Usage guide

This is a guide covering some aspects of using GrapheneOS. See the features page for a list of GrapheneOS features.

Table of contents

System navigation

By default, GrapheneOS uses gesture-based navigation. We recommend reading our guide on gesture navigation and giving it a chance even if you think you won't like it. Our experience is that when armed with the appropriate knowledge, the vast majority of users prefer the newer gesture navigation approach.

The system navigation mode can be configured in Settings > System > Gestures > Navigation mode. The same menu is also available in Settings > Accessibility > System controls > Navigation mode.

Gesture navigation

The bottom of the screen is a reserved touch zone for system navigation. A line is displayed in the center to show that the navigation bar is present across the entire bottom of the screen. In most apps, this area will display padding. Modern apps are able to tell the OS that they can handle not having the padding to display app content there while still not being able to receive touches from it. Open up the Settings app for an example.

Swiping up from the navigation bar while removing your finger from the screen is the Home gesture.

Swiping up from the navigation bar while holding your finger on the screen before releasing is the Recent Apps gesture. The most recently opened activity is always on the furthest right. Each step left goes one step back through the history of recently opened apps. Opening an app with the recent apps activity will place it on the furthest right in the recent apps order just like a new app being opened.

The recent apps activity has a screenshot button as an alternative to holding power and volume down while using an app.

Rather than opening the recent apps activity, you can swipe left on the navigation bar for the Previous app and swipe right for the Next app. This will not change the recent apps order. This is usually the best way to navigate through recent apps.

Swiping from either the left or the right of the screen within the app (not the navigation bar) is the Back gesture. Apps are supposed to avoid implementing conflicting gestures, but have the option to override this gesture if they truly need to get rid of it. Some legacy apps without active development of their UI still haven't addressed this despite gestures being the default for several years on Google Android. You can avoid triggering the back gesture in one of 2 easy ways: avoid swiping from right near the edge or hold your finger on the side of the screen for a moment before swiping. The more advanced option is using a diagonal swipe pointing sharply to the bottom of the screen since this will bypass the back gesture but will still trigger most app gestures. The advanced option is the most convenient approach once you get used to doing it.

The launcher uses a swipe up gesture starting anywhere on the screen to open the app drawer from the home screen. You need to start that gesture above the system navigation bar since any gesture starting on the navigation bar is handled by the OS as a system navigation gesture.

3-button navigation

3-button navigation is Android's oldest touchscreen-based navigation system. It will remain supported for the foreseeable future to provide accessibility for users unable to easily use the gestures. It's older than 2-button navigation but isn't considered a legacy feature.

A large row across the bottom of the screen is reserved for navigation buttons. The Back button is on the left, the Home button is in the center and the Recent Apps button is on the right.

In the recent apps activity, the most recently opened activity is always on the furthest right. Each step left goes one step back through the history of recently opened apps. Opening an app with the recent apps activity will place it on the furthest right in the recent apps order just like a new app being opened.

The recent apps activity has a screenshot button as an alternative to holding power and volume down while using an app.

Storage access

GrapheneOS inherits the same baseline approach to storage access as modern Android and extends it with our Storage Scopes feature as a fully compatible alternative to standard Android storage permissions. This section provides an overview of the standard approach to storage access primarily to provide context for explaining Storage Scopes.

There are two types of app-accessible storage:

- app-private ("internal") storage:
    - inaccessible to other apps
    - doesn't require any permission for full access
    - cleared when the app is uninstalled

- shared ("external") storage:
    - shared with other apps
    - access is regulated with permissions
    - files persist after uninstallation

  Android/data/ and Android/obb/ directories aren't considered to be parts of shared storage.

