A Security Guide for Kotlin Developers

kiuwan
An Idera, Inc. Company

INDEX

Overview

A pragmatic, modern, and statically typed coding language that's essentially a Java alternative, **Kotlin** offers some key benefits for Java Virtual Machine (JVM) and Android app development while also being interoperable alongside Java. Kotlin language is general purpose and open sourced—available on GitHub for anyone to use—combining functional and object-oriented programming features. It has, so far, proven attractive to over 60% of professional mobile app developers for Android OS, as it helps boost productivity and developer satisfaction through improved code safety.

Google officially announced Kotlin support for Android in 2017. So, the programming language joins the likes of Java and C++. It started being built into the Android development toolset starting with Android Studio 3.0, and a plug-in allows it to be added to earlier versions.

Kotlin is meant to be expressive and concise, relying less on boilerplate code to program. Its inclusion of @Nullable and @NonNull types helps avoid inadvertent NullPointerExceptions that plague Java developers, resulting in code that's 20% less likely to crash. Another advantage of the Kotlin language is its structured concurrency: its coroutines make asynchronous programming more streamlined.

Kotlin has been used for some of the most commonly used apps, including Slack, Reddit, and American Express. Being an already popular and still-rising programming language, this guide aims to arm Kotlin developers and other key decision makers in software security and software supply chain vulnerabilities with information regarding the top security risks they can expect to face — from inherent weaknesses to potential attack vectors for data breaches.

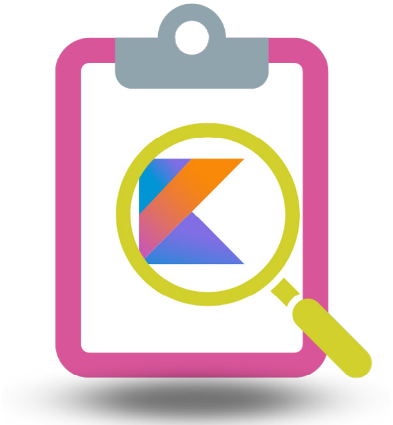
This Kotlin security guide will explore the following topics and top common risks:

- Kotlin's Security Profile
- Most Common Security Attacks
- Improper Control of Resources Through Their Lifetimes
- Not Adhering to Coding Standards
- Improper Checking or Handling of Exceptional Conditions
- Failure of Protection Mechanisms
- Inherent Weaknesses in the Programming Language
- OWASP Mobile TOP 10 Mobile Risks

Kotlin's *Security* Profile

Being broadly **a step up from Java**, Kotlin is fairly more stable and secure. Its strengths mostly lie in null safety, which leads to fewer crashes by avoiding NullPointerException errors. While this feature is geared more towards stability, it does indirectly improve security: the less errors and logs need to be stored, the less possible attack vectors exist. The language also allows developers to specify variables as mutable or immutable via val and var declarations, an inherent feature that makes coding in Kotlin generally more secure.

In terms of features that directly impact security, Kotlin uses solid libraries with decent data encryption that makes it easier for Kotlin developers to encrypt data transmission. Like many modern coding languages, Kotlin strives to continuously update its list of known vulnerabilities, releasing applicable patches as soon as possible.



Of course, the team behind Kotlin **recommends a handful of simple best practices** to ensure application and data security when coding in the language:

For input validation, developers should have these checks in place:

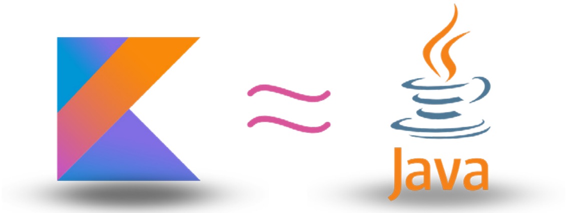
1. *Always use* the latest Kotlin release.
2. *Always use* the latest versions of Kotlin's dependencies, keeping a close eye on new vulnerabilities for the dependencies you use.
3. *Always proactively provide* feedback and report on security issues found through official channels.

