Expert Report Cover Sheet

Administrative Review Tribunal

This cover sheet should be used by an expert who is preparing a report for the purpose of a proceeding in the Administrative Review Tribunal to provide information required by the *Administrative Review Tribunal (Expert Evidence) Practice Direction 2026*.

SECTION 1 - DETAILS OF PROCEEDING

Use this section to provide details of the Tribunal proceeding for which the expert report is being provided.

Tribunal case number(s):	ART 2025/4916
Applicant:	Fraser Tweedale
Respondent:	CEO, Services Australia
Expert engaged by:	Applicant

SECTION 2 - DETAILS OF EXPERT

Use this section to provide your details as the expert providing the report.

Name:	Peter Serwylo
Professional address:	833 Collins St, Docklands, Victoria, 3008
Profession	Senior Software Engineer

SECTION 3 - CHECKLIST

Use this section to ensure you have provided the relevant information.

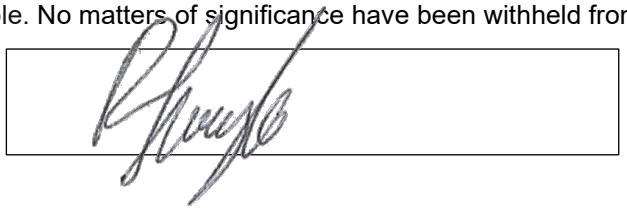
- [x] I have attached my CV or included in the report details of my qualifications and/or experience.
- [x] I have attached the letter of instruction or included in the report details of the questions or issues that I was asked to address and a reference to any documents or other materials that I was given to consider.
- [x] I have included in the report details of any facts and assumptions that inform the report and the sources for those facts and assumptions.
- [x] I have included any other relevant matters such as details of examinations, tests or other investigations that I have relied upon or details of any literature or other secondary sources that I have relied upon.
- [x] The report does not include content generated by using Generative AI.

OR

- [ ] The report includes content generated by using Generative AI and I certify that I have personally checked all the AI content and am satisfied that it is all accurate and reliable.

SECTION 4 - DECLARATION

I understand that I have an overriding duty to provide impartial assistance to the Tribunal. This report contains my independent opinion on the matters set out in it and I am satisfied that the report is accurate and reliable. No matters of significance have been withheld from the Tribunal.

Signature: 

Date: 2026-07-31

Expert Report

Tweedale and CEO, Services Australia (ART 2025/4916)

Prepared by Dr Peter Serwylo

Introduction

1. I have been a professional software developer for 20 years. This includes developing Android applications, websites, backend systems, and databases. I have a PhD in Computer Science from Monash University (awarded in 2016). I am active in the open source community, in particular contributing significantly to F-Droid, an open source alternative to the Google Play store on Android. My role with F-Droid is both a core developer and senior team member, and more recently by serving on the board of directors between 2024 and 2026. I have published many mobile apps to the Google Play store and F-Droid. I currently work at ANZ Bank as an Android developer building facets of the ANZ Plus Android app and the ANZ New Zealand goMoney app.

2. I have been asked by the applicant in these proceedings to provide an expert report on matters within my area of expertise. The letter of engagement is included as **Annexure 1**.

3. I believe that this broad range of expertise in various aspects of software development, in particular Android app development and distribution, makes me qualified to provide expert advice on the matter for determination by the Tribunal.

4. I acknowledge that I have an overriding duty to provide impartial assistance to the Tribunal.

5. No matters of significance have been withheld from the Tribunal.

Questions

1. Is the source code of the App required for a malicious actor to create a counterfeit app that would enable them to steal a victim's myGov login credentials?

6. No. Based on my experience and knowledge of application development, source code is not required to create a counterfeit app that would enable stealing a victim's myGov login credentials.

7. Typically, when developers build applications, they refer to designs to know what the resulting application should look and behave like. By definition, developers generally do not have access to the source code of an app before building it, otherwise they would not need to develop the new application.

8. In my opinion, building a counterfeit app using the original legitimately installed application as a reference is almost identical in difficulty to building a legitimate app using reference designs. Indeed, using publicly available screenshots of an original application would allow a developer to build a counterfeit application with high levels of accuracy.

9. In relation to the myGov Code Generator app, which may rely on a secret key, if a developer wanted to develop a counterfeit version including the secret key, the source code is not required to obtain the key. Android applications can be trivially decompiled and the secret key material extracted. Even in applications where the source code has been intentionally obfuscated prior to distribution, the security key will not be sufficiently hidden from a developer producing a counterfeit application.

10. If a malicious actor wished to develop a counterfeit app that behaved differently to the original app in order to be particularly malicious - for example by asking the user to enter their myGov credentials - then access to the source code of the original app is not required and will not make developing this additional malicious behaviour any easier. Such a modification represents a new feature not present in the original app, and therefore requires new source code in order to implement it.

2. Would access to the App's source code give a malicious actor an advantage in creating a counterfeit app? If so, to what degree?

11. It would in so much as it would reduce the time taken to produce a counterfeit app. However, as per my answer to Question 1, it is by no means necessary for a malicious actor to create a counterfeit app.

12. In relation to the myGov Code Generator app, access to the source code is likely to provide only limited assistance to a malicious actor creating a counterfeit version. This is because code generator apps such as the myGov Code Generator app do not typically have a large set of features to implement. Furthermore, the features are quite similar among different code generator apps, many of which are open source and have their source code freely and legally available (e.g. Aegis (**Annexure 25)**, FreeOTP (**Annexure 26)**).

13. Indeed, now that the myGov Code Generator app has been deprecated, one of the recommended authenticator apps to use is Google Authenticator **(Annexures 3, 4)**. Google Authenticator, along with many other key Android applications such as the phone dialer, SMS app, and camera, was originally distributed as open source (**Annexure 16)** by Google, the developer of the Android system. The source code for a previous version of Google Authenticator is still available (**Annexure 16)** for people to use. As with the phone dialer, SMS app, and camera, newer versions of the Google Authenticator app are no longer open source, as Google has made business decisions to have more control over the Android ecosystem. This change was not made for security reasons. While the Authenticator app remained open source, it remained secure for users, as:

> "In general, both FOSS and proprietary systems are roughly equivalent in terms of security and reliability. Neither is inherently more secure or reliable than the other." (**Annexure 32)**

14. Developers wishing to counterfeit larger, more complex applications (e.g. social networks, banking apps, or web browsers) are at a larger disadvantage if they don't have access to the original app's source code. Due to the limited feature set in the myGov Code Generator app, and the similarity to other open source code generator apps, access to the source code would only provide limited benefit.

15. The benefit afforded by access to the source code has been significantly diminished with the advent of generative AI. Developers can create counterfeit applications from screenshots alone (**Annexure 27)**, without access to original source code.

3. What mechanisms currently exist for the distribution of counterfeit apps to users?

16. The mechanisms to distribute counterfeit apps to users are the same as the mechanisms to distribute legitimate apps. The dominant distribution channel on Android devices is Google Play. This is because "the Play Store app comes pre-installed on Android devices" (**Annexure 19)**. There are other rarely used app stores available in Australia including the Amazon App Store, Samsung Galaxy Store, and F-Droid. By any measure, none of these come close to the popularity of Google Play.

17. **Amazon App Store:** This is the closest in terms of scale, but it discontinued support for Android in 2025 (**Annexure 8)** and so cannot be used to distribute counterfeit apps to Android users.

18. **Samsung Galaxy Store:** This comes pre-installed on all Samsung Android phones. Given this store does not publish download figures, it is hard to measure with certainty how popular it is compared to Google Play. However, one can compare the number of reviews for popular apps between the two platforms to infer the popularity of the app store. One such example, Microsoft Excel, has over 1 Billion downloads in the Google Play Store, also has over 9 million reviews (**Annexure 22)**. In the Samsung Galaxy Store, the same app only has 384 reviews at time of writing (**Annexure 23)**. From this, it can be inferred that only a tiny fraction of people use the Samsung Galaxy Store to obtain apps, compared to the Google Play store. Therefore it is not a particularly viable distribution channel for counterfeit apps.

19. **Installing apps from "Unknown Sources":** Unlike Apple iOS devices, users of Android have the option of directly downloading and installing applications to their phone (**Annexure 9)**. For this mechanism to be used to distribute counterfeit apps to users, the distributor of the counterfeit app must first distribute the malicious content to the user via a non-traditional mechanism, such as by encouraging them to download from a website or open from an email attachment. If this occurs, the user is warned that they are installing from an "Unknown Source" (**Annexure 10)**.

20. **F-Droid:** F-Droid is a catalogue of open source software that is built and then distributed to Android users (**Annexure 14).** There are less than 20 apps in F-Droid that had over 1 million downloads over the past year (**Annexure 28).** Google Play has over 50,000 apps which have earned over 1 million downloads (**Annexure 29).** Furthermore, it is unlikely for a counterfeit government app to be included due to the nature of its inclusion policy (**Annexure 15)** and its open and transparent publication process (**Annexure 30)**. The inclusion policy requires, among other things:

> "Applications must not infringe third party rights, including third party intellectual property rights such as copyright and trade marks. Applications need to obtain necessary trademark and copyright permissions. Any application with a duplicate Android application ID matching another domain name will receive automatic rejection." (**Annexure 15)**

21. Based on the popularity and prevalence of Google Play, the transparency around F-Droid's inclusion process, and the difficulty of a malicious developer distributing an app via websites/email and having users trust "Unknown Sources", it is my opinion that the Google Play store is still the primary mechanism for distributing counterfeit applications.

4. What mechanisms are in place in the official Google Play Store to prevent the admission of counterfeit or malicious apps, or to evict such apps if discovered?

22. Whether an app is counterfeit or not is decided according to violation of relevant policies during app distribution. There are two primary mechanisms Google Play uses to prevent the admission of, or evict counterfeit or malicious apps. The first is based on Google Play policies around "Impersonation" (**Annexure 6)**. The second is based on Google's enforcement of intellectual property laws (**Annexure 7)**.

23. The impersonation policy explicitly states:

> "We don't allow apps that mislead users by impersonating someone else (for example, another developer, company, entity) or another app. Don't imply that your app is related to or authorized by someone that it isn't. Be careful not to use app icons, descriptions, titles, or in-app elements that could mislead users about your app's relationship to someone else or another app." (**Annexure 6)**

Including the explicit example of:

> "...using a national emblem and misleading users into believing it is affiliated with government." (**Annexure 6)**

24. When developers upload apps to the Google Play store with the intent of them being made available for Android users to download and install, they must declare that they are not in violation of any of these policies.

25. Once submitted, apps are reviewed by Google for violations of the policy, and if any are identified, then the publication of the app is blocked. Counterfeit apps fall foul of both the Impersonation policy, and also the Intellectual Property policy in that they would likely try to make use of the original app owner's trademarks (i.e. the myGov or the Services Australia trademark) (**Annexure 2)** in order to deceive users.

26. If a counterfeit app progresses past the review process and gets published to the Google Play store, there is still recourse for the legitimate author of the original app and also regular Android users to report any counterfeit apps identified. This results in the app being removed from the Google Play store, and the developer who uploaded the app potentially being banned from Google Play (e.g. if they are a repeat offender). In one publicly documented example, developers of a large community developed app noticed an app published to the Google Play store which was in violation of the Google Play policies. After reporting it to Google, it was subsequently removed from the Play Store (**Annexure 24)**.

27. The reporting procedure is outlined in more detail in the answer to question 6 (below).

28. In the report "Keeping Google Play & Android app ecosystem safe in 2025" (**Annexure 5)**, Google shared that they run over 10,000 checks on every app submitted to Google Play for publication, resulting in the prevention of over 1.75 million policy violating apps from being published on Google Play.

5. What protections are in place within established app distribution systems or the Android system itself to protect against the installation of counterfeit apps?

29. As outlined above, Google Play is the dominant app distribution system on Android. Google Play has processes to protect against the installation of counterfeit apps. All Android devices come with a service called Google Play Protect (**Annexure 21)** preinstalled. This service checks apps before they are installed on users' devices, regardless of where the app is installed from (Google Play or any other source). If the app is found to be malware or otherwise violates Google's "Unwanted software policy" (**Annexure 18)** the app is prevented from being installed. If a policy violating app bypasses distribution checks and gets installed on users' devices, but is subsequently found to be policy violating, then everyone who has the app installed will receive a notification on their phone about the finding, requesting that they uninstall the app for their protection (**Annexure 21)**.

30. The "Keeping Google Play & Android app ecosystem safe in 2025" (**Annexure 5)** report states that Google Play Protect's real-time scanning identified more than 27 million new malicious apps from outside Google Play. Any user who has any of these apps installed would be notified by Google Play Protect about the finding and prompted to remove the app (**Annexure 21)**.

31. Android has an additional layer of protection for users who, despite the protections above, do manage to find and attempt to install a counterfeit version of an app. Users

of legitimate apps, including the myGov Code Generator app, are prevented from being tricked into replacing it with a counterfeit version. This is enforced by a mechanism called signature verification, whereby:

> "Every app that is run on the Android platform must be signed by the developer. Apps that attempt to install without being signed are rejected by either Google Play or the package installer on the Android device. On Google Play, app signing bridges the trust Google has with the developer and the trust the developer has with their app. Developers know their app is provided, unmodified, to the Android device; and developers can be held accountable for behaviour of their app." (**Annexure 31)**

32. Apps signed by one developer cannot be upgraded to new versions signed by a different developer. The only way in which users with the legitimate app installed can also install a counterfeit version is by either:

    1. First manually uninstalling the existing legitimate application, then initiating an install of the counterfeit version, or

    2. Installing a counterfeit version with a different identifier alongside the original version of the app. Installing a counterfeit app with a different identifier leaves the original app in place, resulting in two copies of the app appearing in the user's application list alongside each other.

6. What mechanisms exist within the Google Play Store to enable legitimate app publishers and/or users to notify or report to Google the existence of counterfeit or malicious apps?

33. Legitimate app publishers and/or users can notify or report to Google the existence of counterfeit or malicious apps via multiple channels. Counterfeit apps fall under the "Impersonation Policy" for which Google suggests if you believe another app is impersonating yours, to flag the app for review (**Annexure 20)**. This is done using the "Report an app or developer" form (**Annexure 20)** and filing a complaint of flagging an app for review. This serves the purpose to report an issue or policy violation directly to Google's team for review and potential action.

34. Counterfeit apps are typically also in violation of intellectual property rights, such as copyrights or trademark rights. The Google Play Impersonation FAQs (**Annexure 17)** state that:

> "If you believe an app on Google Play violates your intellectual property rights, such as copyrights or trademark rights, you can notify us using our reporting process for copyright, trademark, or other legal issues."

35. In my opinion, and based on my experience, these reporting mechanisms work as desired and in a timely fashion. Although the reports are normally private, I have been involved in open source projects which publicly discuss their intent to use these mechanisms to report policy violating apps (**Annexure 24)** and then confirm when

they were successful in having the violating app removed from the Google Play store. In one published discussion, Google removed the app in less than 2 weeks from the original report. (**Annexure 24)**

7. What mechanisms exist in the Android operating system to protect against an app's data from being accessed by or through other apps installed on a user's Android device?

36. Android applications are protected via the "Application Sandbox": (**Annexure 11)**

> This "isolates apps from each other and protects apps and the system from malicious apps" and ensures that "by default, apps can't interact with each other and have limited access to the [OS] (Operating System). If app A tries to do something malicious, such as read app B's data […] without permission, it's prevented from doing so because it doesn't have the appropriate default user privileges".

37. When apps store data on a user's device, the Android documentation strongly suggests they do so using "Internal Storage" (**Annexure 12, 13)** which is private to the app, and automatically removed once the app is uninstalled.

38. Therefore, unless the myGov Code Generator app explicitly decides to share secret key material or other private data with other apps, all data remains private and inaccessible to other apps installed on the user's device.

Glossary

- **Source code:** Humans typically write software in human readable computer languages. Then, in order for it to be understood and executed by a computer, it is compiled into machine code (1's and 0's). The human readable code is referred to as "source code".
- **Decompiled:** Whereas compiling is the process of transforming human readable source code into executable code, the act of decompiling is the reverse. It refers to converting executable code back into something resembling the original source code. Typically some information is lost in this process, but skilled "reverse engineers" are still able to learn a lot about how an application behaves through this process.
- **Obfuscation:** To make the task of reverse engineering harder, the source code is often obfuscated prior to compiling. This refers to making the source code intentionally hard to understand.
- **Signature:** In computer security, a signature is a way of mathematically proving the provenance of a digital file. An app author is required to "sign" their Android apps prior to distribution. This allows Google Play and the Android operating system to mathematically prove that the author of one version of an app is the same as the author of subsequent versions. More specifically, the private key used to sign one version of an app is the same as the private key used to sign another.
- **Application Sandbox:** Sandboxing an application refers to isolating it from other applications, so that one application cannot access resources or otherwise impact the behaviour of another application.
- **Deprecated**: When an application or software feature is no longer recommended for use and either will be removed in the future, or has already been removed.

