# Protecting Your **Android Apps** From Hacking

## A Complete Guide

PreEmptive


# The *State* of Android Security

Android is the most popular operating system in the world. More than 2.7 billion people in nearly 200 countries worldwide use it on their devices every day. Launched as the open-source alternative to iPhone, Android quickly became the preferred operating system for mobile phone manufacturers in the early 2010s. Samsung, Motorola, HTC, and LG offered Android-powered phones, and the operating system quickly grew. Now, its global market share is over 70%. More than one billion Android devices were shipped worldwide in 2021, with Samsung being the largest manufacturer.

Android apps are used for almost everything, including mobile banking, shopping, and more. Unfortunately, they're also vulnerable to cyberattacks. Hackers use a variety of tactics, including malware, code injection, and reverse engineering, to gain unauthorized access to source code or other app inner workings and exploit them. Unfortunately, too many developers make it easy by putting security on the back burner when developing applications. According to a 2022 survey, 86% of developers do not view application security as a top priority when writing code.

That perspective must change, and developers are perfectly positioned to prioritize application security during the development cycle by taking a proactive approach to writing secure applications on purpose. Tactics like using best practices, staying current with knowledge and understanding of potential security issues, keeping abreast on industry trends and topics, and integrating security tools into the software development life cycle are among the most effective.

And it seems like there's a lot of room for improvement in all of these capacities. Earlier this year, it was reported that the 14 top Android apps, which have 140+ million installs had security flaws that could potentially expose user data. A 2021 report analyzed more than 150 apps across four financial sectors and five global markets and found that every single one had at least one security flaw.

And just because it's on the Google Play store doesn't automatically mean it's safe. In fact, applications on Google Play frequently include security flaws that can result in intellectual property theft, revenue loss, loss of trust, fraud, and data theft.

# Security Risks Android Developers Need to *Know* About

Hackers use a variety of methods to gain access to Android applications. First, they might reverse - engineer your source code so they can better understand how the application works. Unless your application is open source, there is no good reason to leave the source code visible. And unless you're hardening your applications, the source code is up for grabs.

Tools like JD-GUI, AndroChef, APKTool, and Fernflower are easy to use and readily available to any hacker who wants to reverse engineer your Android applications. Although the tools aren't nefarious, hackers can use them to reverse-engineer the source code. That allows them to use debugging tools like Eclipse or Android Studio to examine the code thoroughly and look for ways to exploit it.

# Poor Security — *What Could Happen?*

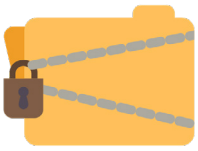
A 2021 inquiry by Atlas VPN found that 63% of Android applications had security vulnerabilities, with an average of 39 vulnerabilities per app. You might wonder, "Okay, so what? Some hackers see how the app works. What are they going to do with that?" Well, unfortunately, there is a lot they can do. Let's look at the possible outcomes if a hacker gets their hands on application source code.

## 1. Release a Modified Version of the Application

A hacker could modify the code to disable the security and validation controls, bypass licensing restrictions, insert malware, and change purchasing requirements or ad displays in the app. The app would look and function the same in every other capacity except that the cracked version could have licensing and validation checks removed or malware inserted to steal passwords. If the hacker can post the application on a public marketplace and trick users into downloading and using it, they could end up with untold amounts of private data.

This is what happened earlier this year when a fake Android app stole over a million Facebook users' account information in a massive breach.

## 2. Obtain Sensitive Information



Source code can reveal the presence of hardcoded credentials and data encryption conventions (or lack thereof). Hackers can view the application's strings, passwords, and data utilization. Hackers may also be able to view how the application communicates with a server and intercept or listen to those communications. These provide clues and information that the hacker can use against the application.

Think no developer would ever hardcode sensitive info? Think again. Research by Cybernews shows that over half of the 30,000 apps they investigated have hard-coded secrets that could provide access to API keys, Google Storage buckets, and unprotected databases.