Application Security for Kotlin Language

Not only will application security be different based on the programming language used, its various facets will depend on which parts of the code are relevant. Software security in JVM, for instance, is separate from (Application Programming Interface) API level security. There are also security vulnerabilities inherent in (and therefore unique to) Kotlin itself that bad faith actors can take advantage of to crash services or steal data.

While the team behind Kotlin performs their part of the application security equation, you should also do yours as a Kotlin developer. This *Kotlin security guide* is a good starting point.

Similarities between Kotlin and Java



As Kotlin is essentially a Java alternative, research has found that the **similarities between the two** have resulted in a few trends:

- The same types of security weaknesses appear to be similarly distributed in mobile apps developed in Kotlin and Java.
- The findings from empirical studies performed on apps developed in Java can be generalized to apps developed in Kotlin.
- One of the most significant security risks is present in both Kotlin and Java: improper control of resources throughout their lifetime.
- Android developers are generally concerned about privacy and confidentiality factors for both programming languages.

This bodes well for developer teams only starting to transition to Kotlin from Java, as many of the high level principles and policies they're already used to will still apply.

Of course, if this is the case for you, it would be best to carefully consider how you can guarantee your developers transition to proper coding practices specific to Kotlin, which means paying attention to the differences between Kotlin and its predecessor Java.

Most Common Security *Attacks*

Before delving into the top security risks for Kotlin, it would be wise to take a peek at the most common attacks that result from those risks. Below are some of the most common security attacks Kotlin developers experience in their apps:

SQL (Structured Query Language) injection - SQL queries can be manipulated by an attacker to expose vulnerabilities in apps developed in Kotlin. SQL injection is a veritable classic when it comes to security attacks, and is an ever-present danger especially for lazy programming. A typical format of SQL injection is manipulating the value for an ID in a URL of an application through user inputs. The usual way to prevent SQL injection is validating user input before sending it through as SQL queries, or using PreparedStatement.

Cross-site Scripting (XSS) - A type of security vulnerability where attackers inject malicious JavaScript snippets into applications through various means, including user inputs. Malicious snippets may be entered into an app's code through reply features, and then executed when a regular user loads the code that contains the snippets. This is why user input validation and sanitization is always the best practice.

Command Injection - A security attack conducted by injecting malicious code into the server. If unchecked, the backend runs the malicious snippets like regular code, potentially allowing the attacker access to server resources and data. Performed via user input, command injection attackers can use URL query parameters to inject their malicious code. Again, input validation and sanitization can prevent command injection, as can avoiding the use of functions executing commands in the code in the first place.

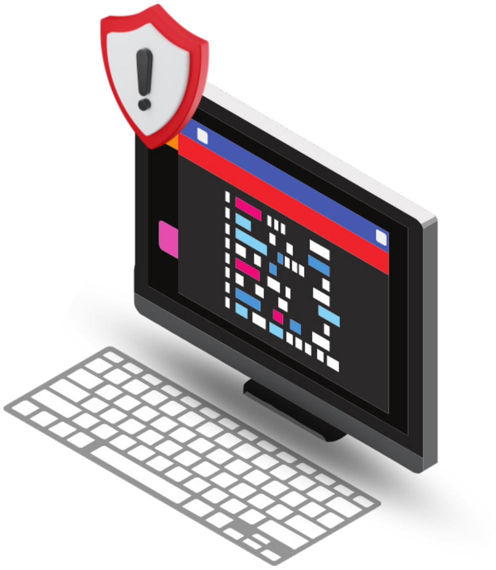
Cross-Site Request Forgery (CSRF) - Apps with authenticated users can be duped into executing unauthorized, malicious commands on behalf of the trusted users. The attack vector for this security vulnerability is the authenticated users themselves. They can be tricked into clicking manipulated URLs, for example. CSRF attacks can are thwarted by verifying authentication before executing a command.

HTTP Strict Transport Security Header - Essentially anytime a web app uses HTTP instead of HTTPS, it's using a nonsecure protocol that can more easily expose data when transmitted. Not standardizing to HTTPS opens up a lot of encryption-related concerns that are simply fixed with a policy switch to the strict use of the protocol.

The principles, standards, and practices in other programming languages where these attacks also occur are generally similar to what you would apply for Kotlin. The difference is in the nuances of the programming language. Take note that all of these common attacks actually take advantage of human error — be it through targeting unsuspecting users or via negligent coding.