Annexure 1

Wise Law

Legal Practitioners:
EJ Wise
Founder & Principal

Ash Rozario
Practice Lead (Vic)

Adam Eldridge-Imamura
Senior Associate (Vic)

ABN 26 257 174 774

Liability limited by a scheme approved under Professional Standards Legislation

Our Ref: 18062026-ADV819

Your Ref:

30 June 2026

PRIVATE AND CONFIDENTIAL

Dr Peter Serwylo
By Email: peter@serwylo.com

Dear Dr Serwylo,

Re: Engagement as an Expert Witness in Tweedale and CEO, Services Australia (ART 2025/4916)

We act for the applicant, Mr Fraser Tweedale, in the above proceeding before the Administrative Review Tribunal (**Tribunal**).

We write to engage you as an independent expert witness in the proceeding.

Scope of Engagement

Your engagement is for the purpose of providing independent and impartial expert evidence to assist the Tribunal in determining the issues in the proceeding that fall within your area of expertise.

In particular, we request that you:

1. Review any materials provided to you, together with any materials that you consider relevant to the issues on which you are asked to assist the Tribunal in determining in the proceeding.

2. Prepare a written report addressing the questions set out below.

3. If required, prepare any supplementary report responding to additional materials, issues, or questions that may arise either before or during the proceeding.

4. Be available to confer with the parties' legal representatives concerning procedural matters relating to your evidence.

5. Be available to give oral evidence before the Tribunal and to be cross-examined on your report and/or oral evidence if required.

6. Reserve availability to attend the hearing in Melbourne on the following dates:
   - Wednesday 2 September 2026, 10am-4.30pm
   - Thursday 3 September 2026, 10am-4.30pm
   - Friday 4 September 2026, 10am-4.30pm

   We will confirm the exact date and time that you are required to attend to give evidence once the schedule of witnesses in the proceeding has been finalised.

7. Promptly notify the applicant's legal representative in writing if:

   (a) you become aware of any matter that may affect your independence, impartiality, availability, or the opinions expressed in your report; or
   (b) if you become aware of any material error or omission relating to any factual matter in your report or you change your opinion on a matter contained in your report for any reason.

8. If you have, or become aware of, any actual or perceived conflict of interest that may impact on your role as an independent and impartial witness in the proceeding, you must notify the applicant's legal representative as soon as practicable.

Paramount duty to the Tribunal

Although we are engaging you as a witness in the proceeding, your paramount duty is to impartially assist the Tribunal on matters falling within your area of expertise.

The opinions expressed in your report and in your oral evidence must be independent, objective, unbiased, and uninfluenced by the interests of any party to the proceeding. To be clear, you must not assume the role of an advocate for any party.

Background

On 13 July 2021, Mr Tweedale made a request to Services Australia under the *Freedom of Information Act 1982* (Cth) (**FOI Act**) for access to a range of documents related to the myGov Code Generator App, including the source code of the App. Mr Tweedale's request was refused by Services Australia on 13 September 2021. Services Australia affirmed its decision on internal review on 11 November 2021.

Mr Tweedale sought review of Services Australia's internal review decision by the Australian Information Commissioner (**Commissioner**). On 6 June 2025, a delegate

of the Commissioner exercised their discretion under s 54W(b) of the FOI Act to decline to review the decision upon being satisfied that it was desirable that the review of the decision be considered directly by the Tribunal.

Mr Tweedale now applies to the Tribunal for review of Services Australia's decision under the *Administrative Review Tribunal Act 2024* (Cth) (**ART Act**). In the Tribunal, Mr Tweedale's application for review is confined to the decision of Services Australia to refuse access to the Android version of the source code of the myGov Code Generator App (**App**), as the App existed at the date of his original FOI application (13 July 2021).

The CEO, Services Australia (**respondent**) argues that the decision to refuse Mr Tweedale's FOI application for access to the source code of the App should be affirmed by the Tribunal on the basis that the source code of the App is an exempt document under the FOI Act. Two grounds of exemption under the FOI Act are relied upon by the respondent – namely that the source code is a document that, if disclosed, 'would, or could reasonably be expected to':

1. 'cause damage to…the security of the Commonwealth' (s 33(a)(i) of the FOI Act); and
2. 'have a substantial adverse effect on the proper and efficient conduct of the operations of an agency' (s 47E(d) of the FOI Act).

Whether either or both of these exemptions apply to deny Mr Tweedale access to the source code of the App under the FOI Act are the issues to be determined by the Tribunal in the proceeding.

Questions for your opinion

In your report, we request that you address the following questions:

1. Is the source code of the App required for a malicious actor to create a counterfeit app that would enable them to steal a victim's myGov login credentials?
2. Would access to the App's source code give a malicious actor an advantage in creating a counterfeit app? If so, to what degree?
3. What mechanisms currently exist for the distribution of counterfeit apps to users?
4. What protections are in place within established app distribution systems or the Android system itself to protect against the installation of counterfeit apps?
5. What mechanisms are in place in the official Google Play Store to prevent the admission of counterfeit or malicious apps, or to evict such apps if discovered?
6. What mechinisms exist within the Google Play Store to enable legitimate app publishers and/or users to notify or report to Google the existence of counterfeit or malicious apps?
7. What mechanisms exist in the Android operating system to protect against an app's data from being accessed by or through other apps installed on a user's Android device?

Form of the report

The report should set out details of your area of specialised knowledge or expertise, your qualifications, and your experience as it relates to the report.

In answering the questions set out above, your report should:

(a) identify all facts, assumptions, and materials upon which any opinions you express are based, including the sources of any factual information relied upon or contained in the report;

(b) state each opinion separately and clearly;

(c) explain the reasoning process that has led you to form each opinion;

(d) provide the details of any examinations, tests or other investigations upon which you have relied in preparing the report; and

(e) provide the details of any literature, secondary sources or other material relied upon in preparing the report.

In preparing the report, if you believe that an opinion that you provide is not a concluded opinion, or you are unable to reach an opinion, you must state this in the report. If you believe that an aspect of the report may be incomplete or inaccurate without some qualification, you must state that qualification in the report. Furthermore, you must make it clear if a particular question that you have been asked to answer falls outside your area of specialised knowledge or expertise.

Any document or other material relied upon in your report to support a factual conclusion or opinion must be provided as an annexure to the report. Your report must also include this letter of engagement as an annexure.

Finally, your report must include the following declaration:

> *I acknowledge that I have an overriding duty to provide impartial assistance to the Tribunal. No matters of significance have been withheld from the Tribunal.*

Generative AI

If you use generative AI to generate any content in the report, you must:

(a) clearly identify the AI content and the application(s) used to generate the AI content; and

(b) certify that you have personally checked all the AI content (including research and other materials cited in support of that content) and that you are satisfied that it is all accurate and reliable.

Confidentiality

Any documents provided to you in the course of your role as an expert witness in the proceeding, unless already in the public domain, are confidential and are supplied solely for the purpose of preparing your expert evidence in the proceeding.

You should not disclose such documents or your report to any person without the prior consent of the applicant or the applicant's legal representative, except as required by law or by order of the Tribunal.

Acceptance of Engagement

I have enclosed a copy of the Tribunal's *Guideline on persons giving expert and opinion evidence* and the *Administrative Review Tribunal (Expert Evidence) Practice Direction 2026.*

If you are willing to accept this engagement, please sign the acknowledgement set out below and return this letter to me via JB@wiselaw.com.au.

Your sincerely,

Jason Bosland
Solicitor
Email: JB@wiselaw.com.au

Expert's Acceptance

I, Dr Peter Serwylo, accept this engagement and confirm that:

1. I have read and understood this letter.
2. I have read and understood the Tribunal's *Guideline on persons giving expert and opinion evidence* and the *Administrative Review Tribunal (Expert Evidence) Practice Direction 2026.*
3. I understand that my paramount duty is to assist the Tribunal in determining the proceeding.
4. I am not aware of any conflict of interest that would prevent me from providing expert evidence in the proceeding.

Signed: 

Date: 9 / 7 / 2026

Annexure 2

Services Australia. Copyright.
https://my.gov.au/en/about/copyright
Obtained July 27th, 2026

Copyright

If you want to share the content we've created, there are a few rules you need to follow.

On this page

Except for the Commonwealth Coat of Arms and where otherwise noted, content on this website is licenced under a Creative Commons Attribution 3.0 Australia licence ©. This is signified by either of these images:

Details of the licence terms and the full legal code are available on the Creative Commons website.

Content obtained from this website is to be attributed to Services Australia as **© Commonwealth of Australia**. Changes or adaptations made to this content must be clearly identified and abide by the other licence terms.

There may be other conditions for some content. Check the licensing information for each data set you want to use.

Use of images

Unless otherwise noted, the original owners have copyright on all images on myGov.

Use of the Coat of Arms

The Coat of Arms' terms of use are available on the Department of the Prime Minister and Cabinet website.

Brand protection

Many of Services Australia's logos, names and symbols are protected by trademark or other Commonwealth legislation.

These include Services Australia, myGov, Medicare, Centrelink and others.

Any unauthorised use of our brands may be a breach of those laws.

Annexure 3

Services Australia. Sign into mygov > use an authenticator.
https://my.gov.au/en/about/help/mygov-website/sign-in-to-mygov/use-an-authenticator
Obtained July 27th, 2026

Use an authenticator

Find out about authenticators and how to sign in to myGov using one.

On this page

Authenticators generate a code you use to sign in to your myGov account. They are a type of multifactor authentication (MFA). They can be an app or service you can download to your device.

Common ones include Google Authenticator, Apple's Passwords app and 1Password.

Before you add an authenticator

Use an authenticator you're familiar with. Check if you already use one such as Google Authenticator or Apple's Passwords.

Some authenticators can't be used with your myGov account because they don't meet the SHA256 security standard. This includes Microsoft Authenticator.

If you want to use a password manager, it will need to support authenticator codes. You can check if your password manager does in your password manager settings.

Authenticators use a time-based code. Sometimes these are called TOTP or OTP. For the code to work, the device your authenticator is on needs to accurately show your local time.

Add an authenticator

If you're downloading an authenticator, make sure it's from the Google Play store, Apple App store or a trusted service.

You should only use an authenticator on a trusted device. A trusted device is a device you own and don't share with anyone else.

If you already have myGov Code Generator connected to your account, adding a new authenticator will replace it. The myGov Code Generator app is no longer available to download from app stores. If you use it, you'll be prompted through the Security review feature to switch to a more secure sign in method or another authenticator. The myGov Code Generator app will be retired.

You can only add one authenticator to your myGov account.

Annexure 4

Services Australia. Sign into mygov > use the mygov code generator app.
https://my.gov.au/en/about/help/mygov-website/sign-in-to-mygov/use-the-mygov-code-generator-app

Obtained July 27th, 2026

Use the myGov Code Generator app

The myGov Code Generator app creates codes. You use these codes as your multifactor authentication option.

On this page

The myGov Code Generator app is no longer available to download from app stores. You can now connect other authenticators to your myGov account.

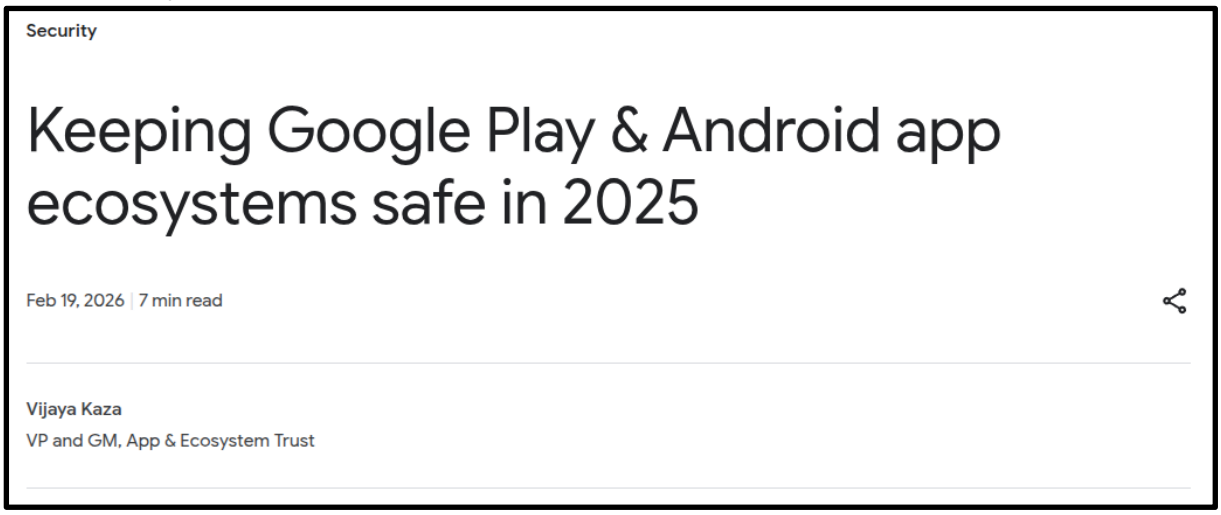
If you currently use the myGov Code Generator app, you'll be prompted through the Security review feature in your myGov account to switch to a more secure sign in method or another authenticator.

If you already have the app, you can continue using it for now. The myGov Code Generator app will be retired.

You can change your multifactor authentication option to use a code by SMS or use an authenticator.

Annexure 5

Google Security Blog. Keeping google play & android app ecosystems safe in 2025.
https://blog.google/security/keeping-google-play-android-app-ecosystem-safe-2025/
Obtained July 27th, 2026



Security

Keeping Google Play & Android app ecosystems safe in 2025

Feb 19, 2026 | 7 min read

Vijaya Kaza
VP and GM, App & Ecosystem Trust

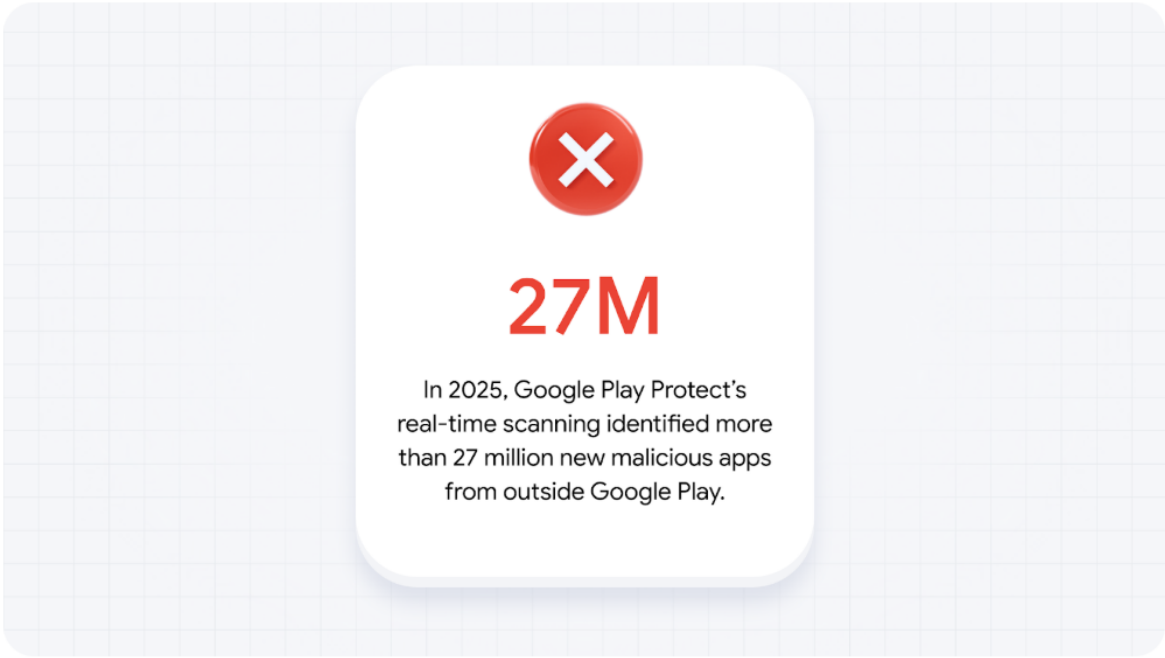
Upgrading Google Play's AI-powered, multi-layered user protections

We've seen a clear impact from these safety efforts on Google Play. In 2025, **we prevented over 1.75 million policy-violating apps from being published on Google Play** and **banned more than 80,000 bad developer accounts that attempted to publish harmful apps.** These figures demonstrate how our proactive protections and push for a more accountable ecosystem are discouraging bad actors from publishing malicious apps, while our new tools help honest developers build compliant apps more easily. Initiatives like developer verification, mandatory pre-review checks, and testing requirements have raised the bar for the Google Play ecosystem, significantly reducing the paths for bad actors to enter.

User safety is at the core of everything we build. Over the years, we've continually introduced ways to help users stay safe and make informed app choices — from parental controls to data safety transparency and app badges. We're constantly improving our policies and protections to encourage safe, high-quality apps on Google Play and stop bad actors before they cause harm.