For modern apps, access to the shared storage is controlled in the following way:

- Without any storage permission, an app is allowed to:
    - create media files in standard directories (audio in Music/, Ringtones/, etc, images in Pictures/ and DCIM/, videos in DCIM/ and Movies/)
    - create files of any type (both media and non-media) in Documents/ and Download/
    - create new directories inside standard directories
    - rename/delete files that were created by the app itself
    - rename/delete directories if it can rename/delete all files within those directories
- Media access permission ("Allow access to media only", `READ_EXTERNAL_STORAGE`) allows the app to read media files that were created by other apps. Non-media files remain invisible to it. For apps targeting Android 13, the media access permission is split into `READ_MEDIA_IMAGES`, `READ_MEDIA_VIDEO` and `READ_MEDIA_AUDIO`.
- Media management special access permission ("Allow app to manage media", `MANAGE_MEDIA`) allows the app to delete and to rename media files created by other apps.
- "All files access" special access permission (`MANAGE_EXTERNAL_STORAGE`) allows the app to read, create, rename and delete files and directories of any type in any directory of the shared storage (including the root directory).

For legacy apps (those that target Android 9 or lower and those that target Android 10 and request legacy storage mode), storage access permissions have a different meaning:

- Without a storage permission, app is not allowed any type of access to any files or directories inside the shared storage.
- `READ_EXTERNAL_STORAGE` permission allows the app to read both media and non-media files in any directory.
- `WRITE_EXTERNAL_STORAGE` permission allows the app to create, rename and delete files (of any type) and directories in any directory of shared storage (including the root directory).

Additionally, both modern and legacy Android apps can open the system file picker interface to have the user store or load one or more files/directories on their behalf. This type of access doesn't require any of the permissions listed above. Using this approach gives the user control over where files are stored in their home directory and which files/directories can be used by the app. This is based on the Storage Access Framework (SAF) introduced in Android 4.4. SAF allows the user to grant access to files/directories in their home directory, external drives and also app-based storage providers such as network shares, cloud storage, an encrypted volume, an external drive with a filesystem the OS doesn't support for external drives, etc. This is the only way to use those app-based storage providers and modern Android has removed the legacy approach for accessing external drives.

Storage Scopes

GrapheneOS provides the Storage Scopes feature as a fully compatible alternative to the standard Android storage permissions. Storage Scopes can be enabled only if the app doesn't have any storage permission. Enabling Storage Scopes makes the app assume that it has all of storage permissions that were requested by it, despite not actually having any of them.

This means that the app can't see any of the files that were created by other apps. The app is still allowed to create files and directories, same as any other modern app that doesn't have any storage access permission.

Apps that would normally use the legacy storage mode are switched to the modern storage mode when Storage Scopes is enabled.

If the app requests the "All files access" permission (or is a legacy app that requests `WRITE_EXTERNAL_STORAGE` permission), then the write restrictions that are normally applied to apps that don't have a storage access permission are relaxed to provide the same write access that the app would have if it was granted the "All files access" permission. This is done to ensure compatibility with apps that, for example, create a new directory in the root of shared storage, or write a text file (eg lyrics.txt) to the Music/ directory (normally, only audio files can be placed there). No additional read access is granted to such apps, they still can see only their own files.

For all other apps, enabling Storage Scopes doesn't grant any additional storage access beyond what a modern app that doesn't have any storage permission already has.

Optionally, users can specify which of the files created by other apps the app can access. Access can be granted to a specific file or to all files in a directory. The standard SAF picker is used for this purpose in a special mode where it shows only shared storage files/directories.

The most significant limitation of Storage Scopes is the fact that the app will lose access to files that it created if it's uninstalled and then installed again, same as any other app that doesn't have a storage access permission. As a workaround, users can manually grant access to these files/directories via SAF picker.

Contact Scopes

On Android, contact access is controlled with an all-or-nothing Contacts permission, which grants both read and write access to all contacts stored on the device.

A lot of apps (e.g. popular messaging apps) refuse to work unless the Contacts permission is granted.

GrapheneOS provides the Contact Scopes feature as an alternative to granting the Contacts permission. Enabling Contact Scopes makes the app assume that it has the Contacts permission, despite not actually having it. By default, an app that has Contact Scopes enabled is not allowed any kind of contact access.

Optionally, read access can be granted to the following scopes:

- Contact data (phone number or email). Access to each type of number and email in a contact is granted separately. Access to the contact name is granted automatically.
- Single contact. Access is granted to all contact data, except contact photo.
- Contact group ("label"). Equivalent to granting access to all contacts in the group. Any contact can be in any number of contact groups.