## 3. Steal Intellectual Property

Whether you're building applications for release on the market, as part of a financial or manufacturing business, or for internal use, the software is likely to contain intellectual property and trade secrets. A hacker or competing organization can steal your technological innovations simply by reviewing your source code. Although from a legal perspective, patents, copyright protection, and trade secrets can protect intellectual property in your code, prevention is much easier than reparation.

### 4. Spy On the Application's Communication

Suppose an application runs in a compromised environment, such as a jailbroken or rooted device. In that case, it may be possible to spy on its communication without reverse engineering or tampering with the app itself. In other words, your app could be secure, but if the environment isn't — the app and its data may still be at risk.

# What Can Developers *Do* to Create Secure Android Apps?

Creating Android apps that are helpful to users and advantageous to businesses but impervious to hackers is attainable. Still, software development teams must be proactive about using best practices and programming tools while following a layered approach to security that includes protecting, testing, and monitoring.

**Protect** — Mobile application protection solutions include code hardening techniques such as obfuscation, encryption, and runtime application self-protection to stop hackers from viewing and gaining access to source code.

**Test** — Test throughout the software development lifecycle to detect vulnerabilities as soon as they are introduced so they can be mitigated as quickly as possible.

**Monitor** — Protection doesn't stop after deployment. Monitoring mobile apps for security threats in real-time can ensure integrity is maintained.

## Protect Android Apps With Security Tools

Security automation tools can integrate security directly into the software development life cycle and provide multiple layers of protection quickly and automatically. Developers use common strategies to ensure their source code and applications are safe.

## Renaming Obfuscation

Renaming obfuscation makes source code unreadable by altering the names of methods and variables to make it difficult to understand. After obfuscation, the logic, while not destroyed, is beyond comprehension. The following example illustrates obfuscation in action:

**Original Source Code Before Obfuscation**

```
1  private void CalcPayroll
2  (SpecialList employeeGroup) {
3
4     while(employeeGroup.HasMore()) {
5
6        employee =
7  employeeGroup.GetNext(true);
8
9        employee.UpdateSalary();
10
11       DistributeCheck(employee);
12
13    }
14
15 }
16
17
18
19
20
```

**Reverse-Engineered Source Code After Obfuscation**

```
1  private void a(a b) {
2
3     while (b.a()) {
4
5        a = b.a(true);
6
7        a.a();
8
9        a(a);
10
11    }
12
13 }
14
15
16
17
18
19
20
```

## Control Flow

Traditional control flow obfuscation introduces false conditional statements and other misleading constructs to confuse and break decompilers. This process synthesizes branching, dependent, and iterative constructs that produce valid forward (executable) logic but yield non-deterministic semantic results when decompilation is attempted. Control Flow obfuscation has spaghetti logic that can be difficult for a cracker to analyze.

In addition to adding code constructs, advanced control flow obfuscation will destroy the code patterns that decompilers use to recreate source code. The resulting code is semantically equivalent to the original but contains no clues about how the code was written. Even if advanced decompilers are developed, the output will be guesswork.

**Original Source Code Before Control Flow Obfuscation**

```
1  public int CompareTo(Object o) {
2
3     int n = occurrences -
4
5     ((WordOccurrence)o).occurrences;
6
7     if (n == 0) {
8
9        n = String.Compare(word,
10 ((WordOccurrence)o).word);
11
12    }
13
14    return(n);
15
16 }
17
18
19
20
```

**Reverse-Engineered Source Code After Control Flow Obfuscation**

```
1  public virtual int _a(Object A_0) {
2
3     int local0;
4
5     int local1;
6
7     local 10 = this.a - (c) A_0.a;
8
9     if (local0 != 0) goto i0;
10
11    while (true) {
12
13       return local1;
14
15       i0: local1 = local10;
16
17    }
18
19    i1: local10 =
20 System.String.Compare(this.b, (c)
21 A_0.b);
22
23    goto i0;
24
25 }
```

## String and Resource Encryption

A common technique is to locate critical code sections by looking for string references inside the binary. For example, if your application is time-locked, it may display a message when the timeout expires. Attackers search for this message inside the disassembled or decompiled output; chances are, when they find it, they will be very close to your sensitive time lock algorithm. String encryption protects these sensitive parts of your application and the system it's running, providing an effective barrier against attack.