Enhancing Google Play Protect to help keep the entire Android ecosystem safe

We also continued to improve our protections for the broader Android ecosystem, by expanding Google Play Protect and real-time security measures like in-call scam protections to help keep users safe from scams, fraud, and other threats.



As Android's built-in defense against malware and unwanted software, **Google Play Protect now scans over 350 billion Android apps daily.** This proactive protection constantly checks both Play apps and those from other sources to ensure they are not potentially harmful. And, last year, its real-time scanning capability identified **more than 27 million** new malicious apps from outside Google Play, warning users or blocking the app to neutralize the threat. To benefit from these protections, we recommend that users always keep Google Play Protect on.

While fraudsters are constantly evolving their tactics, Google Play Protect is evolving faster. Last year, we expanded:

- **Enhanced fraud protection:** Google Play Protect's enhanced fraud protection analyzes and automatically blocks the installation of apps that may abuse sensitive permissions to commit financial fraud. This protection is triggered when a user attempts to install an app from an "Internet-sideloading source" — such as a web browser or messaging app — that requests a sensitive permission. Building on the success of our initial pilot in Singapore, we expanded enhanced fraud protection to **185 markets**, now covering more than **2.8 billion Android devices**. In 2025, we **blocked 266 million risky installation attempts** and **helped protect users from 872,000 unique, high-risk applications.**
- **In-call scam protection:** We also introduced new protections to combat social engineering attacks during phone calls. This feature preemptively disables the ability to turn off Google Play Protect during phone calls, stopping bad actors from being able to trick users into disabling their device's built-in defenses to download a malicious app while on a call.

Apps on Google Play undergo rigorous reviews for safety and compliance with our policies. Last year, we shared that Google Play runs **over 10,000 safety checks** on every app we publish, and we continue to check and recheck apps after they've been published. In 2025, we continued scaling our defenses even further by:

- **Boosting AI-enhanced app detection**: We integrated Google's latest generative AI models into our review process, helping our human review team continue to find complex malicious patterns faster.
- **Preventing unnecessary access to sensitive data:** We prevented **over 255,000 apps** from getting excessive access to sensitive user data and continued to strengthen our privacy policies. Our commitment to privacy-forward app development, supported by tools like Play Policy Insights in Android Studio and Data safety section, has empowered developers to continue to: minimize privacy-sensitive permission requests, and prioritize the user in their design choices.
- **Blocking spam ratings and reviews:** Whether they lead to review inflation or deflation, spam ratings and reviews can negatively impact our users' trust and our developers' growth. We're continually evolving our detection models to help ensure app reviews are accurate. Our anti-spam protections blocked **160 million spam ratings and reviews** last year, including inflated and deflated reviews. We also **prevented an average 0.5-star rating drop** for apps targeted by review bombing, protecting our users and developers from unhelpful reviews.
- **Safeguarding kids and families**: Our approach to kids and families is built on the core belief that children deserve a safe, enriching digital environment. Our commitment is to empower parents with robust tools while providing children with access to high-quality, age-appropriate content. Last year, we announced new layers of protection, in addition to our existing safeguards, to prevent younger audiences from discovering or downloading apps involving activities like gambling or dating.

Annexure 6

Google Play Policy Centre. Impersonation.
https://support.google.com/googleplay/android-developer/answer/9888374
Obtained July 27th, 2026

Impersonation

Disclaimer: *Policy summaries and Key Considerations are overviews only; always refer to the full policy for compliance. The full policy takes precedence in case of conflict.*

Policy Summary

Google Play prohibits apps and developer accounts that impersonate or misrepresent their identity, ownership, or primary purpose. You must not coordinate with others to mislead users. Please review the full policy to ensure compliance.

Full Policy

We don't allow apps that mislead users by impersonating someone else (for example, another developer, company, entity) or another app. Don't imply that your app is related to or authorized by someone that it isn't. Be careful not to use app icons, descriptions, titles, or in-app elements that could mislead users about your app's relationship to someone else or another app.

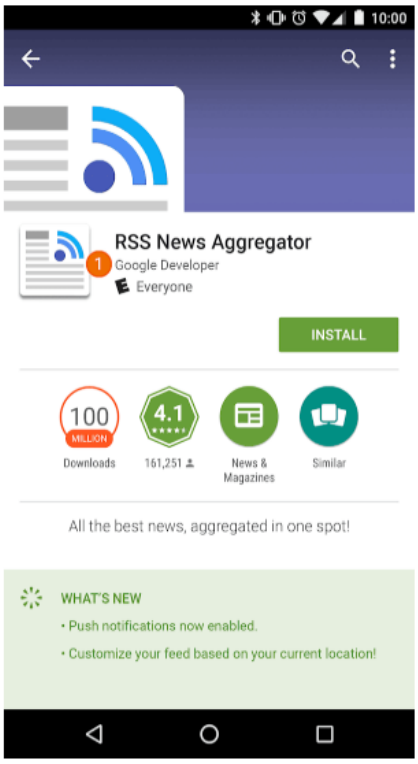
Examples of common violations

Key Considerations

Do	Don't
Accurately represent your identity as a developer and your app's purpose.	Don't conceal your identity or app's primary purpose.
Ensure your ownership details are truthful.	Don't coordinate with others to mislead users, especially for political or social content.
	Don't falsely imply an affiliation or relationship with another company, developer, entity, or organization.

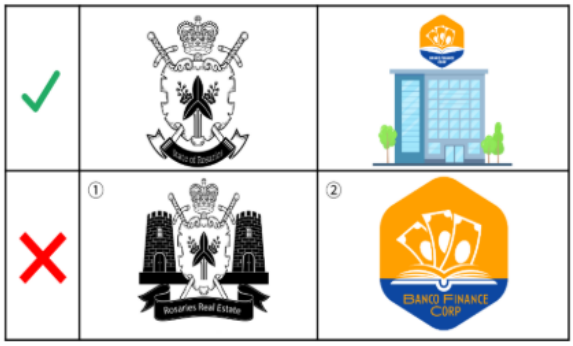
Examples of common violations

- Developers that falsely imply a relationship to another company / developer / entity / organization.



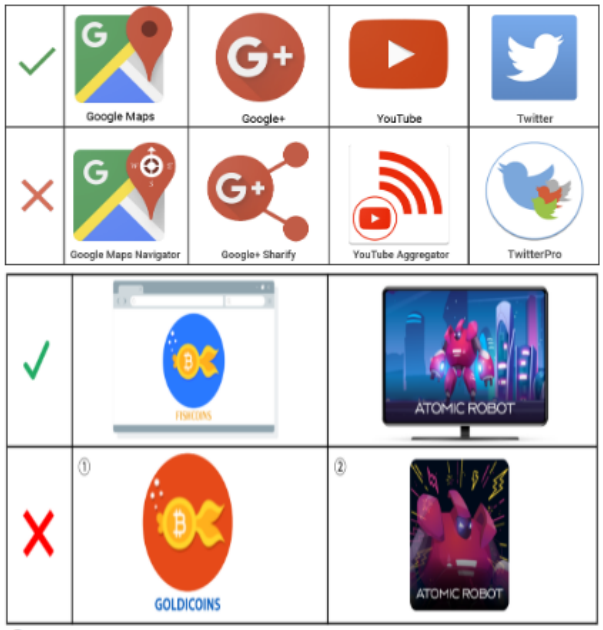
① The developer name listed for this app suggests an official relationship with Google, even though such a relationship doesn't exist.

- Apps whose icons and titles are falsely implying a relationship with another company / developer / entity / organization.



①The app is using a national emblem and misleading users into believing it is affiliated with government.
②The app is copying the logo of a business entity to falsely suggest it is an official app of the business.

- App titles and icons that are so similar to those of existing products or services that users may be misled.



①The app is using the logo of a popular cryptocurrency website in its app icon to suggest it is the official website.
②The app is copying the character and title of a famous TV show in its app icon and misleading users to think that it is affiliated with a TV show.

- Apps that falsely claim to be the official app of an established entity. Titles like "Justin Bieber Official" are not allowed without the necessary permissions or rights.
- Apps that violate the Android Brand Guidelines.

For frequently asked questions about the Impersonation policy, see this Help Center article.

Annexure 7

Google Play Policy Centre. Intellectual property.
https://support.google.com/googleplay/android-developer/answer/9888072
Obtained July 27th, 2026

Intellectual Property

Disclaimer: *Policy summaries and Key Considerations are overviews only; always refer to the full policy for compliance. The full policy takes precedence in case of conflict.*

Policy Summary

This policy prohibits apps and developer accounts from infringing on the intellectual property rights of others, including copyright, trademark, and patents. You must ensure that all content in your app and store listing is either your own original work or that you have the necessary licenses and/or permissions to use it. Please review the full policy to ensure compliance.

Full Policy

We don't allow apps or developer accounts that infringe on the intellectual property rights of others (including trademark, copyright, patent, trade secret, and other proprietary rights). We also don't allow apps that encourage or induce infringement of intellectual property rights.

We will respond to clear notices of alleged copyright infringement. For more information or to file a DMCA request, please visit our copyright procedures.

To submit a complaint regarding the sale or promotion for sale of counterfeit goods within an app, please submit a counterfeit notice.

If you are a trademark owner and you believe there is an app on Google Play that infringes on your trademark rights, we encourage you to reach out to the developer directly to resolve your concern. If you are unable to reach a resolution with the developer, please submit a trademark complaint through this form.

If you have written documentation proving that you have permission to use a third party's intellectual property in your app or store listing (such as brand names, logos and graphic assets), contact the Google Play team in advance of your submission to ensure that your app is not rejected for an intellectual property violation.

Unauthorized Use of Copyrighted Content

We don't allow apps that infringe copyright. Modifying copyrighted content may still lead to a violation. Developers may be required to provide evidence of their rights to use copyrighted content.

Please be careful when using copyrighted content to demonstrate the functionality of your app. In general, the safest approach is to create something that's original.

Examples of common violations

Encouraging Infringement of Copyright

We don't allow apps that induce or encourage copyright infringement. Before you publish your app, look for ways your app may be encouraging copyright infringement and get legal advice if necessary.

Examples of common violations

Trademark Infringement

We don't allow apps that infringe on others' trademarks. A trademark is a word, symbol, or combination that identifies the source of a good or service. Once acquired, a trademark gives the owner exclusive rights to the trademark usage with respect to certain goods or services.

Trademark infringement is improper or unauthorized use of an identical or similar trademark in a way that is likely to cause confusion as to the source of that product. If your app uses another party's trademarks in a way that is likely to cause confusion, your app may be suspended.

Counterfeit

We don't allow apps that sell or promote for sale counterfeit goods. Counterfeit goods contain a trademark or logo that is identical to or substantially indistinguishable from the trademark of another. They mimic the brand features of the product in an attempt to pass themselves off as a genuine product of the brand owner.

Key Considerations

Do	Don't
Use only original content you have created for your app and store listing.	Don't use another party's trademarks, such as logos or brand names without permission and/or in a way that could confuse users.
Obtain written documentation or a license for any third-party intellectual property you use.	Don't modify or use copyrighted content without permission.
Contact the Google Play team in advance with notice and documentation if your app uses a third party's intellectual property.	Don't create an app that encourages or facilitates the infringement of intellectual property rights (e.g. unauthorized streaming).

Annexure 8

Amazon Developer Documentation. Upcoming changes to Amazon appstore for Android devices and other programs.
https://developer.amazon.com/apps-and-games/blogs/2025/02/upcoming-changes-to-amazon-appstore-for-android-devices-and-coins-program

Obtained July 27th, 2026

Upcoming changes to Amazon Appstore for Android devices and other programs

Amazon Developer Feb 20, 2025 Share:

Announcements

In our ongoing effort to streamline and improve our services and programs, we are making some changes to Amazon Appstore for Android devices and Amazon Coins program.

We will be discontinuing support of Amazon Appstore for Android devices on August 20, 2025. As of February 20, 2025, developers will no longer have the option to submit new apps targeting Android devices. However, developers will have the option to submit updates to their existing live apps on Amazon Appstore for Android devices until August 20, 2025. With these changes, starting today, customers in JP marketplace will no longer be able to make any in-app purchases (IAPs).

The aforementioned change specifically impacts Android mobile devices only.

We will also discontinue support for Amazon Coins program on August 20, 2025 in all marketplaces. As of February 20, 2025, customers will no longer be able to buy Amazon Coins. However, customers will be able to use their previously purchased Amazon Coins until August 20, 2025.

Amazon Appstore will continue to be available and supported on Fire TV, Fire Tablet, and Fire TV built-in products. We are committed to assisting developers during this transition. Please submit a Contact Us form to our support team for any questions.

FAQs

1. **Will my apps on Amazon Appstore for Android devices be affected in any way?**
   All existing apps on Amazon Appstore for Android devices will continue to be available to customers until August 20, 2025. Developers can continue to submit app updates until August 20, 2025.

2. **Do I need to take any action?**
   No, no action is required of developers at this time. Please note that, as of February 20, 2025, you will no longer have the option to submit new apps targeting Android devices. However, you will have the option to submit updates to your live apps on Amazon Appstore for Android devices until August 20, 2025..

3. **What happens to my apps after discontinuation of Amazon Appstore for Android devices?**
   Starting August 20, 2025, any apps downloaded from the Amazon Appstore will not be guaranteed to operate on Android devices. Note that this change specifically impacts Android mobile devices only. Amazon Appstore, however, will continue to be available on Fire TVs and Fire Tablets.

4. **Can I submit any new apps to the Amazon Appstore for Android devices during this period?**
   No, as of February 20, 2025, you will no longer have the option to submit new apps targeting Android devices.

Annexure 9

Android Developer Documentation. Alternative distribution options.
https://developer.android.com/distribute/marketing-tools/alternative-distribution
Obtained July 27th, 2026

Alternative distribution options

As an open platform, Android offers choice. You can distribute your Android apps to users in any way you want, using any distribution approach or combination of approaches that meets your needs. From publishing in an app marketplace to serving your apps from a website or emailing them directly to users, you're never locked into any particular distribution platform.

The process for building and packaging your apps for distribution is the same, regardless of how you distribute them. This saves you time and lets you automate parts of the process as needed. You can read Preparing for Release for more information.

The sections below highlight some of the alternatives for distributing your apps.

Distributing through an app marketplace

Usually, to reach the broadest possible audience, you'd distribute your apps through a marketplace, such as Google Play.

Google Play is the premier marketplace for Android apps and is particularly useful if you want to distribute your apps to a large global audience. However, you can distribute your apps through any app marketplace you want or use multiple marketplaces.

Unlike other forms of distribution, Google Play allows you to use the In-app Billing service and Licensing service. The In-app Billing service makes it easy to sell in-app products like game jewels or app feature upgrades. The Licensing service helps prevent unauthorized installation and use of your apps.

Distributing your apps by email

A quick and easy way to release your apps is to send them to users by email. To do this, you prepare the app for release, attach it to an email, and send it to a user. When the user opens your email on their Android-powered device, the Android system recognizes the APK and displays an **Install Now** button in the email message. Users can install your app by touching the button. Users need to opt in for installing unknown apps if they haven't already to proceed with the installation.

Distributing apps through email is convenient if you're sending them to a few trusted users, as it provides few protections from piracy and unauthorized distribution; that is, anyone you send your apps to can simply forward them to others.

Distributing through a website

If you don't want to release your apps on a marketplace such as Google Play, you can make them available for download on your website or server, including on a private or enterprise server. To do this, first prepare your apps for release in the normal way, then host the release-ready APK files on your website and provide users with a download link. To install an app distributed in this way, users must opt-in for installing unknown apps.

Annexure 10

Android Developer Documentation. Alternative distribution options > User opt-in for installing unknown apps.
https://developer.android.com/distribute/marketing-tools/alternative-distribution#unknown-sources
Obtained July 27th, 2026

User opt-in for installing unknown apps

Android protects users from inadvertent download and install of *unknown apps*, or apps from sources other than Google Play, which is trusted. Android blocks such installs until the user opts into allowing the installation of apps from other sources. The opt-in process depends on the version of Android running on the user's device:

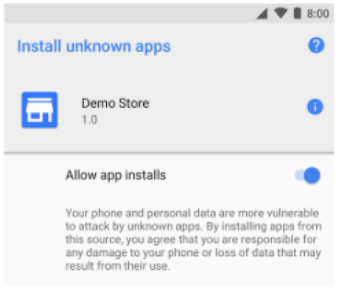


Figure 1: The *Install unknown apps* system settings screen, where users grant permission for a particular source to install unknown apps.

- On devices running Android 8.0 (API level 26) and higher, users must navigate to the *Install unknown apps* system settings screen to enable app installations from a particular location, as shown in Figure 1.
- On devices running Android 7.1.1 (API level 25) and lower, users should enable the **Unknown sources** system setting, found in **Settings > Security** on their devices.

★ **Note:** When users attempt to install an unknown app on a device running Android 7.1.1 (API level 25) or lower, the system sometimes shows a dialog that asks the user whether they want to allow only one particular unknown app to be installed. In almost all cases, users should allow only one unknown app installation at a time if the option is available to them.

