



GHOST
PROTOCOL

MA

THE FIELD LIBRARY

DEEP TRACK

THE APP AND THE WIRE

Mobile and APIs

How a mobile app is built, Android and iOS testing, API security, BOLA and the API classes.



Ghost Protocol (Pvt) Ltd · Colombo · ghosts.lk

Security internship training · The Field Library · Edition 1 · 2026

THE GHOST PROTOCOL FIELD LIBRARY · DEEP TRACK

Mobile and APIs

How a mobile app is built, Android and iOS testing, API security, BOLA and the API classes.

Ghost Protocol (Pvt) Ltd · Colombo

The Field Library, Edition 1 · 2026

Copyright 2026 Ghost Protocol (Private) Limited. All rights reserved.

Sold as a PDF for the buyer's own reading. Please do not share or resell the file.

The library

The Field Library is a whole shelf, not one book. The Core Path is the spine everyone reads. The rest is depth and breadth, taken on as the work calls for it. The shelf is complete now, and you are reading the one marked here.

A · THE CORE PATH

O The Oath BUILT

I Groundwork BUILT

II The Web BUILT

III The Wider Field BUILT

IV The Work BUILT

V The Craft BUILT

B · DEEP TRACKS

AI and Agents BUILT

Reading Code BUILT

Internal and Active Directory BUILT

The Web, Deeper BUILT

Cloud and Containers BUILT

Mobile and APIs YOU ARE HERE

Reverse Engineering BUILT

Hardware and the Edges BUILT

C · THE OPERATIONS

Red and Purple Team BUILT

OSINT and Recon BUILT

Report Writing Mastery BUILT

The Consulting Craft BUILT

Bug Bounty BUILT

D · THE OTHER SIDE

Detection and Response BUILT

Threat Hunting and Forensics BUILT

Hardening and Architecture BUILT

E · THE GROUND RULES

The Law, Deeper BUILT

Standards and Compliance BUILT

Ethics and Disclosure BUILT

F · COMPANIONS

The Reference Pack BUILT

The Lab Workbook BUILT

Case Files BUILT

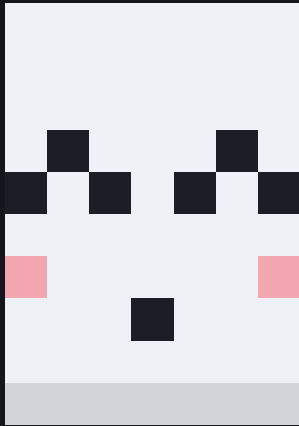
The Bestiary BUILT

The Glossary and the Ledger BUILT

Contents

- Opening 5
- 1** How a Mobile App Is Built 6
- 2** Android Testing 10
- 3** iOS Testing 16
- 4** API Security 20
- 5** BOLA and the API Classes 25
- Worked answers 32

The app and the wire



The scope was a mobile app, and Blink had already installed it, poking at buttons, delighted by the animations.

BLINK "I am testing the app. Look, it has a nice login."

SOL "That is the easy half. The app on the phone is a client, the same as a browser, and you will pull it apart: its storage, its secrets, how it talks. But the real target is the wire behind it. The app asks an API for things, and the API, not the app, decides what you are allowed to see. Most of the serious findings here are one account asking the API for the account next door and being handed it."

Worm's eye opened at the word decides, because the thing that decides is the thing you test.

WYRM "the wire."

So the book is two things that are really one: the app in your hand, and the API it speaks to. The app is where you start. The wire is where the account next door leaks.

