delivery, Installation, and Callback Evidence

Before the device interaction, the listener was placed in a waiting state, as shown in Figure 3. The virtual handset then navigated to the locally hosted file and received a standard APK download prompt, shown in Figure 4. To the user, the package appeared as a downloadable application file rather than as an obvious exploit.

Figure 3. Listener prepared on the lab host before the device-side interaction occurred.

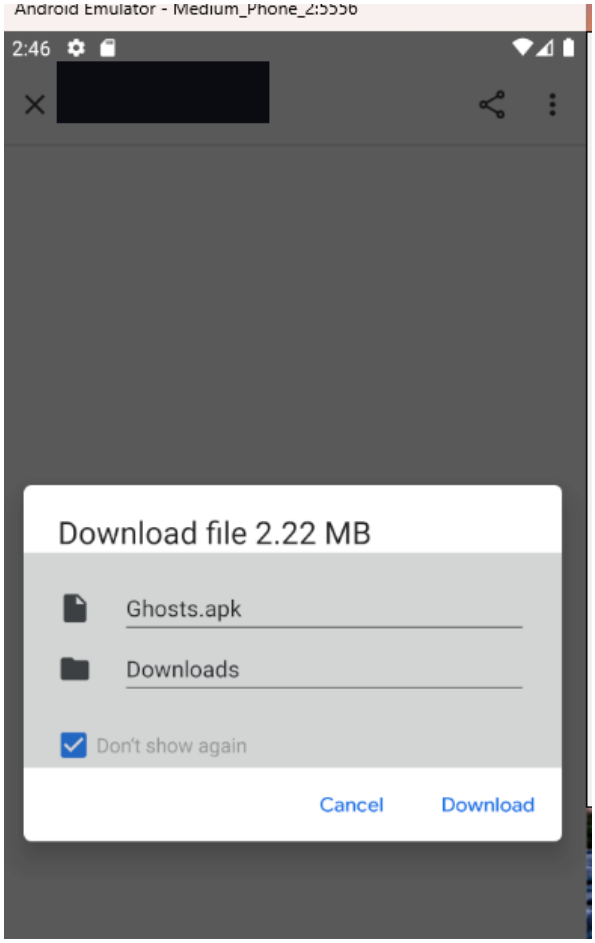

Figure 4. Virtual Android device presenting the sideloaded APK download prompt after navigating to the locally hosted file.

After the application was opened on the virtual device, the listener registered an inbound callback. Figure 5 is the primary forensic artifact because it shows that the APK did not merely arrive on disk; it executed and initiated communication back to the lab host. A downloaded file is potential exposure. A successful callback is evidence of initial device-side execution.

```
Session  Actions  E
Got connection fro
GET /
Host:
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 8.1.0; Pixel 7 Build/OPM6.171019.030.E1; wv) AppleWebKit/537.36 (KHTML, li
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-e
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
X-Requested-With: org.chromium.webview_shell

Interpreter:/> camlist
Interpreter:/>
```

Figure 5. Listener output after the APK was opened on the virtual device, showing the inbound connection and request context with local network details redacted.

5. Why the Exploit Worked on Android 8

The result is consistent with how Android 8 handles software from outside the trusted store path. Installations from non-Play sources are governed per source, meaning a user can explicitly allow a browser or file-handling path to install unknown apps. If that approval is granted, the package installer can continue the workflow even though the application did not originate from Google Play.

The case also highlights a limit of scanning-based protection. Google Play Protect is an important security layer, but protections depend on detection, configuration, device age, and user decisions. In this lab instance, no meaningful interruption was observed before the callback occurred. The safer conclusion is that user-approved sideloading on a legacy Android 8 environment can still provide a viable entry path.

6. Real-World Impact to the Individual

For an individual user, a malicious app install can be more damaging than a one-time phishing page because the phone itself may become the instrument of surveillance or follow-on fraud. Depending on the application capabilities and permissions, consequences can include exposure of messages, calls, screenshots, audio, location data, device identifiers, or other personal artifacts.

The personal impact is amplified because a smartphone concentrates identity, communication, photos, banking access, cloud sessions, and recovery channels in one place. Once trust in the handset is broken, the user may need to reset accounts, rotate credentials, revoke sessions, re-enroll authentication methods, and treat the device as untrusted until it is cleaned or rebuilt.

7. Mitigation, Patching, and Prevention

The patch path begins with removing the entry route. Real devices should run a supported Android version, stay current on operating system and security updates, keep Google Play services and Play Protect enabled, and avoid granting install rights to browsers or file-handling paths unless there is a clearly documented need.

After a suspected malicious install, response should be immediate: disconnect the device from sensitive accounts, uninstall the application, review high-risk permissions such as accessibility or device administration access, rotate passwords beginning with email, revoke active sessions, and update the device.

8. Reader Preparation Checklist

Before installing	Use trusted app stores, verify the publisher, and avoid granting install rights to browsers or file managers without a documented reason.
During the prompt	Treat unknown-source installation as a high-risk decision, even when the file name or download page appears familiar.
After a suspected install	Remove the app, review high-risk permissions, revoke sessions, rotate passwords beginning with email, and update or rebuild the device if trust is lost.

9. Ethical Consideration

This case study was conducted in a controlled environment strictly for educational and analytical purposes. The delivery path, listener, and virtual device were operated by the researcher using self-controlled resources, and local network identifiers visible in the original evidence were redacted before publication. No real users, public distribution channels, or third-party devices were involved.

The defensive lesson is straightforward: reduce sideloading, keep the platform current, and treat unexpected app-install prompts as high-risk decisions rather than routine tap-through screens.