In both cases, users need to complete the opt-in process *before* they can download and install unknown apps onto their devices.

★ **Note:** Some network providers don't allow users to install applications from unknown sources.

Annexure 11

Android Developer Documentation. Application sandbox.
https://source.android.com/docs/security/app-sandbox
Obtained July 27th, 2026

Application Sandbox

The Android platform takes advantage of the Linux user-based protection to identify and isolate app resources. This isolates apps from each other and protects apps and the system from malicious apps. To do this, Android assigns a unique user ID (UID) to each Android app and runs it in its own process.

Android uses the UID to set up a kernel-level Application Sandbox. The kernel enforces security between apps and the system at the process level through standard Linux facilities such as user and group IDs that are assigned to apps. By default, apps can't interact with each other and have limited access to the OS. If app A tries to do something malicious, such as read app B's data or dial the phone without permission, it's prevented from doing so because it doesn't have the appropriate default user privileges. The sandbox is simple, auditable, and based on decades-old UNIX-style user separation of processes and file permissions.

Because the Application Sandbox is in the kernel, this security model extends to both native code and OS apps. All of the software above the kernel, such as OS libraries, app framework, app runtime, and all apps, run within the Application Sandbox. On some platforms, developers are constrained to a specific development framework, set of APIs, or language. On Android, there are no restrictions on how an app can be written that are required to enforce security; in this respect, native code is as sandboxed as interpreted code.

Protections

Generally, to break out of the Application Sandbox in a properly configured device, one must compromise the security of the Linux kernel. However, similar to other security features, individual protections enforcing the app sandbox are not invulnerable, so defense-in-depth is important to prevent single vulnerabilities from leading to compromise of the OS or other apps.

Android relies on a number of protections to enforce the app sandbox. These enforcements have been introduced over time and have significantly strengthened the original UID-based discretionary access control (DAC) sandbox. Previous Android releases included the following protections:

- In Android 5.0, SELinux provided mandatory access control (MAC) separation between the system and apps. However, all third-party apps ran within the same SELinux context so inter-app isolation was primarily enforced by UID DAC.
- In Android 6.0, the SELinux sandbox was extended to isolate apps across the per-physical-user boundary. In addition, Android also set safer defaults for app data: For apps with `targetSdkVersion >= 24`, default DAC permissions on an app's home dir changed from 751 to 700. This provided safer default for private app data (although apps can override these defaults).
- In Android 8.0, all apps were set to run with a `seccomp-bpf` filter that limited the syscalls that apps were allowed to use, thus strengthening the app/kernel boundary.
- In Android 9 all nonprivileged apps with `targetSdkVersion >= 28` must run in individual SELinux sandboxes, providing MAC on a per-app basis. This protection improves app separation, prevents overriding safe defaults, and (most significantly) prevents apps from making their data world accessible.
- In Android 10 apps have a limited raw view of the filesystem, with no direct access to paths like /sdcard/DCIM. However, apps retain full raw access to their package-specific paths, as returned by any applicable methods, such as Context.getExternalFilesDir().

Guidelines for sharing files

Setting app data as world accessible is a poor security practice. Access is granted to everyone and it's not possible to limit access to only the intended recipient(s). This practice has led to information disclosure leaks and confused deputy vulnerabilities, and is a favorite target for malware that targets apps with sensitive data (such as email clients). In Android 9 and higher, sharing files this way is explicitly disallowed for apps with `targetSdkVersion>=28`.

Instead of making app data world-accessible, use the following guidelines when sharing files:

- If your app needs to share files with another app, use a content provider. Content providers share data with the proper granularity and without the many downsides of world-accessible UNIX permissions (for details, refer to Content provider basics).
- If your app has files that genuinely should be accessible to the world (such as photos), they must be media-specific (photos, videos, and audio files only) and stored using the MediaStore class. (For more details on how to add a media item, see Access media files from shared storage.)

The **Storage** runtime permission controls access to strongly-typed collections through **MediaStore**. For accessing weakly typed files such as PDFs and the MediaStore.Downloads class, apps must use intents like the `ACTION_OPEN_DOCUMENT` intent.

To enable Android 10 behavior, use the `requestLegacyExternalStorage` manifest attribute, and follow App permissions best practices.

- The manifest flag default value is `true` for apps targeting Android 9 and lower.
- The default value is false for apps targeting Android 10. To temporarily opt out of the filtered storage view in apps targeting Android 10, set the manifest flag's value to `true`.
- Using restricted permissions, the installer allowlists apps permitted for nonsandboxed storage. Nonallowlisted apps are sandboxed.

Annexure 12

Android Developer Documentation. Improve your app's security.
https://developer.android.com/privacy-and-security/security-best-practices#internal-storage
Obtained July 27th, 2026

Store private data within internal storage

Store all private user data within the device's internal storage, which is sandboxed per app. Your app doesn't need to request permission to view these files, and other apps can't access the files. As an added security measure, when the user uninstalls an app, the device deletes all files that the app saved within internal storage.

Annexure 13

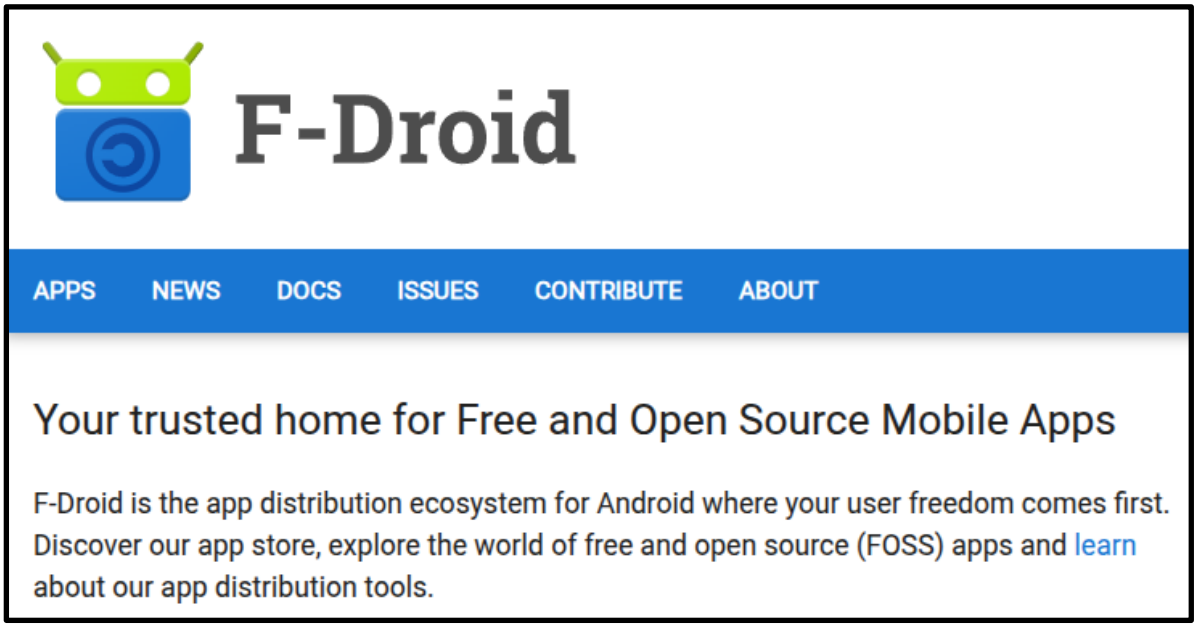
Android Developer Documentation. Security checklist.
https://developer.android.com/privacy-and-security/security-tips#internal-storage
Obtained July 27th, 2026

Internal storage

By default, files that you create on internal storage are accessible only to your app. Android implements this protection, and it's sufficient for most applications.

Avoid the deprecated `MODE_WORLD_WRITEABLE` and `MODE_WORLD_READABLE` modes for IPC files. They don't provide the ability to limit data access to particular applications, and they don't provide any control of data format. If you want to share your data with other app processes, consider using a content provider instead, which offers read and write permissions to other apps and can make dynamic permission grants on a case-by-case basis.

Annexure 14
F-Droid. Homepage.
https://f-droid.org
Obtained July 28th, 2026



Annexure 15

F-Droid. Inclusion policy.
http://f-droid.org/docs/Inclusion_Policy
Obtained July 28th, 2026

Inclusion Policy

The purpose of this inclusion policy is to ensure that all applications included in the main F-Droid repository support F-Droid's core mission: providing a trusted way to find and share privacy-respecting, free and open source software apps for Android. If an app does not meet these criteria, you can still make the app available to F-Droid users via a separate repository.

Security & Legal Compliance

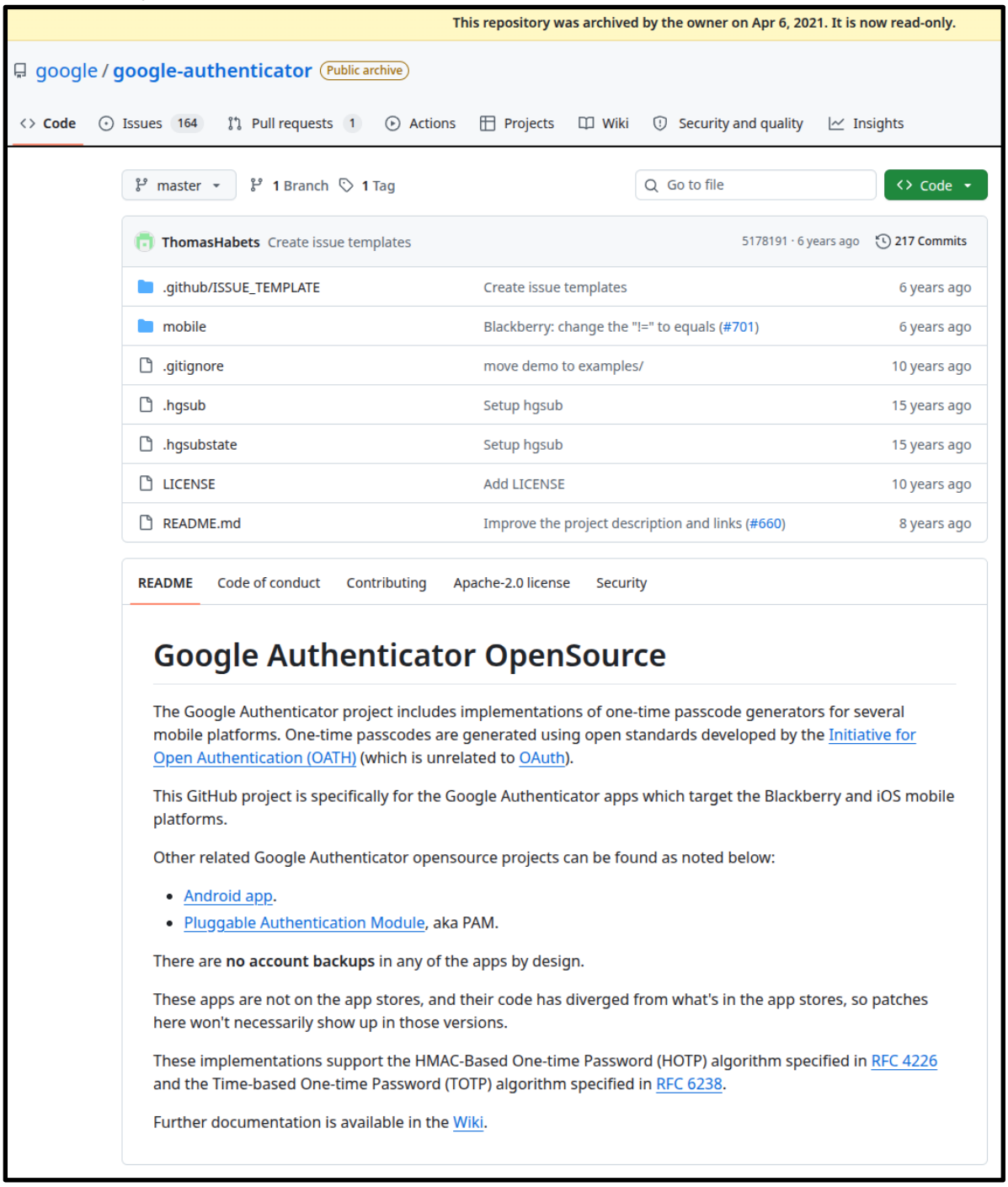
1. F-Droid does not sign up for any API keys. Even if provided by a third party, we include them in both binary and source code releases.
2. All donation links need verification by upstream developers to stop unauthorized alterations. F-Droid contributors can consider *funding.json*, *FUNDING.yml* or *README* files. Also, verification can happen via messages that come from the same account as the app's source code, e.g. `@foobar` for `https://gitlab.com/foobar/myapp`, if that account is on the same GitLab instance as F-Droid itself, e.g. `gitlab.com`. The application's main website can also host donation links.
3. Applications must not infringe third party rights, including third party intellectual property rights such as copyright and trade marks. Applications need to obtain necessary trademark and copyright permissions. Any application with a duplicate android application ID matching another domain name will receive automatic rejection.
4. F-Droid uses pre-defined labels known as Anti-Features which serve as warning indicators about user freedom, privacy or etc. without necessarily disqualifying applications from inclusion.
5. Applications must not download additional executable binary files (e.g. add-ons, auto-updates, etc.) without explicit user consent. Consent means it needs to be opt-in *(it must not be harder to decline than to accept or presented in a way users are likely to press accept without reading)* and structured in a way that clearly explains to users that they're choosing to bypass F-Droid's checks if they activate it.

Policy Enforcement

- The repository will remove or reject non-compliant applications until developers fix their issues. The F-Droid community encourages developers to work together to achieve compliance.
- Apps which fail to rebuild (such as requiring Play Services) or contain undisclosed anti-features will receive a rejection.
- Upstream developers can ask for re-review after they make necessary fixes to their applications.
- F-Droid maintains the authority to implement various actions against persistent offenders which may include warnings and temporary suspensions or permanent content removal while providing proper notification and appeal options. The documentation will record all removal actions.

Annexure 16

GitHub. Google Authenticator.
https://github.com/google/google-authenticator
Obtained July 28th, 2026



Annexure 17

Google. Impersonation FAQs.
https://support.google.com/googleplay/android-developer/answer/16341334?hl=en
Obtained July 28th, 2026

Impersonation FAQs

How do I report if I believe another app is impersonating my app?

We take impersonation seriously. You can always flag an app for review on Google Play using this process.

How do I report if I believe another app is distributing content or media I have the rights to own, without the license to do so?

If you believe an app on Google Play violates your intellectual property rights, such as copyrights or trademark rights, you can notify us using our reporting process for copyright, trademark, or other legal issues. Make sure to use the correct reporting form, as incorrect submissions may delay or prevent processing.

What is the difference between Trademark, Copyright, and Impersonation policies?

Please consult your own legal counsel to understand the differences between trademark and copyright, or to understand the scope of your intellectual property rights.

The Impersonation Policy governs how developers display apps on the Play Store and prohibits developers from misleading users about an app's connection to someone else (for example, another developer, company, entity) or another app. If you believe an app on the Play Store is misleading users in this way, but you do not own or are not asserting intellectual property rights over the content in question, you can report that to Google Play using this process.

Annexure 18

Google. Unwanted software policy.
https://www.google.com/about/unwanted-software-policy.html
Obtained July 28th, 2026

Unwanted Software Policy

At Google, we believe that if we focus on the user, all else will follow. In our Software Principles, we provide general recommendations for software that delivers a great user experience. The policy below expands upon those general recommendations by providing a list of basic criteria for user-friendly software on the web. Software that violates these principles is potentially harmful to the user experience, and we will take steps to protect users from it.

We've found that most unwanted software displays one or more of the same basic characteristics:

- It is deceptive, promising a value proposition that it does not meet.
- It tries to trick users into installing it or it piggybacks on the installation of another program.
- It doesn't tell the user about all of its principal and significant functions.
- It affects the user's system in unexpected ways.
- It is difficult to remove.
- It collects or transmits private information without the user's knowledge.
- It is bundled with other software and its presence is not disclosed.

In contrast, we believe that software that meets the basic criteria below upholds the spirit of our Software Principles and provides a good user experience. We'll continue to refine the policy as we see new use cases, and we welcome your feedback and suggestions in our help forum.

Transparent Installation and Upfront Disclosure

The software installation process should be straightforward, easy-to-understand and based on clear choices made by the user. It should present a clear value proposition to the user.