CHAPTER ONE

How a Mobile App Is Built

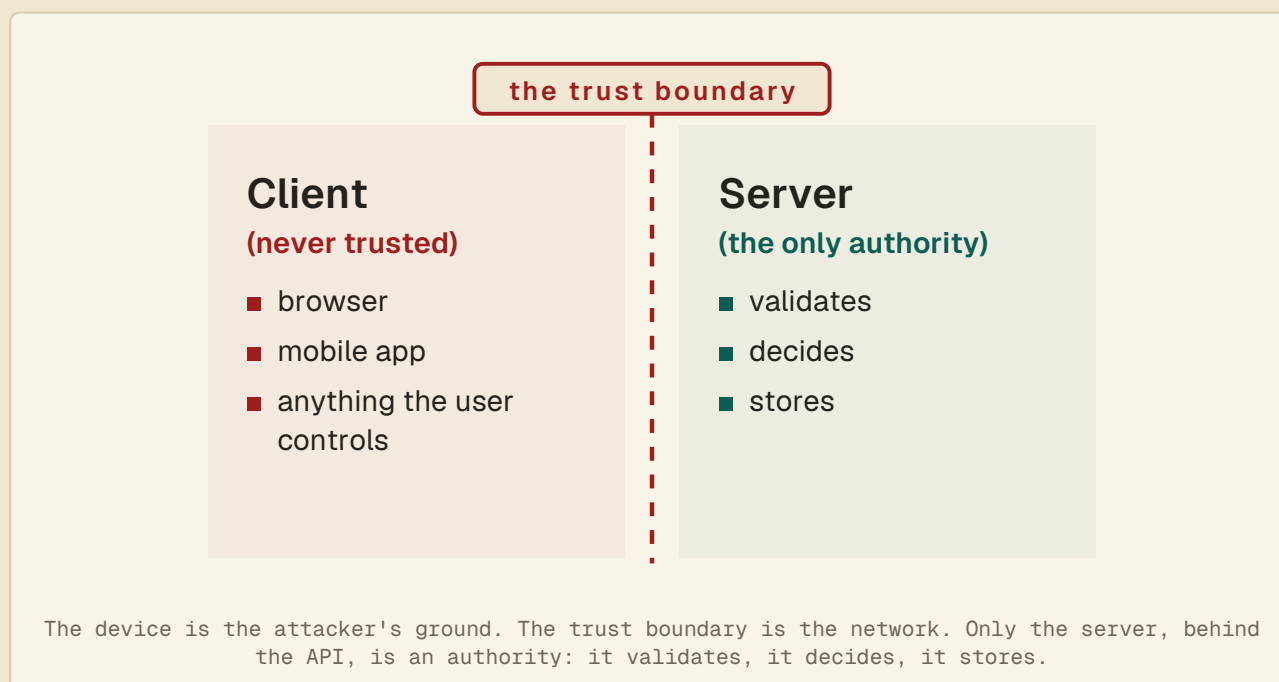
The app on the phone looks like the whole product. It is the smaller half. This chapter shows you the mobile stack, the API behind it, and the one fact that decides every finding: the client is running on the attacker's device.

Volume III introduced mobile as a thin client over an API you already know how to think about. That was the awareness. This volume is the depth, and it starts where the depth has to start: with a clear picture of what an app actually is, which parts of it the user controls, and where a security decision is allowed to live. Almost every mobile finding is a decision that was left on the device when it belonged on the server.

The client is on the attacker's device

A web application keeps its code on a server. You see the HTML it chooses to send and nothing else. A mobile application is the opposite arrangement: the client is a file, an Android package or an iOS build, and it is installed on hardware the user owns completely. Whoever holds the phone can copy that file, decompile it, read the strings inside, run it under a debugger, change it, repackage it, and watch every byte it sends over the network. Nothing about the binary is private from the person it runs on.

Volume I taught this for the browser and Volume II made it the spine of the web methodology: the client is never a trusted place. On a phone it is truer, because the attacker does not merely control the inputs to your code, they control the code. A check written in the app is a check the attacker can read and delete. A key compiled into the app is a key the attacker can extract. The question for every rule the app seems to enforce is simple and unforgiving: is this rule also enforced by the server, or is the app the only thing holding the line?