# Enhance Security With Real-Time Defense

When you consider that nearly three billion devices are running Android, there are many untested fields for something to go awry. And that's why your applications need active, real-time defenses. These make sure that your app is running in a friendly environment; if it's not, it can shut down, throw an exception, or respond in another way. Make sure *your security tool* has the following:

## Anti-Debugger Detection

Anti-debugger detection checks can determine if debugging or other tampering is being attempted on your app and exit the app or respond in another pre-specified way.

## Shelf Life

Shelf Life is an application inventory management function that embeds an application's expiration, deactivation, and notification logic. Users can schedule an application's expiration/deactivation for a specific date and optionally issue warnings to users that the application will expire/deactivate within a particular number of days.

## Tamper Defense

Verify application integrity at runtime, and if tampering is detected, shut down the application. You can also invoke random crashes to disguise that the crash resulted from a tamper check.

## Watermarking

Watermarking tracks unauthorized copies of software by embedding data like copyright information or unique ID numbers.


## Root Check

Rooted devices are an essential security concern for Android applications. A root check determines when an app runs on a rooted device, either offline or online, and can abort the session, quarantine the application, or report the incident.

## Emulator Check

Emulator detection can tell your application to shut down or react otherwise when it is determined that it's being run within an emulator.

## Hooking Check

Hackers use hooking frameworks to inspect and change applications at runtime without modifying the binaries. A hooking check detects the presence of hooking frameworks in the environment. The app can respond by shutting down, throwing an exception, hanging, or sending an alert.

# Security *Begins* With Coding Best Practices

And while security tools can be beneficial, don't underestimate the value of putting app security best practices into play when writing code. According to the Android developer's website, that means:

### Enforcing Secure Communication

- Safeguard communication between apps
- Ask for credentials before displaying sensitive information
- Apply network security measures
- Use WebView objects with care

### Providing the Right Permissions

- Use intents to defer permissions
- Share data securely across apps

### Storing Data Safely

- Store private data within internal storage
- Store data in external storage based on the use case
- Store only non-sensitive data in cache files
- Use SharedPreferences in private mode

### Keeping Services and Dependencies Up to Date

- Check the Google Play services security provider
- Update app dependencies

# *Strengthen* Your Defenses With PreEmptive

Through prevention, detection, and response, PreEmptive secures and defends intellectual property, code integrity, and ultimately reputation and revenue against piracy, counterfeiting, and tampering. Our layered security and obfuscation protection are infused into Android applications to reduce risks.

## Protecting Your Android Applications – With PreEmptive

OWASP recommendations and Android developer best practices are essential in your software development lifecycle process. Still, they need to be paired with a software development tool that protects, tests, monitors, and defends against all forms of mobile app threats, including reverse engineering, static and dynamic analysis, and source code tampering.

Application hardening and layered security measures are critical to Android application development. PreEmptive Protection applies a layered approach to binary code protection using multiple forms of obfuscation, encryption, root detection, shielding, and tamper-proofing to secure Android apps against piracy, malware injection, tampering, and a host of other application and data-related risks. Attacking an app becomes nearly impossible. Whenever a hacker attacks one layer of the protection, another layer stops them. And, because security is infused into your application, the protection goes wherever your mobile app goes.

Over 5,000 manufacturing, life science, aerospace, financial, and software organizations trust PreEmptive to secure and measure their work without compromising the quality and functionality of their code.

Do you want to see how PreEmptive helps organizations worldwide protect their android apps so you can avoid being the next data breach headline? Contact us for a demonstration to learn how PreEmptive can help.

***Smart App Protection** for an Unsafe World**!***

## GET IN TOUCH:


**Headquarters in USA**
10801 N Mopac Expressway
Building 1, Suite 100
Austin, TX, 78759
phone: **+1 (512) 226-8080**
email: **solutions@preemptive.com**

**European Sales**
140 bis rue de Rennes
75006 Paris
France
Tel: **+33 01.83.64.34.74**
email: **EuroSolutions@preemptive.com**

**JAPAN**
AG-Tech Corp
Tel: **+81-3-3293-5300**
Email: **info@agtech.co.jp**