- Programs should have a valid and verified code signature issued by a code-signing authority that presents verifiable publisher information.
- Download of the software should only begin when the user has consented to the download by clicking on a clearly-labelled download button.
- At the time of installation, all principal and significant functions of the software should be described in clear and straightforward language that is clearly visible and easy to read on the screen.
- The user must have a meaningful opportunity to review and approve all principal and significant proposed installation options and system changes. For example, at the time of installation, the software might list each of the proposed settings changes, and note that the programme collects the user's personal data, with links to learn more about each of the changes.
- As part of the install flow, any bundled software must be clearly disclosed. No software should be installed silently without the user's permission. The name and principal and significant functions of every piece of software that will be installed should be visible to the user, and the user should be able to skip the entire bundled software or offer as well as individual components of the bundle.
- Before and during the install process, the software must not engage in any deceptive behaviours. Some examples of deceptive behaviours include:
  - Making false or misleading claims about the state of the user's system. For example, misleading claims related to antivirus protection, system performance, system optimisation, a new version of a plug-in, etc.
  - Claiming or implying to be official software from a company or a partner of the company if that is not the case.
  - Charging fees for software that is available elsewhere for free without disclosing this to the user and explaining what extra service justifies the fee.
  - Providing unproven or misleading endorsements.
- The software and download page must contain a link to an End User Licence Agreement (EULA) or Terms of Service (TOS).

Simple Removal

It should be easy for users to disable or uninstall software.

- Uninstall information must be easily accessible, simple to perform, and clearly identifiable after the software has been installed.
- During the uninstall process, users must be presented with clearly-labelled and prominent instructions that explain how to return their browser's and/or computer's user settings to the previous settings.
- The software must provide a clear uninstall process and must not engage in any deceptive behaviours to deter uninstallation. Some examples of deceptive behaviours include:
  - Making false or misleading claims about potential negative effects on the user's system or privacy if the software is uninstalled.
  - Charging fees for software removal.
  - Including additional prompts or offers that are unnecessary to the uninstall process.
  - Making the default option in the uninstaller to hide the software instead of removing it.
- Uninstallation must not impact unrelated files.
- Once software is disabled or deleted, the removal should be complete. Configuration changes that affected the behaviour of already existing software should be reverted. It may not keep any pieces of the software running after uninstallation and must not be automatically enabled later by itself or another programme.

Clear Behaviour

Once installed, software should behave as expected and deliver a clear value proposition to the user.

- After installation, the program should not download or install additional software, or make system settings changes, beyond what was offered during the initial installation, unless it is doing so at the explicit, informed direction of the user.
- When accessing Google services or products, software must use and adhere to the terms of publicly-available Google APIs for interacting with the user's system or any program installed. In addition, software must comply with any other applicable Google policies.
- Programs that modify a system's settings must clearly disclose what has changed and how the user can undo it.
- After installation, programs should not engage in deceptive or unexpected behaviour. Some examples of deceptive or unexpected behaviour include:
  - Displaying false or misleading messages about the system status.
  - Hiding or cloaking the software's behaviour. Behaving differently when running in a virtualised environment.
  - Impairing usability of the system. Remapping inputs, unless the program is doing so at the explicit, informed direction of the user.
  - Preventing the user from controlling the software, or interfering with control of, or access to, any other program already installed on the system.
  - Affecting the integrity of other programs, including disabling or circumventing security and protection measures, unless the program is doing so at the explicit, informed direction of the user.
  - Intercepting and redirecting network traffic, unless it is the stated purpose of the software.
- The software should not send spam. It should not inject ads, unless that is the stated purpose of the program.
- If the software makes updates, it should provide a clear notification to the user. The user must have a meaningful opportunity to review and approve any principal and significant updates or settings changes.

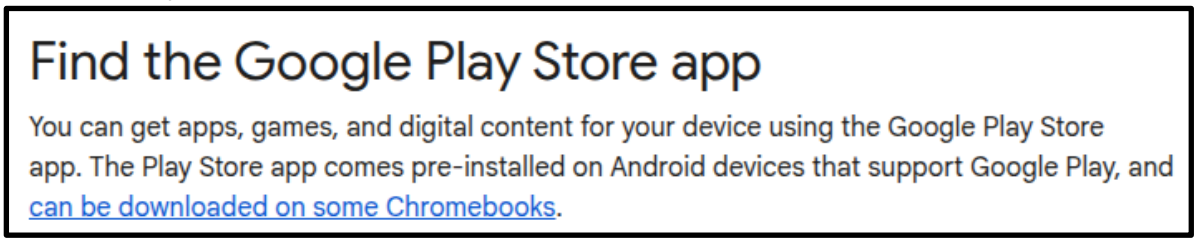
Snooping

Software that collects or transmits a user's personal information must be transparent about doing so.

- Software that collects and/or transmits users' personal information must be transparent about it by providing an explanation in clear and straightforward language that describes what information would be collected or transmitted and for what purpose. The language should be clearly visible and easy to read on the screen. Disclosure is especially important if data collection is a non-obvious feature of the software.
- Software must not collect sensitive information such as banking details without proper encryption.

Annexure 19

Google Play Help. Find the google play store app.
https://support.google.com/googleplay/answer/190860
Obtained July 28th, 2026



Find the Google Play Store app

You can get apps, games, and digital content for your device using the Google Play Store app. The Play Store app comes pre-installed on Android devices that support Google Play, and can be downloaded on some Chromebooks.

Annexure 20

Google Play Help. Flag an app or review on google play.
https://support.google.com/googleplay/answer/2853570
Obtained July 28th, 2026

Flag an app or review on Google Play

If you find an issue with an app or review on Google Play, you can flag it to our team with the options below. We take your feedback seriously, and we appreciate your help in improving the Google Play experience.

Report an app or developer

Android Computer

Provide feedback about an app

Filing a complaint or flagging an app or review:

- **Purpose:** This mechanism is used to **report an issue or policy violation directly to Google's team** for review and potential action.
- **What to report:** You would flag an app for issues beyond simply disliking it, such as violations of Google Play Developer Program Policies (e.g., developers attempting to manipulate app placement through fraudulent or incentivized reviews), or illegal content. You can also flag inappropriate comments or reviews left by other users, or developer replies that violate the Developer Comment Posting Policy.

You can flag an app on Google Play if you find an issue. Please read the options below to understand the types of issues we review.

Flag an app on Google Play

If you're unhappy with an app you've found on Google Play, you can flag it.

Flag an app

1. Open the Google Play App.
2. Go to the detail page for an app or game.
3. Tap More ⋮ > **Flag as inappropriate**.
4. Choose a reason.
5. Tap **Submit**.

Tip: To provide feedback about the app to other users, you can leave a public review on Google Play. Learn more about reviews.

Report an app violating Google Play Developer Policy

If you think an app violates one of our Developer Program Policies, you can report it.

Report an app for illegal content

If you think an app violates your rights or applicable laws, you can report it.

Annexure 21

Google Play Help. Use google play protect to help keep your apps safe & your data Private.
https://support.google.com/googleplay/answer/2812853
Obtained July 28th, 2026

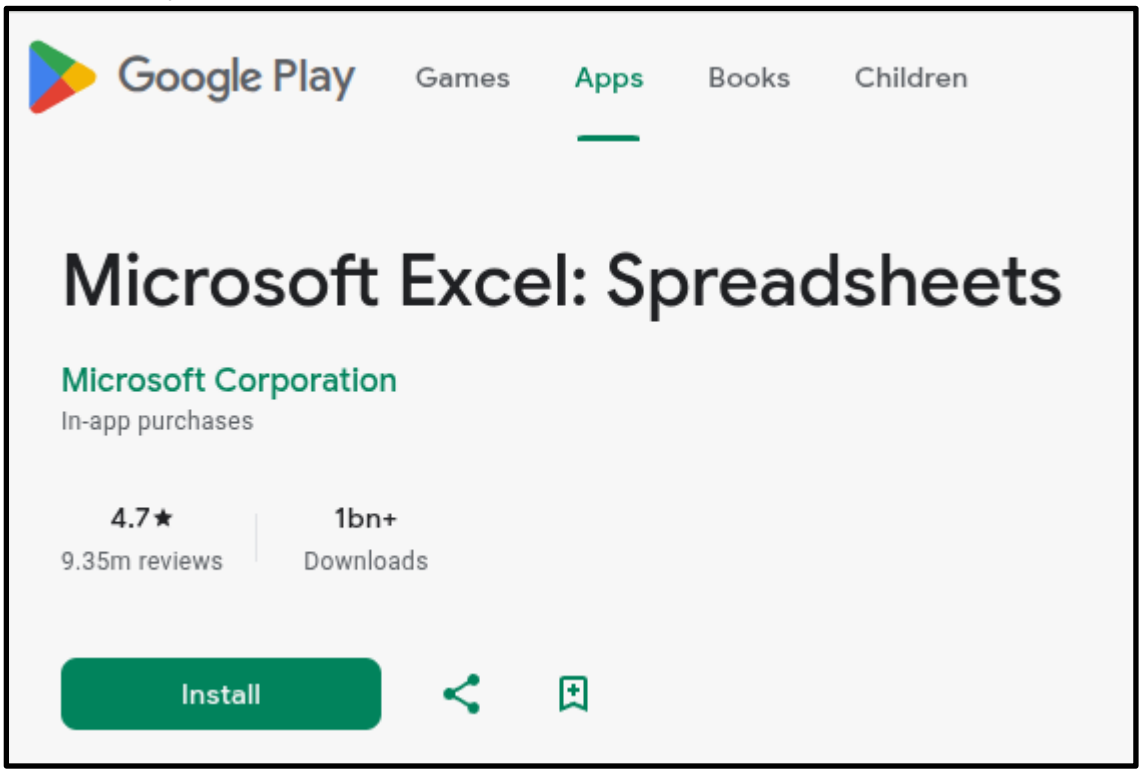
Use Google Play Protect to help keep your apps safe & your data private

Google Play Protect checks your apps and devices for harmful behavior.

- It runs a safety check on apps from the Google Play Store before you download them.
- It checks your device for potentially harmful apps from other sources. These harmful apps are sometimes called malware.
- It warns you about potentially harmful apps.
- It may deactivate or remove harmful apps from your device.
- It warns you about detected apps that violate our Unwanted Software Policy by hiding or misrepresenting important information.
- It sends you privacy alerts about apps that can get user permissions to access your personal information, violating our Developer Policy .
- It may reset app permissions to protect your privacy on certain Android versions.
- It may prevent an application from being installed that is unverified and uses sensitive device permissions that are commonly targeted by scammers to commit financial fraud.

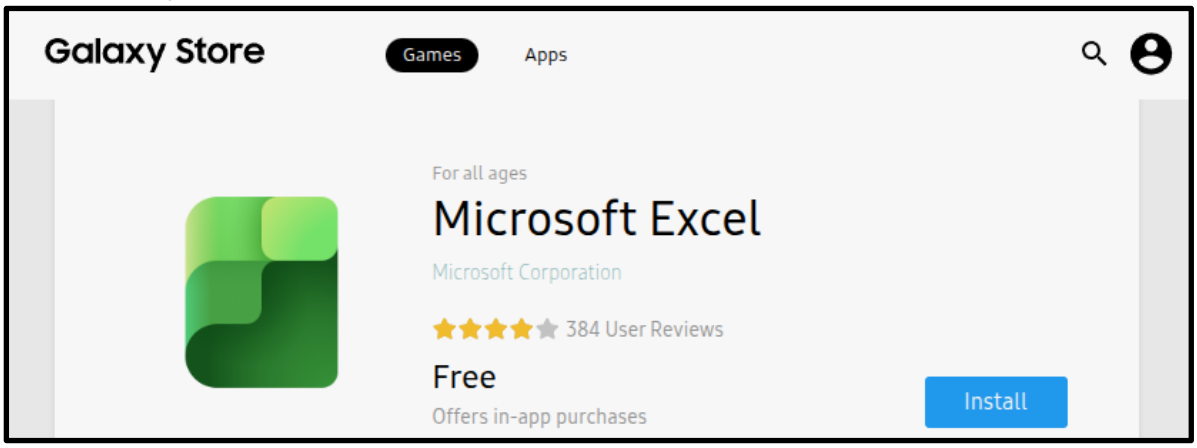
Annexure 22

Google Play Store Listing. Microsoft Excel.
https://play.google.com/store/apps/details?id=com.microsoft.office.excel
Obtained July 28th, 2026



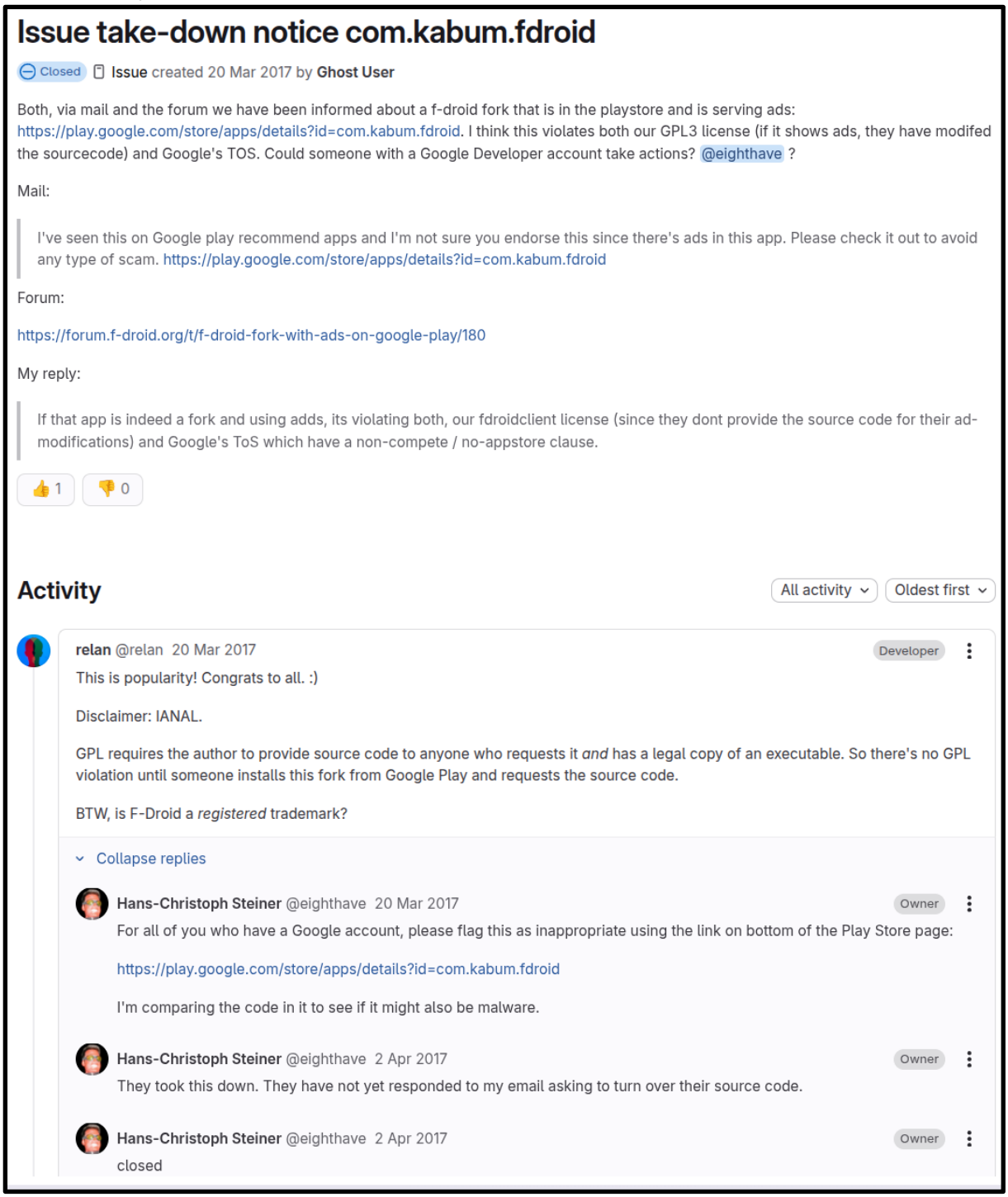
Annexure 23

Samsung Galaxy Store Listing. Microsoft Excel.
https://galaxystore.samsung.com/detail/com.microsoft.office.excel
Obtained July 28th, 2026



Annexure 24

F-Droid Project. Reporting counterfeit app to Google Play.
https://gitlab.com/fdroid/admin/-/work_items/21
Obtained July 28th, 2026



Issue take-down notice com.kabum.fdroid

Closed · **Issue** created 20 Mar 2017 by **Ghost User**

Both, via mail and the forum we have been informed about a f-droid fork that is in the playstore and is serving ads: https://play.google.com/store/apps/details?id=com.kabum.fdroid. I think this violates both our GPL3 license (if it shows ads, they have modifed the sourcecode) and Google's TOS. Could someone with a Google Developer account take actions? @eighthave ?

Mail:

> I've seen this on Google play recommend apps and I'm not sure you endorse this since there's ads in this app. Please check it out to avoid any type of scam. https://play.google.com/store/apps/details?id=com.kabum.fdroid

Forum:

https://forum.f-droid.org/t/f-droid-fork-with-ads-on-google-play/180

My reply:

> If that app is indeed a fork and using adds, its violating both, our fdroidclient license (since they dont provide the source code for their ad-modifications) and Google's ToS which have a non-compete / no-appstore clause.

👍 1 👎 0

Activity

All activity · Oldest first

relan @relan 20 Mar 2017 — Developer

This is popularity! Congrats to all. :)

Disclaimer: IANAL.