The stack: Four layers and a wire

Open any app and it is built in layers, and each layer is a different kind of target. Name them before you test, because a finding in one is a different thing from a finding in another.

- **The interface layer.** Screens, buttons, the view logic. This is what the user sees and the least interesting place to attack, because it decides nothing that the server could not overrule.
- **Local storage.** What the app writes to the device: preference files, a small SQLite database, cached files, logs. Sensitive data left here in clear is one of the two most common mobile findings.
- **Platform services.** The operating system APIs the app calls: the secure key store, biometrics, the camera, inter-app messaging. These are the platform's to get right, and the finding is usually that the app did not use the secure one it was given.
- **The network client.** The code that talks to the backend. Follow this layer and it leads you off the device entirely, across the wire, to the API. That is where the other most common finding lives.

Past the network client is the wire, almost always HTTPS, and past the wire is the server side: the API, its authentication, its business logic, and its database. That server side is not on the phone at all. It is a web target, and the whole of Volume II applies to it unchanged. Half of mobile testing is realising that the interesting half is not mobile.

Thin client, thick client

Apps sit on a spectrum. A **thin client** is a shell: it draws screens and sends every real decision to the API, which holds the logic and the data. A banking app that shows your balance but computes nothing is thin. A **thick client** keeps real work on the device: an offline-first notes app with its own database, a game with local rules, an app that runs a model on the phone. Most commercial apps lean thin, which is good news, because it means the money finding is usually the API, and you already know how to test an API.

Where an app is thick, the device holds logic worth reading and data worth protecting, and the on-device analysis in the next two chapters earns its place. Decide which kind of app you are holding early, because it tells you where to spend your hours.

Where trust must not live

Two rules fall straight out of the first fact, and between them they account for most of what you will report.

Every security decision belongs on the server. The price of an item, whether a discount is valid, whether this user is an administrator, whether a purchase actually happened, whether an account may see a record: all of it is decided by the API and never by the app. The app may grey out a button or hide a screen, and that is fine as a courtesy to honest users. It is not a control. If the only thing stopping a request is code on the phone, there is nothing stopping the request, because the person sending it owns the phone.

No secret belongs in the client. Hardcoded API keys, cryptographic keys, passwords, credentials for internal systems: a value compiled into an app is a published value. People unpack and read binaries

routinely, with ordinary tools. "It is inside the binary, nobody will look" is not a control, it is a delay measured in minutes. A key the app needs to talk to a third party must be brokered by your own server, so the real secret stays off the device.

**SOL SAYS**

When an intern says a mobile app is secure, I ask them one question: what happens if the user is hostile and owns the device? Every honest answer to that question is a finding or a reason there is none. The app is not your client's private property once it is installed. It is a copy in the hands of someone who may want to take it apart, and the only things that hold under that pressure are the ones the server enforces. Test the app as though the person holding it is the attacker, because sometimes they are.

What that means for how you test

Mobile testing is three surfaces at once, and this volume gives each one a home.

- **The API behind the app.** Proxy the traffic, read the calls, and test the backend as the web target it is. This is where the money usually is. Chapters Four and Five are this surface at depth.
- **What the app decides on the device.** Read and instrument the binary to find logic and secrets the developers assumed nobody would see. Android is Chapter Two, iOS is Chapter Three.
- **What the app stores on the device.** Inspect local storage for sensitive data left in clear, and check whether the platform's secure store was used. Also Chapters Two and Three.

The thread through all three is the oldest lesson in the Library, carried from Volume I and Volume II and now applied to a surface where it bites harder: the client is not trusted, so the server has to be the authority, and your job is to find every place it is not.

■ THE LINE

You test a mobile app on your own device or emulator, and you test a client's app only under a written authorisation that names it, exactly as Volume 0 requires. An app being free to download is not permission to attack the servers it talks to. That backend is somebody's production system, and calling it without authorisation is unauthorised access under the Computer Crime Act, the same offence the whole of Volume 0 is built around. Pull a stranger's app, decompile it to learn, and the moment you point its traffic at the live backend you have crossed the line. Practise on apps you own and on the sanctioned training apps named through this volume.