One of the pillars of application security is practicing defensive programming and adhering to guidelines laid out in internal company policy and best practices ascribed by the people behind the specific programming language you're using.

Top Kotlin *Security Risk #1*: Improper Control of Resources Through Their Lifetimes



Developers that either do not maintain or incorrectly maintain control over resources throughout their lifetimes (from creation to use to release) can open up an application to exploitation. In Kotlin (and Java), concurrency issues are among the most prominent security risks that fall under this category. Concurrency issues can easily open up even with simple mistakes, like using the collection type mutableMapOf instead of ConcurrentHashMap.

Another common security vulnerability that falls under this category is exposing resources to wrong spheres, i.e., when resources are inadvertently exposed due to unexpected execution scenarios or insecure permissions.

To avoid concurrency issues, it's integral for development teams to standardize coding policy to strictly follow such rules like using ConcurrentHashMap only, wherever possible. In the case of inadvertently exposing resources, developer teams can implement the strict use of features like setting FLAG_SECURE to windows showing sensitive information. Flags like this disable screen capture capabilities when methods like ShowPassword are executed. In official Android documentation, this flag is recommended in windows containing sensitive data.

An additional recommendation to avoid this vulnerability is using keyword matching mechanisms to automatically identify code that handles sensitive data or deals with displaying windows. Developer teams can then implement the automatic addition of the right flags to the right pieces of code to minimize attack vectors.

So, for this security risk, it's a matter of diligent and defensive coding with mindful policies within the developer team as well as some manner of automatic identification, so that the quick fixes can be applied without hassle.

Top Kotlin *Security Risk #2*: Not Adhering to Coding Standards

It's no surprise that one of the most common security risks in Kotlin is rooted in **manual negligence**. Developers that ignore not only general best practices but also internal guidelines and policy relating to programming open up their teams to various security risks depending on what they're neglecting to perform. For Kotlin, one of the most prominent representations of this security risk is the presence of **irrelevant or dead code**.

These code snippets are not executed in any of the app's features and as such only serve as potential points of failure or attack. Even if dead code is not executed throughout the normal operation of an application, it can still be unintentionally invoked or tested by users or developers, resulting in unexpected behavior at least and security breaches, at worst. These snippets can be exploited by attackers, and worse, they pose a threat that persists throughout security updates to the application because they are not maintained through code updates. Moreover, they persist from Android Package Kits (APKs) after compilation.

One common example of dead code coming back to life with disastrous consequences is when the implementation of a new feature inadvertently invokes a database access method from an irrelevant code snippet. When the app is deployed, this leads to loss of information, and when detected by attackers, can be a juicy breach waiting to be exploited.

Dead or irrelevant code is easily addressed through:

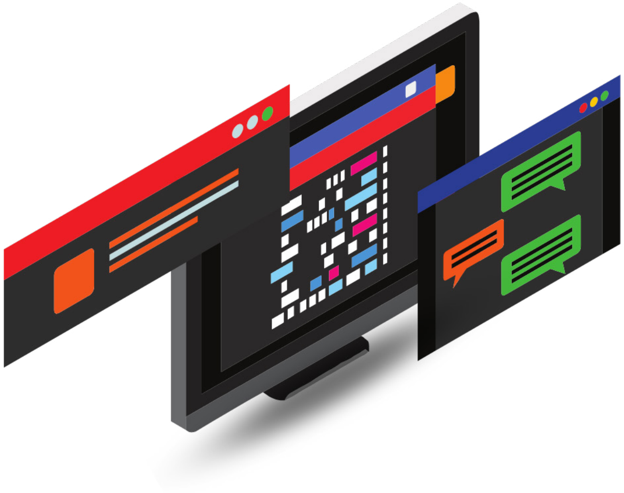
1. **Adding a comment before the snippet identifying potentially dead code and prioritizing a review, and...**

2. **Commenting out the dead code snippet so that it is not executed until properly reviewed and removed or reinstated**

Ideally, you perform both. Irrelevant code is so prevalent in applications developed in Kotlin and Java that developer teams would do well to seek out measures that can automatically identify snippets that can be removed without consequence to the application. Relying only on developers manually spotting these dead code snippets will likely result in you missing some of them, especially if your application code base is on the expansive side.