GPL requires the author to provide source code to anyone who requests it *and* has a legal copy of an executable. So there's no GPL violation until someone installs this fork from Google Play and requests the source code.

BTW, is F-Droid a *registered* trademark?

Collapse replies

Hans-Christoph Steiner @eighthave 20 Mar 2017 — Owner

For all of you who have a Google account, please flag this as inappropriate using the link on bottom of the Play Store page:

https://play.google.com/store/apps/details?id=com.kabum.fdroid

I'm comparing the code in it to see if it might also be malware.

Hans-Christoph Steiner @eighthave 2 Apr 2017 — Owner

They took this down. They have not yet responded to my email asking to turn over their source code.

Hans-Christoph Steiner @eighthave 2 Apr 2017 — Owner

closed

Annexure 25

GitHub. Aegis.
https://github.com/beemdevelopment/Aegis
Obtained July 28th, 2026

Aegis Authenticator

build passing
localized 80%
donate buy us a beer
chat 437 users

Aegis Authenticator is a free, secure and open source 2FA app for Android. It aims to provide a secure authenticator for your online services, while also including some features missing in existing authenticator apps, like proper encryption and backups. Aegis supports HOTP and TOTP, making it compatible with thousands of services.

For a list of frequently asked questions, please check out the FAQ.

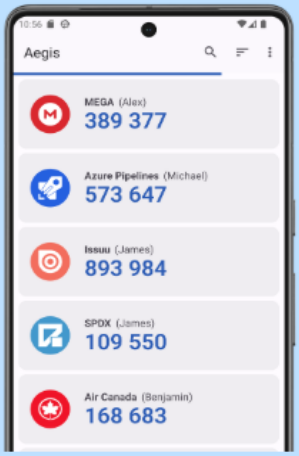
The security design of the app and the vault format is described in detail in this document.

Features

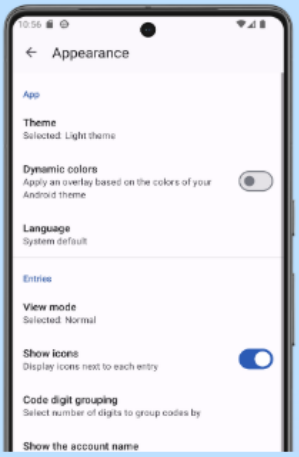
- Free and open source
- Secure
  - The vault is encrypted (AES-256-GCM), and can be unlocked with:
    - Password (scrypt)
    - Biometrics (Android Keystore)
  - Screen capture prevention
  - Tap to reveal
- Compatible with Google Authenticator
- Supports industry standard algorithms: HOTP and TOTP
- Lots of ways to add new entries
  - Scan a QR code or an image of one
  - Enter details manually
  - Import from other authenticator apps: 2FAS Authenticator, Authenticator Plus, Authy, andOTP, FreeOTP, FreeOTP+, Google Authenticator, Microsoft Authenticator, Plain text, Steam, TOTP Authenticator and WinAuth (root access is required for some of these)
- Organization
  - Alphabetic/custom sorting
  - Custom or automatically generated icons
  - Group entries together
  - Advanced entry editing
  - Search by name/issuer
- Material design with multiple themes: Light, Dark, AMOLED
- Export (plaintext or encrypted)
- Automatic backups of the vault to a location of your choosing

Screenshots

Organized
All your tokens in a polished interface



Customizable
Many customization options, encryption and backups



Annexure 26

GitHub. FreeOTP.
https://github.com/freeotp/freeotp-android
Obtained July 28th, 2026

FreeOTP

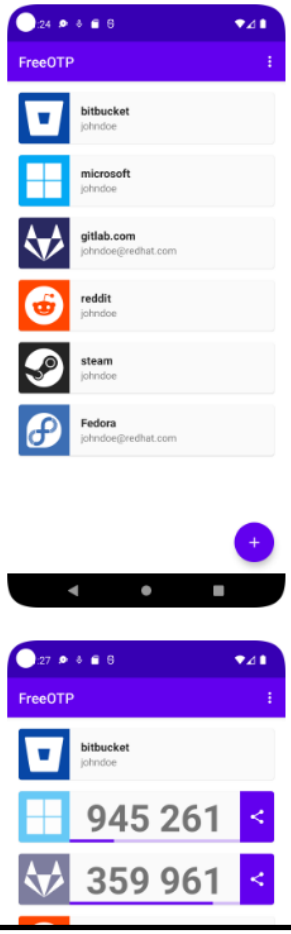
FreeOTP is a two-factor authentication application for systems utilizing one-time password protocols. Tokens can be added easily by scanning a QR code.

FreeOTP implements open standards:

- HOTP (HMAC-Based One-Time Password Algorithm) RFC 4226
- TOTP (Time-Based One-Time Password Algorithm) RFC 6238

This means that no proprietary server-side component is necessary: use any server-side component that implements these standards.

Screenshots



Annexure 27

Android Developer Documentation. Generate UI with image attachments.
https://developer.android.com/studio/gemini/generate-ui-with-images
Obtained July 28th, 2026

Generate UI with image attachments

Note: Image attachment is only available in Gemini's no-cost tier.

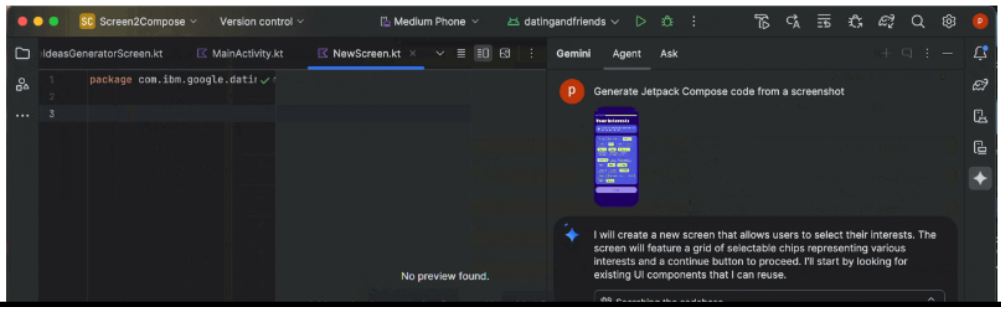
The AI agent is uniquely equipped to help you make your Android app UI vision a reality, using Jetpack Compose and following Android best practices. This page describes how to create and iterate on an app UI with AI.

To generate a UI with AI, follow these general steps:

1. Create a mockup of the app UI that you want. You can export a PNG from a design tool or even use a hand-drawn image.

Figure 1: A wireframe of an app user interface.

2. Attach the image to your query by clicking the **Attach image file** button. You can also click **Generate Code From Screenshot** directly from the **Preview** panel in a file without an existing preview.



3. In the chat field, as the AI agent to generate the UI code, for example "Generate Jetpack Compose code for the image provided." When you submit the query and image the AI agent suggests code to create the proposed UI. The AI agent usually provides the code for the Compose preview too, so you can quickly visualize the UI once you import it into your project—if it doesn't, ask it to generate the Compose previews.

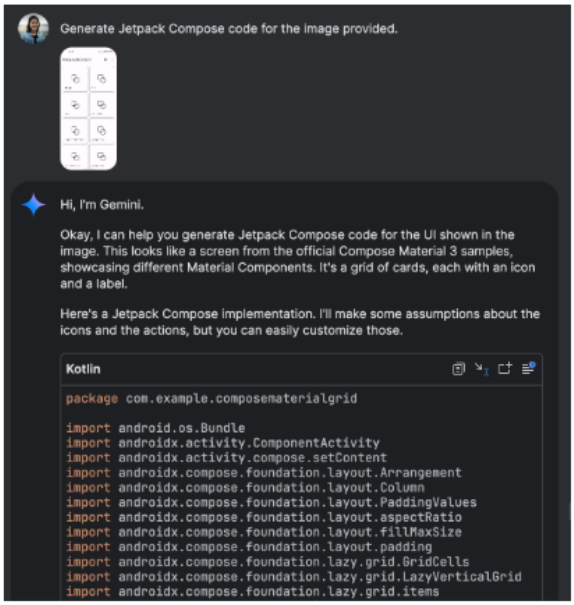
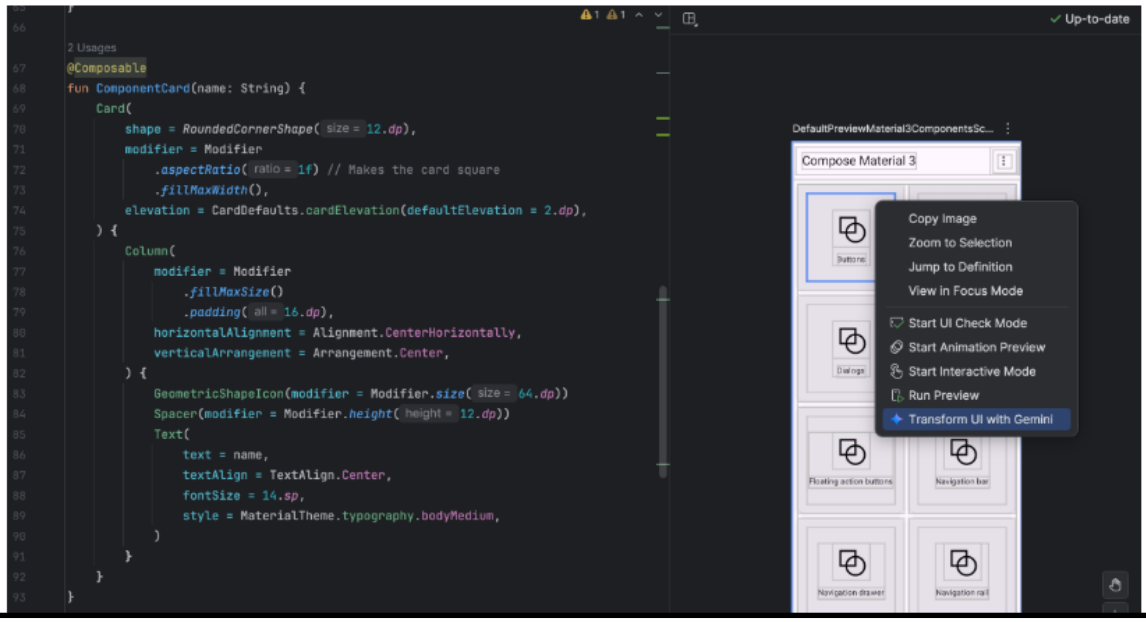


Figure 2: Gemini generating Jetpack Compose code from an attached image.

4. Once you import the code and can see the Compose preview in the preview panel, you can iterate on the UI by clicking directly on the preview and asking Gemini to transform it. If you have an image of what you want, you can also iterate on the UI by right-clicking the preview and selecting **AI Actions** > **Match UI to Target Image**.



Annexure 28

F-Droid Top Downloads.
https://grote.gitlab.io/fdroid-metrics-distilled/#total-downloads-per-app
Obtained July 28th, 2026

Total downloads per app

1. F-Droid 20_567_367 (App Store & Updater, System)
2. Termux 10_352_933 (Development)
3. Aurora Store 3_509_984 (App Store & Updater, System)
4. Fennec F-Droid 2_908_540 (Browser, Internet)
5. DAVx⁵ 2_137_949 (Internet)
6. Organic Maps • Offline Map & GPS 1_895_702 (Navigation)
7. PipePipe 1_834_922 (Internet, Multimedia, Online Media Player)
8. Fossify Gallery 1_823_908 (Gallery, Graphics, Multimedia)
9. Librera Reader 1_812_831 (Ebook Reader, Reading)
10. ProtonVPN - Secure and Free VPN 1_733_170 (Connectivity, Internet, VPN & Proxy)
11. Thunderbird: Free Your Inbox 1_586_828 (Connectivity, Email, Internet, Writing)
12. DuckDuckGo Privacy Browser 1_538_153 (AI Chat, Browser, Internet)
13. Tasks.org: Open-source To-Do Lists & Reminders 1_274_297 (Task, Writing)
14. KeePassDX Passkey Vault 1_266_074 (File Encryption & Vault, Keyboard & IME, Password & 2FA, Security)
15. VLC 1_250_834 (Cast, Local Media Player)
16. Tuta Mail 1_233_778 (Email, Internet)
17. K-9 Mail 1_156_012 (Email, Internet)
18. OsmAnd~ 1_105_761 (Navigation)
19. Nextcloud 1_052_737 (Calendar & Agenda, Cloud Storage & File Sync, Contact, Email)
20. Linux Command Library 988_931 (Reading, Science & Education)
21. Seal 986_329 (Download)
22. Fossify Calendar 980_319 (Calendar & Agenda)
23. NouTube 929_401 (Internet, Multimedia, Online Media Player)
24. SimpMusic 921_425 (Multimedia, Online Media Player)

Annexure 29

Android Rank. All Applications.
https://www.androidrank.org/categorystats?category=&price=all
Obtained July 28th, 2026

All applications

Ranking | Stats

Android statistics on android apps, an overview on android market. Charts below display an overview on the best android apps, indicate distribution of average rating and downloads of all apps indexed on androidrank.org.

App downloads distribution

Number of installs	Number of applications
10,000,000,000 - 50,000,000,000	20
5,000,000,000 - 10,000,000,000	19
1,000,000,000 - 5,000,000,000	132
500,000,000 - 1,000,000,000	131
100,000,000 - 500,000,000	1153
50,000,000 - 100,000,000	1556
10,000,000 - 50,000,000	10189
5,000,000 - 10,000,000	8816
1,000,000 - 5,000,000	33899
500,000 - 1,000,000	21143
100,000 - 500,000	61296
50,000 - 100,000	26656
10,000 - 50,000	37866

Indicates no. of installs distribution. Includes 202876 apps achieved at least 10,000 installs.

Annexure 30

F-Droid. Submitting to F-Droid Quick Start Guide.
https://f-droid.org/docs/Submitting_to_F-Droid_Quick_Start_Guide/
Obtained July 28th, 2026

Application Review Process

Once the inclusion proposal is filed, the application will enter a reviewing process where F-Droid staff look into the applications source code and determine whether it fits for inclusion (and when it's not, determine all necessary steps to make it so).

As F-Droid is a software repository which promises users free software, a review process is for ensuring that all applications distributed from the F-Droid main repository are Free Software.

This is a nonexhaustive list of what a reviewer would do:

- They will go to your source code repository, and look for copyright notices in license files, including *README*, to check that the proposed application is released under a recognized Free Software and/or OSI license(s).
- They will look at your build script to see which build system you use, and whether F-Droid build server can handle it (Ant and Gradle are the most common and easiest ones).
- They will try to download a copy of your source code.
- They will look in all source code files to verify that their licenses are consistent with corresponding license/README files.
- They will check if your application uses any pre-compiled libraries or binary blobs.
- They will look at your non-source code files to identify Non-Free resources used in your application.
- They will skim through the source code to see if your application uses Non-Free dependencies, shows advertisements, tracks users, promotes or depends on Non-Free or non-changeable services/applications, or does anything that is harmful or otherwise undesirable for users.
- They will list a summary of any *AntiFeatures* in your application.
- They will try patching your application to remove usage of third-party proprietary software (if there is any).
- They will try to determine a suitable update process for your application (e.g. by looking at how your releases relate to VCS tags and/or version information in *AndroidManifest.xml*, build.gradle.kts, pubspec.yaml or elsewhere).
- They will try writing a suitable metadata file for your application, and add it to local F-Droid build server instance. (`fdroid rewritemeta` , `fdroid lint` are used to ensure that metadata is well-formed)
- They will try to build your application in an isolated environment to see if the process succeeds and yield a functional APK.
- Usually the Gitlab CI will be used, but for apps that need more resources *(storage, memory, time)* than what the runners can offer, they might use local machines to prepare and test the metadata.
- If all went smoothly, they will add a new metadata file to their local *fdroiddata* git repository and synchronizes the change to GitLab.

In the case that the application failed some steps in the review, feedback will be given in the original submission queue thread where the proposal was posted.

Once the *fdroiddata* repository is updated on GitLab, it's mostly just a matter of time before F-Droid's official build server will fetch, build, and publish your application on the main F-Droid repository.

You can confirm the inclusion of your application by looking at the GitLab *fdroiddata* revision history.

Build Process

After the application metadata is added to *fdroiddata* GitLab repository, the next step is for the main F-Droid build server to fetch the applications source code and related components, build the application, and publish it on the main F-Droid repository.

This build process is not running on a schedule, a cycle is started after the previous one has been published, applications are processed in batch *(you can infer the average frequency by looking at past cycles)*. As steps are done behind the scene and are mostly automatic; all the submitter needs to do is to wait for it to finish.

A record of a successful build process for one application is provided on the F-Droid's website page for that specific app (e.g. see the Build Log for the F-Droid Client).

Apps that fail will have the log available during the build cycle on the F-Droid Monitor - Running page or, if in the previous cycle, on the Build page. This is useful to aid in diagnosing problems when the build unexpectedly failed.