■ VERIFY IT

Before you call a value "enforced on the server" or "trusted from the client", prove which one it is, because the whole report hangs on the difference. The claim is about the server's behaviour, not the app's screen. To show a value is server-authoritative, change it in the proxy to something the app would never send and show the server rejects or ignores it. To show it is client-trusted, change it and show the server accepts it and acts on it. A greyed-out button in the app proves nothing either way. One tampered request and the server's response is the evidence.

■ EXERCISE

On an Android emulator you own, install DIVA (Damn Insecure and Vulnerable App), a training app built for exactly this. Put a proxy between it and the network and list every API call it makes, if any, and every file it writes to the device. Then draw the boundary for one feature: name one thing the app decides or stores on the device, and one thing it asks the server for. For the on-device decision, say what would happen if a hostile user changed it, and why. You are not exploiting yet. You are learning to see the four layers and the wire in a real app.

■ ON YOUR FEET

A prospect says: "The product is the app. Just test the app, it is one thing." You download it and proxy it for ten minutes. It makes forty calls to an API on one host, loads images from a second host, and sends crash reports to a third. What does "test the app" actually include once you have seen that, which of those surfaces are in scope and which are not yours to touch, and what is the single most important sentence you put in the Rules of Engagement before you start?



WHAT WYRM KEEPS

A mobile app is a client running on the attacker's own device. The trust boundary is the network. Only the server, behind the API, is an authority. Most findings are a decision left on the device.

- The stack is four layers (interface, local storage, platform services, the network client) and a wire to the API. The API is a web target: Volume II applies to it unchanged.
- Thin clients put the logic and data on the server (most apps, so the API is the money); thick clients keep real work on the device (so the on-device analysis earns its place). Decide which you hold first.
- Two rules: every security decision is the server's, and no secret belongs in the client. A key compiled in is a published key.
- Three surfaces: the API behind it, what the app decides on the device, what it stores. Own device or emulator and sanctioned apps only; a downloadable app is not permission to hit its backend.

CHAPTER TWO

Android Testing

The APK is a file you can take apart. This chapter covers reading it statically with jadx, driving it at runtime with Frida and objection, getting past certificate pinning, finding data left in clear, abusing exported components and deep links, and the one place a key is allowed to live: the Keystore.

Android is the friendlier of the two platforms to learn on, because the tooling is open, the emulator is free, and the package is easy to open. The work divides cleanly: read the app without running it (static), watch and change it while it runs (dynamic), look at what it left on disk (storage), and check the doors it opens to other apps (components). Everything here is done on a device or emulator you own, against your own app or a sanctioned training app. Nothing in this chapter is for a phone that is not yours.

The APK, and what is inside it

An **APK** is a ZIP archive with a fixed shape. Rename it to `.zip` and you can list it, which is the first thing worth doing because it tells you what you are dealing with. Inside:

- `classes.dex` (one or more): the app's code, compiled to Dalvik bytecode. This is what you decompile.
- `AndroidManifest.xml`: the app's declaration of itself. Its components, its permissions, and the flags that matter most to you. In the APK it is in binary form; the tools below render it readable.
- `resources.arsc` and `res/`: strings, layouts, and other resources, sometimes holding URLs and keys developers thought of as "just resources".
- `lib/`: native libraries (`.so` files) per processor architecture, for code written in C or C++.
- `assets/`: raw bundled files, occasionally an entire configuration or a secret.

Modern apps ship from the store as an Android App Bundle, and the device may install several split APKs (by language, density, architecture). For testing you want the installed set, which you can pull from a device you own with `adb`. The point to hold is that the code and its secrets are all in that archive, and the archive is yours to read.