Top Kotlin *Security Risk #3*: Improper Checking or Handling of Exceptional Conditions

Exception handling is a discipline of its own in programming. So when developer teams are busy coding their app, it might be strayed to the peripherals. Improper checking or handling of exceptional situations, however, can lead to unpredictable app behavior. The worst case is you miss an exceptionally rare occurrence during normal app operation, but the code doesn't crash, and you have no logging protocols to note the issue. So, you never register something is actually wrong. That's an invisible attack vector that when exploited will catch you completely unaware and unprepared.



Improper input validation and neutralization are very common in this category. Always properly check data and messages to ensure they're valid, well-formed, and benign. Otherwise, you end up with SQL and command injection attacks.

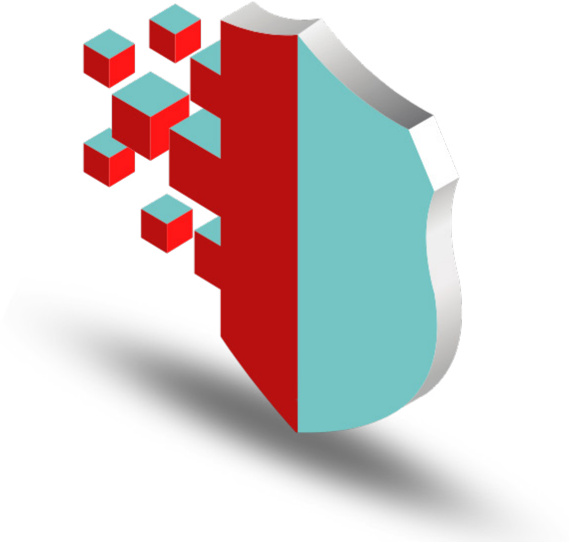
It is recommended to use fuzz testing in dynamic analysis to address this type of issue. Feeding unexpected input data in an effort to intentionally crash the app, exploit security vulnerabilities, or induce exceptional states during testing is the best way to prevent these types of attacks. Manually testing for situations you're not even aware might occur is obviously impossible in this regard.

Another occurrence that falls under this category are uncaught exceptions, which may lead to application crashes that can expose sensitive data. Uncaught exceptions are widely known issues in Android apps, particularly those that rely on abstractions. So, pay close attention to this concern, especially if it's relevant to your code base.

Top Kotlin *Security Risk #4*: Failure of Protection Mechanisms

These vulnerabilities stem from unauthorized actors being able to access resources due to incorrect restrictions. Attackers that bypass poorly coded protection mechanisms can compromise application or even server security by accessing sensitive information, gaining privileges for further attacks, and corrupting resources.

Double check authorization protocols and stress test protection mechanisms to help identify the simple programming mistakes that lead to these vulnerabilities. In many cases, it's just a matter of a missing statement in the code. Unfortunately, these issues are trickier to spot even with automated testing tools, as those are more geared towards application crashes.



Another common way for improper access control to fall into the hands of an attacker is if an app sends private tokens as parameters within GET requests. This means the token is exposed to anyone who can catch the URL, opening up the app to a Man-in-the-Middle attack, where an attacker can impersonate an authorized user with the token.

These vulnerabilities also stem from poor coding practices, such as making use of hard-coded credentials for testing purposes. Negligent post-testing data checks may also leave sensitive information in places where they can easily be exposed.

As with many other instances, in terms of using protection mechanisms, prevention is better than cure: always espouse diligent and defensive programming in your development team.