What to Expect

When your application metadata is approved and accepted into the *fdroiddata* git repository on GitLab, **it won't immediately appear** in the main F-Droid repository.

Provided that your application does not have any build problems, it would takes somewhere **around 24 to 48 hours** from *fdroiddata* merge until the application to appears in the main repository.1 This timing limitation is due to the APK signing part of the build process, which requires human intervention on keystore access step.2

After publishing, the apps are immediately available in the repository and any client can now update or install. The f-droid.org website takes time to generate pages for updated apps and new apps, for all the languages, so there's a slight delay until it gets updated.

External Links

- F-Droid application submission queue on GitLab (for new submissions)
- F-Droid application submission queue on forum (for following-up old submissions)
- *fdroiddata* GitLab repository
- *fdroiddata* revision history

Annexure 31

Android Developer Documentation. App signing.
https://source.android.com/docs/security/features/apksigning
Obtained July 28th, 2026

App signing

Note: If you are using Android App Links make sure to update the SHA256 fingerprints of your keys in the corresponding Digital Asset Links JSON file on your website.

App signing allows developers to identify the author of the app and to update their app without creating complicated interfaces and permissions. Every app that is run on the Android platform must be signed by the developer. Apps that attempt to install without being signed are rejected by either Google Play or the package installer on the Android device.

On Google Play, app signing bridges the trust Google has with the developer and the trust the developer has with their app. Developers know their app is provided, unmodified, to the Android device; and developers can be held accountable for behavior of their app.

On Android, app signing is the first step to placing an app in its Application Sandbox. The signed app certificate defines which user ID is associated with which app; different apps run under different user IDs. App signing ensures that one app can't access any other app except through well-defined IPC.

When an app (APK file) is installed onto an Android device, the Package Manager verifies that the APK has been properly signed with the certificate included in that APK. If the certificate (or, more accurately, the public key in the certificate) matches the key used to sign any other APK on the device, the new APK has the option to specify in the manifest that it shares a UID with the other similarly signed APKs.

Apps can be signed by a third-party (OEM, operator, alternative market) or self-signed. Android provides code signing using self-signed certificates that developers can generate without external assistance or permission. Apps don't have to be signed by a central authority. Android currently doesn't perform CA verification for app certificates.

Apps are also able to declare security permissions at the Signature protection level, restricting access only to apps signed with the same key while maintaining distinct UIDs and Application Sandboxes. A closer relationship with a shared Application Sandbox is allowed using the shared UID feature where two or more apps signed with same developer key can declare a shared UID in their manifest.

Annexure 32

'Open-source versus proprietary software: Is one more reliable and secure than the other?' A. Boulanger, *IBM Systems Journal* (Volume 44, Issue 2, 2005)

Open-source versus proprietary software: Is one more reliable and secure than the other?

A. Boulanger

One of the most powerful movements in the information technology community today is the widespread adoption of free and open-source software (FOSS). What was once an idealistic fringe movement conceived and formalized by MacArthur award laureate Richard Stallman has now become one of the most powerful influences in the world of information technology. As FOSS systems grow in popularity, questions of the reliability and security of these systems emerge, especially in comparison with proprietary systems. This paper surveys the arguments presented by proponents of each type of software in published reports and discusses the deployment and reliability figures for both FOSS and proprietary systems as well.

The explosive increase in the number of deployed free and open-source software (FOSS) systems has changed the world of information technology. When the first FOSS systems were developed, many of the users of these early systems were themselves technologists. Moreover, the distribution and use of such FOSS systems was initially limited to academia, research laboratories, and technical user groups. Today, however, FOSS systems are being developed and designed for mass consumption. Most of the businesses on the Internet use FOSS-developed systems, and retail stores such as Wal-Mart are offering to the general public steeply discounted computers that take advantage of FOSS-developed software. As the group of people and organizations that depends on FOSS technologies continues to grow, it becomes increasingly important that FOSS systems be secure and reliable.

Many FOSS systems were originally developed by a loose collaboration of volunteer programmers. The completed systems were then released to the public, and anyone could acquire and use these systems without paying a licensing fee. Free support for these systems was also provided by the volunteer community in the form of mailing lists and Web sites. Currently, however, many FOSS projects are professional efforts in which development is performed by a team of paid programmers, and the system is supported either without charge or through fees and subscriptions. In contrast, traditional proprietary systems are developed by a team of designers,

project managers, programmers, technical writers, and quality assurance engineers. The systems they produce undergo design reviews, development progress reports, and formal quality assurance testing. Once completed, these systems are packaged commodities that are sold or licensed to the public for a fee. Support for the software product is usually provided by the developer of the system.

Which model is more reliable in terms of availability and security? Many papers discussing these issues have been published by proponents of each type of software. This paper examines the arguments presented in these published reports as well as the deployment and reliability figures for both open and proprietary systems.

SECURITY AND RELIABILITY CONSIDERATIONS FOR FOSS AND PROPRIETARY SYSTEMS

The security and reliability of FOSS-based systems are currently topics of an often heated debate. Proprietary vendors are funding, producing, and publishing reports supporting the position that closed-source proprietary systems offer superior security relative to their FOSS counterparts. For every report that is published claiming the superior security of proprietary systems, the FOSS community responds with a report refuting these claims.

Perhaps a significant reason for this heated debate is the fact that widespread adoption of the FOSS model would directly threaten the revenue stream of vendors of proprietary software. In several recent 10-Q quarterly filings with the Securities and Exchange Commission, Microsoft, one of the world's largest software publishers, has stated that the popularization and adoption of FOSS systems pose a significant challenge to its business model.[1] It is not surprising then that proprietary software vendors are on the offensive, attempting to discredit FOSS-developed systems.

Arguments about the relative security and reliability of FOSS and proprietary software typically focus on two key issues: availability of source code and software defect levels. We discuss these issues in the following sections.

Availability of source code

In June 2002, the white paper "Opening the Open Source Debate"[2] was released by the Alexis de Tocqueville Institution, an organization funded in part by Microsoft. Among its most controversial findings was that "Open source GPL [General Public License] use by government agencies could easily become a national security concern. Government use of software in the public domain is exceptionally risky." The basis for this assertion is the assumption that publicly available source code invites "hackers"[3] to examine the code in order first to search for exploitable vulnerabilities and then to develop and deploy Trojan horses and other types of malicious software. Therefore, the study concludes that the availability of source code is a significant security threat to government organizations using FOSS.

There are several problems with this assertion. It is inferred that closed-source proprietary systems are automatically more secure than their FOSS counterparts, a "security through obscurity" approach. If this assertion were true, then the number and rate of published vulnerability reports for closed-source systems should be significantly lower than those of their FOSS equivalents.

However, the available data does not support this assertion. In fact, many FOSS systems have substantially lower rates of published vulnerabilities than their closed-source counterparts. For example, a recent report[4] showed that Apache**, a FOSS Web server whose history and development[5] will be discussed in more detail later in this paper, suffered from substantially fewer published vulnerabilities than Microsoft's IIS (Internet Information Server) and marginally more vulnerabilities than Netscape** Enterprise Server. If the de Tocqueville Institution's assertion were true, then Apache should have had significantly more published vulnerabilities than the closed-source Web servers.

Hiding the source code for a system does not provide any additional security. People searching for vulnerabilities do not require source code to discover software defects. For example, a common way to locate a software defect is to send a program unexpected and unusual data and then monitor how the system responds.[6] If the system fails or behaves erratically as a result of the input, this might indicate a flaw in the system that would warrant further investigation.

With the prevalence of sophisticated software monitoring, debugging, and disassembly tools,

much of the source code can be derived from the binary version of the executable program. Anyone interested in obtaining the source code would simply have to apply one of many widely available tools to the program. The output from these programs, while not perfect, would deliver sufficient information to make it fairly easy to understand the internal working of the system.

Moreover, source code for many deployed systems has often been inadvertently leaked. Many software producers outsource code development or exchange source code with other organizations for a variety of reasons. Anyone with access to the systems involved, authorized or not, can obtain a copy of the source code and distribute it. It is a very common practice in the hacker community to traffic in source code packages. In the course of investigating systems intrusions, our organization has discovered copies of the source code for nearly all of the major operating systems on hacker systems. By examining the source code, hackers hope to locate new vulnerabilities and produce what are known as *zero-day* exploits, attacks against vulnerabilities which the software vendor has not yet found. Zero-day exploits are the most prized exploits in the hacker community because the targeted systems are defenseless. On the surface, the possibility of such exploits appears to support the most common rationale for keeping source code secret, namely preventing miscreants from using the code to discover software defects and attack vulnerable systems. However, the real problem may not be that the source code is being used to discover vulnerabilities. Rather, it may be that only two groups of people have access to the source code, the small group of developers, who are tasked with developing and maintaining the system, and the potentially larger community of hackers, who are motivated to discover and exploit vulnerabilities. When the source code is widely available to everyone, as is the case with FOSS systems, there are more opportunities for people outside of the hacker community to examine and correct potential software defects before they can be exploited.

There are, of course, examples of FOSS systems that have been notoriously insecure. Software defects in *sendmail*, a FOSS mail transfer agent, have been a favorite target of attack for years. In November of 1988, the first Internet worm was released by Cornell graduate student Robert Morris.[7] The worm quickly spread throughout the emerging Internet, in part by exploiting vulnerabilities in the *sendmail* and *fingerd* routines, causing widespread outages.

■ The security and reliability of FOSS-based systems are currently topics of an often heated debate ■

According to one report, the Internet worm of 1988 impacted 6,000 out of 60,000 systems, or 10 percent of the systems on the emerging Internet. Morris discovered these vulnerabilities while analyzing the source code for *sendmail* and *finger*. However, it was the availability of the source code that enabled the Internet community to respond quickly and mitigate this threat. In their paper "An Analysis of the Internet Virus of November 1988,"[8] Mark Eichin and Jon Rochlis remarked, "Source availability was important. All of the sites that responded quickly and made progress in truly understanding the virus had the UNIX** source code."

Having the source code widely available is a multi-edged sword. The hacker community can use the code to analyze systems and locate vulnerabilities. The developer community can maintain and improve the code through analysis and can use formalized testing methodologies to discover and remedy software defects before they can be exploited. The user community can also use the source code in response to an attack to discover and correct the vector through which the attacker has exploited the system. After the flaw has been discovered, the user community can respond rapidly by patching the system, removing the vulnerability, and then sharing the patch with the public. With a closed system, users are completely dependent upon the vendor's ability to discover the vulnerability, and then develop and distribute fixes.

A differentiator then between widely deployed FOSS systems and their proprietary counterparts is the value of source code in responding to security threats. Having the source code available was critical in assisting systems administrators to respond to the Internet worm of 1988. Technical staffs reviewed the source code of the affected systems and were able to understand the vulnerability and develop defenses against both the immediate threat

and future attacks. This advantage of having source code available was outlined in the government-funded research report "A Business Case Study of Open Source Software," published in July of 2001 by MITRE**.[9] This report was the result of a publicly funded study to help program managers evaluate open-source software and development methodologies, as well as their potential application within the managers' technical programs. The MITRE report stated that in the FOSS community "popular open-source products have access to extensive expertise, and this enables the software to achieve a high level of efficiency," and that software patches, or corrections, happen "potentially an order of magnitude faster than those of proprietary software."

■ Widespread adoption of the FOSS model would directly threaten the revenue stream of vendors of proprietary software ■

Software defect levels

Software defects are a fact of life, and any software package, whether FOSS or proprietary, is likely to have a substantial number of flaws. A percentage of these defects will directly impact the security of the system. According to the Software Engineering Institute, an experienced programmer produces approximately one defect per 100 lines of code, or an average defect rate of 1 percent.[10] If, during the software development life cycle, 99 percent of those defects are discovered and remedied, then approximately 1000 software defects will remain in a software package consisting of one million lines of source code, a modest code base by current standards. For example, the Red Hat Linux** 7.1 distribution consists of approximately 30 million lines of code (LOC), and Microsoft Windows** XP contains approximately 40 million LOC. Even with extensive testing and defect remediation, there will still remain a very significant number of software defects in these systems. Using the defect rate statistics noted earlier, Windows XP and Red Hat Linux would be estimated to have approximately 40,000 and 30,000 undiscovered defects respectively. In addition, both proprietary and FOSS systems are constantly evolving. As they progress through the software life cycle, features are added to the system and defects are discovered and remedied. Because most software packages are in this constant state of evolution, it is difficult to estimate the number of software defects a particular system will contain. When code is added to a system, there is also the possibility that the developers will introduce a new defect into the system. There are even examples where a developer has produced modifications to correct one security problem but inadvertently introduced another. Software defects can even occur due to problems having nothing to do with changes in the source code. For example, there have been instances where a defect in a development tool has introduced a defect into a system build.

Software defects are an unavoidable problem that can reduce the overall reliability of a system. When the reliability of a system is reduced, the overall security of the system is also reduced. Reliability and security are inextricably intertwined. A highly reliable system has fewer software defects and, in general, is more secure than an unreliable system. To use an analogy from the physical world, a padlock built with an excellent locking mechanism (security) but constructed with defective steel (a defect) offers very little security. The same is true for a lock using excellent construction materials (security), but having a defect in the locking mechanism (reliability) that makes the lock vulnerable. A classic example of the latter involved the published demonstration that a particular high-security bicycle lock could be quickly opened with an ordinary Bic** pen.[11]

Every software package release, whether FOSS or proprietary, contains a number of defects. The available data suggest, however, that when a software flaw is discovered, the FOSS community responds more rapidly than proprietary software vendors. According to an article published by e-Week Labs, FOSS organizations in general respond to problems more quickly and openly, while proprietary "software vendors instinctively cover up, deny, and delay."[12] The FOSS organizations produce small fixes that are available publicly through mailing lists and Web sites. Users of the affected software can download the update, apply the patch, and rebuild the system. The patch itself is in turn a segment of source code that is available for inspection by the general public. In contrast,

proprietary software vendors tend to wait and roll out large cumulative releases known as *service packages*. These service packages usually contain only binary code, which is not open for public scrutiny and can potentially introduce new problems into the system. The users of proprietary software products must blindly trust the integrity and competence of the software publisher; whereas, FOSS users know that the patches they have installed can be (and will be) publicly scrutinized.

VENDOR-NEUTRAL STUDIES

If the FOSS community is more responsive to software defects, then it would be reasonable to conclude that FOSS products should show higher reliability statistics than their proprietary counterparts. Although each side of the FOSS issue has offered anecdotal evidence supporting its respective position, there are several vendor-neutral studies that favor popular FOSS-developed systems.

In 1990, Professor Barton Miller from the University of Wisconsin developed the *fuzz* system, a system which produced random data streams that were then fed as input to programs from several proprietary versions of the UNIX system.[6] Dr. Miller discovered that 24–33 percent of the programs tested failed when fed the fuzz-generated data. Each of these failures can be directly attributable to a software defect. A properly written program should not fail when it receives unexpected or random data. In 1995, Dr. Miller revisited this work and included in his test both Linux-derived and other freely available GNU utilities. (GNU is a recursive acronym for "GNU's not UNIX.") The new study tested over 80 utility programs from nine different versions of UNIX. Of these UNIX platforms, seven were from proprietary vendors and two were from the FOSS community. The major results of the study were published in the paper "Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services."[13] Since the original study, the proprietary systems had improved, and their overall failure rate dropped from 24–33 percent down to 18–23 percent. For the FOSS systems, the failure rate for the Linux utilities was second lowest at 9 percent failures, and the failure rate for the GNU utilities was the lowest at 6 percent. The results of the study demonstrate that popular FOSS implementations of UNIX utilities can have significantly better resilience to unexpected input than their proprietary counterparts. This in turn could indicate higher quality and reliability for the FOSS utilities. This is not to say

■ Hiding the source code for a system does not provide any additional security ■

that all FOSS systems are more reliable than proprietary systems. However, this experiment does demonstrate that the quality of software developed under the FOSS model can equal or exceed the quality of commercially developed software.

Another study conducted in 1999 by Bloor Research, an independent IT analysis and consultancy organization based in the United Kingdom, pitted Windows NT** against GNU/Linux in a head-to-head comparison.[14] The test was conducted for one year and rated each of the platforms according to nine criteria. Overall, GNU/Linux was rated as superior to Windows NT in seven of the nine categories. In particular, in the OS (Operating System) Availability category, GNU/Linux had a significantly better reliability rating than Windows NT. During the year of testing, the GNU/Linux system did not experience a single outage that was attributable to software. However, a hardware failure resulting from a hard disk malfunction did cause an outage for the GNU/Linux system that lasted four hours before service was restored. The resulting availability figure for GNU/Linux was thus 8756 hours out of 8760 hours, or 99.95 percent uptime. During the same period of time the Windows NT system suffered a total of 68 failures. Of the 68 failures, one failure was a hard disk failure; twenty-six failures were attributed to memory management faults; eight failures were attributable to file system faults; and the remaining failures were of unknown origin. The amount of lost time due to these outages was reported as 65 hours, resulting in an overall availability of the Window NT system of 99.26 percent or 8695 hours out of a total of 8760 hours. The results of this one-year study are impressive and demonstrate that FOSS-developed systems, at the very least, can compete well with their proprietary counterparts, providing stable and reliable server platforms.

Additional information comes from companies that develop automated software-inspection services. Some background will help explain the role of these

companies in the software industry. In large projects the people responsible for maintaining a system are often not the same people who originally developed the system. Unless the maintainers are careful and

■ Reliability and security are inextricably intertwined ■

fully understand the system, it becomes very easy to make a mistake that can affect the overall quality of the system code. One of the ways to increase the reliability of a system is to review the source code for defects and remedy them before the system is released. Typically the inspection process is performed through formal code reviews and evaluations. This process is very labor-intensive and time-consuming. Historically, as systems grew and it became more expensive to perform formalized code reviews, researchers developed ways to automate this process and make code reviews less labor-intensive than manual inspection. Several companies now offer automated software-inspection services, allowing software publishers to outsource their code reviews. One such company, Reasoning, Inc., has been assisting organizations to improve the quality of their systems through automated software inspection for almost 20 years and is considered a leader in this field.

In 2003, Reasoning conducted a study of the implementation of the Internet protocol code in the 2.4.19 version of the Linux kernel and in five proprietary operating systems.[15] The purpose of the study was to use automated code inspection techniques to compare the quality and defect rate of each implementation of the TCP/IP (Transmission Control Protocol/Internet Protocol) networking software. Reasoning discovered that the defect rate for the Linux code was 0.1 reported defects per 1000 lines of code (KLOC). The defect rate for proprietary implementations was reported to be 0.55 defects per KLOC. Reasoning concluded that the FOSS implementation of TCP/IP had a significantly lower defect density compared to the implementations in the five proprietary operating systems. The study also concluded that the overall quality of the FOSS package rated in the top third of all source-code projects that had been inspected by Reasoning.

In July of 2003, Reasoning analyzed the popular Apache Web server software package.[16] The Apache Web server is a FOSS system developed and maintained by the Apache Software Foundation, a membership-based not-for-profit corporation.[5] The Apache server is the dominant HTTP (Hypertext Transfer Protocol) server package on the Internet today, according to a recent survey by Netcraft.[17] This survey, conducted in June 2004, reported that of the 51.6 million identifiable servers on the Internet at that time, Apache had over 67 percent of the market, followed by Microsoft with a 21 percent market share. With so many organizations relying on FOSS technology for their Internet presence, it would obviously be valuable for IT managers to have a vendor-neutral software-quality metric to assist in a decision whether to deploy FOSS or proprietary systems. Reasoning concluded in their study that the defect density for the 2.1 release of the Apache system was 0.53 defects per KLOC. To put that figure into perspective, Reasoning compared the defect density of the Apache system to the 200 other projects Reasoning had analyzed at that time, both FOSS and proprietary, involving a total of 33 million analyzed lines of code. The top third of these 200 projects showed defect densities of less than 0.36 defects per KLOC; defect densities of the middle third ranged from 0.36 to 0.71 defects per KLOC; the bottom third had defect densities greater than 0.71 defects per KLOC. Given these statistics, the defect rate for the Apache system falls somewhere in the middle compared to the rest of the industry and slightly above the average defect density Reasoning has found for proprietary software (0.51 defects per KLOC).

Reasoning then continued their study of FOSS quality by inspecting the source code for the 4.0.16 version of MySQL**, the leading open-source database system.[18] In this December 2003 study, Reasoning examined 236,000 lines of MySQL source code and detected 21 software defects in the system. The study determined that the code quality of the MySQL system was six times better than that of comparable proprietary code, with a defect density for the MySQL system of 0.09 defects per KLOC compared to the average defect rate of 0.51 defects per KLOC for proprietary code. Not incidentally, as a result of this study the maintainers of the MySQL package promptly corrected the problems Reasoning had found and produced a maintenance release,

Version 4.0.17, that was made available for download from their distribution site.

These studies suggest that FOSS systems can meet, or even exceed, the quality of their proprietary counterparts. Furthermore, the Reasoning studies only inspected defect rates in source code. Their studies did not include other aspects of software packages such as usability, compatibility, features, and support costs, all important aspects that need to be considered in the decision-making process when deploying systems. The studies from Reasoning do suggest that FOSS-developed systems offer viable alternatives to proprietary systems in terms of software quality and reliability.

COMPARISON OF DEVELOPMENT PROCESSES

The empirical evidence from these studies suggests that popular FOSS-developed systems are, at the very least, as secure and reliable as their proprietary counterparts. Obviously, proprietary software is expensive to develop and support. How can a disparate loose-knit group of developers produce software of comparable quality for free? A look into how open-source software is produced provides some insight into the success of FOSS projects.

Most traditional proprietary software projects use a variant of the waterfall model[19] in their software development process. The waterfall model has five well-defined phases:

1. *The requirements phase*, in which the problem and the requirements of the proposed system are defined.

2. *The system and software design phase*, in which a technical solution is applied to the problem.

3. *The implementation and unit-testing phase*, in which the components of the technical solution are developed and individually tested.

4. *The integration and system-testing phase*, in which all of the individual components are aggregated and tested as a whole unit and compared to the defined requirements.

5. *The support and maintenance phase*, which begins when the tested system, having met the defined requirements, is deployed and maintained.

The entire process is iterative, in that at any phase in the process the project may be forced to return to an earlier phase as new problems and requirements are defined. This is the developer feedback loop.

■ FOSS systems can meet, or even exceed, the quality of their proprietary counterparts ■

Problems that are detected downstream in the testing and maintenance phases are communicated back to the programmers. The programmers research a solution to the problem, develop a correction, and resubmit the component for testing and integration.

FOSS development is often less structured, but still shares features of traditional development. The main difference is the addition of a consumer feedback loop, wherein the users of the systems are encouraged to directly participate as part of the development community. In a proprietary environment, consumers may report software defects and offer suggestions, but they are unable to directly participate in the development process because they lack access to the system source code. In FOSS development, every consumer has access to the source code and can thus directly participate in the continuous improvement of the software package. In reality only a small percentage of the user base will have the desire or expertise to actively participate in the project. However, when the user base grows large enough, that small percentage of users can swell into a substantial number of contributors.

The development of the Apache Web server demonstrates this process.[5] The Apache system emerged in 1995 and was derived from a set of patches that were applied to the then-popular NCSA (National Center for Supercomputing Applications) Web server source code (leading to the name "Apache" server, a play on the words "a patchy" server). These patches were contributed by frustrated users of the then-largely-unsupported NCSA server who required additional functionality. These users had access to the server source code and were able to modify the system to meet their requirements. They then submitted their patches to a group of volunteers who maintained the system. Eventually the group abandoned the initial set of patches and completely redesigned the system. In December 1995, Apache

Version 1.0 was formally released and, as noted previously, quickly became the dominant Web server on the Internet. As the Apache system grew in popularity, more organizations became dependent on this server technology to support their missions, and as a result, more people contributed to the project. The group of volunteers who reviewed submissions and maintained the Apache source code continued to grow in size, eventually forming the Apache Group in 1995, and then the Apache Software Foundation in 1999.

■ FOSS-developed systems have a distinct advantage in their ability to respond to security threats ■

The Apache Software Foundation currently consists of 22 core developers who contribute to the development and maintenance of the Apache system. Users of the system submit patches, bug reports, and suggestions for improvement to this core team of developers. The Apache development community communicates through Internet mailing lists and Web sites. Any piece of software that is submitted is peer-reviewed and extensively tested before being included in the source code repository. Every Apache source code repository contains status information on changes and plans for improvements, as well as references to outstanding issues, so that all developers can stay informed. This collaborative effort is synchronized through status information and identification of the developers who are responsible for specific parts of the project. The actual software development and module testing are performed on the developer's machine. When a module has been tested and is ready for release, the developer posts the software patches to the developer mailing list. Subscribers to the developer list then review the proposed modifications and test them against their own systems. Only when the proposed changes have been peer-reviewed and tested are they incorporated into the Apache source repository. If there are problems with the proposed code changes, such as incompatibility issues or software defects, the developer community has the opportunity to detect and remedy these defects before they can impact the rest of the system. This is the developer feedback loop of collaborative software development: people responsible for developing the system reviewing one another's work.

In the consumer feedback loop, users of the published system uncover defects and submit bug reports and suggestions for improvement to the core developers. Because the source code is publicly available, users are able to locate defects, submit suspect code fragments, and contribute patches to correct the problem. Once a patch is submitted from the consumer loop, it undergoes rigorous peer review and testing in the developer loop before making its way into the system source code repository. This is a remarkably efficient system for distributing the burden of code review across both the developer and user domains. Every user can potentially become a developer and contribute to the overall success of the software package. One caveat is that this system may only work well with popular FOSS projects. A small orphaned FOSS project could lack the critical mass of resources that a larger, more popular project enjoys with its extensive user base. However, for large FOSS projects this system appears to work very well. There are many examples of FOSS-developed systems that enjoy massive user bases who continually work to improve the reliability and functionality of the system. Once a critical mass of developers and users emerges, the open-source development project can achieve the same reliability and security standards as proprietary systems.

CONCLUSIONS

Which is more secure: closed or open-source software? Unfortunately the answer is not that clear. In general, both FOSS and proprietary systems are roughly equivalent in terms of security and reliability. Neither is inherently more secure or reliable than the other. Analytical arguments made in favor of either approach are not conclusive. Empirical studies have suggested that FOSS can potentially outperform proprietary systems. Nonetheless, any system that was not developed to be secure invariably will not be. There are certainly proprietary systems deployed that are more secure than their FOSS counterparts (e.g., S/COMP or GEMSOS** versus Gnu/Linux), just as there are FOSS-deployed systems that appear to be more secure than their proprietary equivalent (e.g., Apache versus Microsoft IIS).[20] One problem with attempting to quantify the security of proprietary and FOSS systems is the fact that verifiably trustworthy systems are very

difficult and expensive to develop and certify. To date there are only seven operating systems that have achieved a Common Criteria Certification evaluation assurance level (CC EAL) rating of 4 (meaning "methodically designed, tested and reviewed").[21] The highest rated FOSS-developed operating system is the SUSE** Enterprise Server V8 distribution of Gnu/Linux with a rating of CC EAL 3+ (meaning "methodically tested and checked"). This does not necessarily mean that FOSS operating systems with lower ratings are less reliable than proprietary systems. This can also mean that the funding required for formal certification has not been made available. Because the CC certification is expensive to obtain, only larger organizations can afford to sponsor a CC evaluation, especially at the higher levels of certification.

Another problem is that every software system mentioned in this article, both open-source and proprietary, requires frequent patching to remediate defects. Any system that requires frequent patching is inherently insecure. Using patch counts as a metric for security is misleading. A system that requires a security patch every six months is not twice as secure as a system that requires patching every three months. They are both insecure. To use another analogy from the physical world, a car that explodes once every 1000 miles cannot be considered twice as safe as a car that explodes once every 500 miles. As in the software example, both vehicles should be considered unsafe. Another issue with using published vulnerabilities as a security metric is that software systems are under constant change. Whenever old software defects are discovered and remedied, new software defects may be introduced into the system. Because systems are constantly evolving, there is no easy way to determine the absolute number of defects in a system at a given instant. However, FOSS-developed systems have a distinct advantage in their ability to respond to security threats. As noted previously, the organizations that responded most successfully to the Internet Worm of 1988 had access to the UNIX source code. Having the source code enabled technical personnel to understand the immediate threat and then share that information with other affected organizations.

It is this sharing of information that is the key strength behind the FOSS movement. FOSS developers have the ability to analyze how previous systems were constructed and "stand on the shoulders of giants." As in the scientific research community, this free exchange of information promotes innovation and advances the field. The FOSS movement can use this shared information to encourage participation from a global talent pool. When the number of users of a FOSS project increases, so too will the number of developers who can potentially participate in the project. Once a critical mass of users has formed, the momentum from this combined effort will yield quality systems that meet and exceed the security and reliability metrics of their proprietary counterparts—at a much reduced cost.

The FOSS movement is gaining traction. What was once an idealized concept espoused by hackers, hobbyists, and academics is now formalized and organized and is the dominant technology behind the Internet. As FOSS-based technologies continue to gain market share, proprietary software publishers will be forced to innovate to remain competitive and survive. It will be interesting to watch and see where FOSS technology takes us in the future. It will be even more interesting to participate.

**Trademark or registered trademark of Aesec, Linus Torvalds, Microsoft Corporation, MySQL AB Company, Netscape Communications Group, Red Hat, Inc., Société BIC, SUSE LINUX AG, The MITRE Corporation, The Apache Software Foundation, or The Open Group.

CITED REFERENCES

1. See, for example, *Securities and Exchange Commission Form 10-Q Quarterly Report For the Quarterly Period Ended March 31, 2004*, Microsoft Corporation (May 3, 2004), p. 35, http://www.microsoft.com/msft/download/FY04/MSFT_3Q2004_10Q.doc.
2. *Opening the Open Source Debate: A White Paper*, Alexis de Tocqueville Institution (June 2002), http://www.adti.net/opensource.pdf.
3. The term "hacker" is used here to mean "a person who illegally gains access to and sometimes tampers with information in a computer system," as defined in *Merriam-Webster's Collegiate Dictionary, Eleventh Edition*, Merriam-Webster, Inc., Springfield, MA (2003), p. 559. Others prefer to use the term "cracker" to describe such a person.
4. CERT® Coordination Center, Software Engineering Institute, http://www.cert.org/.
5. Apache HTTP Server Project, The Apache Software Foundation, http://httpd.apache.org/ABOUT_APACHE.html.
6. B. Miller, L. Fredriksen, and B. So, "An Empirical Study of the Reliability of UNIX Utilities," *Communications of the ACM* **33**, No. 12, 32–44 (1990).
7. K. Hafner and J. Markoff, *Cyberpunk: Outlaws and Hackers on the Computer Frontier*, Simon & Schuster, Inc., New York (July 1992).

8. M. Eichin and J. Rochlis, "With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988," *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1–3, 1989, IEEE, New York (1989), pp. 326–343.
9. C. A. Kenwood, *A Business Case Study of Open Source Software*, MITRE Corporation (July 2001), http://www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/kenwood_software.pdf.
10. W. S. Humphrey, *The Quality Attitude*, Software Engineering Institute (2004), http://www.sei.cmu.edu/news-at-sei/columns/watts_new/watts-new.htm.
11. J. S. Clark, "Kryptonite Bic-picking," *New Cyclist* (October 1992).
12. J. Papoza, *eWeek Labs: Open Source Quicker at Fixing Flaws*, Ziff-Davis Media, Inc. (September 30, 2002), http://www.eweek.com/article2/0,3959,562226,00.asp.
13. B. Miller, D. Koski, C. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl, *Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services*, Technical Report, Computer Science Department, University of Wisconsin (November 1995), ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz-revisited.pdf.
14. *Linux versus Windows NT: The Verdict*, Bloor Research (October 1999), http://www.bloor-research.com/research_library.php?productid=245.
15. *Linux TCP/IP Inspection Report*, Reasoning, Inc. (2003), http://www.reasoning.com/downloads.html.
16. *Apache Open Source Inspection Report*, Reasoning, Inc. (2003), http://www.reasoning.com/downloads.html.
17. *June 2004 Web Server Survey*, Netcraft, Ltd. (June 6, 2004), http://news.netcraft.com/archives/2004/06/06/june_2004_web_server_survey.html.
18. *How Open Source and Commercial Software Compare: MySQL White Paper*, Reasoning, Inc. (2003), http://www.reasoning.com/downloads.html.
19. W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," *1970 Western Electronic Show and Convention (WESCON) Technical Papers* **14**, Los Angeles, CA, August 25–28, 1970, IEEE, New York (1970), pp. 1–9; reprinted in *Proceedings of the Ninth International Conference on Software Engineering (ICSE'87)*, Monterey, CA, March 30–April 2, 1987, IEEE Computer Society Press, Los Alamitos, CA (1987), pp. 328–338.
20. D. A. Wheeler, *Why Open Source Software/Free Software (OSS/FS)? Look at the Numbers!* (November 7, 2004), http://www.dwheeler.com/oss_fs_why.html.
21. Consumers—List of Evaluated Products, Common Criteria Project, http://www.commoncriteriaportal.org/public/consumer/index.php?menu=4.

Accepted for publication October 27, 2004.
Published online April 12, 2005.

Alan Boulanger
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (boulange@us.ibm.com).
Alan Boulanger joined IBM in October 1995 as a member of the Thomas J. Watson Global Security Analysis Laboratory. His research interests include network security, intrusion detection and remediation, applied penetration testing techniques, data forensics, telephony-related security, and emerging threat analysis. Since joining IBM, Mr. Boulanger has filed several information-security-related patents and has provided security-related technical assistance to the business community and to federal government agencies. He is an active member of both the New York and New England Electronic Crimes Task Forces. ■