Top Kotlin Security Risk #5: Inherent Weaknesses in the Programming Language

Lastly, there are inherent weaknesses in the Kotlin language that need to be compensated for through effective coding policy, like with every programming language in existence. Some of Kotlin's weaknesses include:

- **Double serialization**: This is where mishandling data can lead to the degradation of the information's integrity quality.
- **Lack of code obfuscation**: This makes app code susceptible to reverse engineering. Someone with enough technical know-how can make use of non-obfuscated code to reverse engineer a Kotlin-based app and gain an understanding of its inner workings, allowing them to retrieve sensitive resources and information.
- **Internet-driven APIs**: Bundling API credentials and other sensitive data in your Kotlin app risks exposing the information when transmitted through nonsecure protocols. The problem is that every app will need to connect to the internet at some point. So, APIs and API calls are practically inevitable. Kotlin does have built-in tools for addressing this issue that developers can leverage, including WebView and HttpsURLConnection. Note, however, that improper use of these tools can also result in some vulnerabilities. For example, you may inadvertently allow JavaScript execution when using WebView API, which can then enable the app to visit any URL (regardless of transmission protocol), and thus, opening up the app to data security breaches.
- **Graphical User Interface (GUI) driven information exposure**: Apps that show sensitive information in their GUIs rely only on the user's prerogative to protect their visible information from prying eyes. Even someone standing behind a mobile app user can gain access to confidential information, if it's freely available in the GUI. This is less of an issue with Kotlin and more of an issue with GUI security and mobile user experience (UX) in general, but it bears mentioning here.
- **Component hijacking**: That's where app components can be controlled by an actor to gain access and privileges that allow them to stage further attacks. This too is more of a broad category that applies to mobile apps in general and not specifically to Kotlin, but bears repeating as it's completely in the hands of the development team to avoid.

Your development team should always account for these factors inherent in the Kotlin language and take steps so that your coding standards make up for any potential technical weaknesses in the code itself.

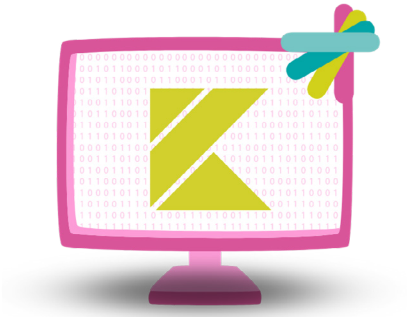
OWASP TOP 10 Mobile Risks

As Kotlin is a programming language for Android mobile apps, it's generally susceptible to the broader set of security risks that apply to all mobile apps, not just the language specifically. In 2016, the Open Web Application Security Project (OWASP) released the **top ten mobile risks compiled** from global submissions:

1. Improper Platform Usage
2. Insecure Data Storage
3. Insecure Communication
4. Insecure Authentication
5. Insufficient Cryptography
6. Insecure Authorization
7. Client Code Quality
8. Code Tampering
9. Reverse Engineering
10. Extraneous Functionality

These ten risks present slightly differently for each programming language. So, the specific language will dictate how the issues need to be addressed. For Kotlin developers, these top ten risks are worth closely following as a checklist for internal team policy and coding standards. Of course, most of these risks overlap with the top **security vulnerabilities specific to Kotlin**, as discussed above.

Kiuwan's Capabilities
Secure Kotlin Applications



While not completely exhaustive nor comprehensive, this Kotlin security guide is a good jumping off point that provides foundational data that can inform your application development policy. Aside from adopting best practices and keeping abreast of security vulnerabilities related to Kotlin and its dependencies, you can also partner with Kiuwan to secure applications with code security and analysis tools that automatically identify and remediate those vulnerabilities.

Kiuwan can automatically run your code through Kiuwan Code Security, a static application security testing (SAST) suite that is compliant with the most stringent security standards, such as OWASP and Common Weakness Enumeration (CWE). Kiuwan also provides software composition analysis (SCA) called Kiuwan Insights Open Source, which reduces risk from third-party components, helps fix vulnerabilities, and ensures license compliance and policy automation throughout your software development life cycle.

Of course, aside from being able to secure Kotlin applications, Kiuwan covers more than 30 coding languages with its security capabilities. **Book a demo** with Kiuwan today.

*YOU KNOW **CODE**, WE KNOW **SECURITY!***

GET IN TOUCH:

Headquarters
2950 N Loop Freeway W, Ste 700
Houston, TX 77092, USA

United States **+1 732 895 9870**
Asia-Pacific, Europe, Middle East and Africa **+44 1628 684407**
contact@kiuwan.com
Partnerships: **partners@kiuwan.com**