The type and name of the account that the contact is stored in are fully hidden from the app. The name of the contact account is usually the same as the email address of that account.

When Contact Scopes is enabled, write access is fully blocked: the app is not allowed to edit any contact data, add or remove contacts, etc.

Accessibility

GrapheneOS includes all of the accessibility features from the Android Open Source Project and strives to fill in the gaps from not including Google apps and services. We include our own fork of the open source TalkBack accessibility service along with a Monochromacy option for the standard color correction menu.

GrapheneOS does not yet include a text-to-speech (TTS) service in the base OS due to limitations of the available options. Including one is planned in the future when a suitable option is available. RHVoice and eSpeak NG are both open source and are the most common choices by GrapheneOS users. Both of these work fine but have licensing issues. eSpeak NG has added Direct Boot based on our request for it, meaning it is able to function before the first unlock. RHVoice is missing Direct Boot and can't run before the first unlock. Installing and setting up either one of these or another TTS app will get TalkBack working. TalkBack itself supports Direct Boot and works before the first unlock but it needs to have a TTS app supporting it in order to do more than playing the activation sound before the first unlock. After installing a TTS service, you need to select it in the OS configuration to accept activating it. The OS will display one of them as already selected, but it won't simply work from being installed as that wouldn't be safe. This is the same as the stock OS but it comes with one set up already.

GrapheneOS disables showing the characters as passwords are typed by default. You can enable this in **Settings > Security & privacy > Privacy > Show passwords**.

Third party accessibility services can be installed and activated. This includes the ones made by Google. Most of these will work but some may have a hard dependency on functionality from Google Play services for some of their functionality or to run at all. Accessibility services are very powerful and we strongly recommend against using third party implementations if you can get by well without them. We plan to add safeguards in this area while still keeping them working without problematic barriers.

Auditor

See the tutorial page on the site for the attestation sub-project.

Updates

The update system implements automatic background updates. It checks for updates approximately once every six hours when there's network connectivity and then downloads and installs updates in the background. It will pick up where it left off if downloads are interrupted, so you don't need to worry about interrupting it. Similarly, interrupting the installation isn't a risk because updates are installed to a secondary installation of GrapheneOS which only becomes the active installation after the update is complete. Once the update is complete, you'll be informed with a notification and simply need to reboot with the button in the notification or via a normal reboot. If the new version fails to boot, the OS will be rolled back to the past version and the updater will attempt to download and install the update again.

The updater will use incremental (delta) updates to download only changes rather than the whole OS when one is available to go directly from the installed version to the latest version. As long as you have working network connectivity on a regular basis and reboot when asked, you'll almost always be on one of the past couple versions of the OS which will minimize bandwidth usage since incrementals will always be available.

The updater works while the device is locked / idle, including before the first unlock since it's explicitly designed to be able to run before decryption of user data.

Release changelogs are available in a section on the releases page.

Settings

The settings are available in the Settings app in System > System update.

The "Check for updates" option will manually trigger an update check as soon as possible. It will still wait for the configuration conditions listed below to be satisfied, such as being connected to the internet via one of the permitted network types.

The "Release channel" setting can be changed from the default Stable channel to the Beta channel if you want to help with testing. The Beta channel will usually simply follow the Stable channel, but the Beta channel may be used to experiment with new features.

The "Permitted networks" setting controls which networks will be used to perform updates. It defaults to using any network connection. It can be set to "Non-roaming" to disable it when the cellular service is marked as roaming or "Unmetered" to disable it on cellular networks and also Wi-Fi networks marked as metered.

The "Require battery above warning level" setting controls whether updates will only be performed when the battery is above the level where the warning message is shown. The standard value is at 15% capacity.

The "Require device to be charging" setting controls whether updates will only be performed when the device is charging.

Enabling the opt-in "Automatic reboot" setting allows the updater to reboot the device after an update once it has been idle for a long time. When this setting is enabled, a device can take care of any number of updates completely automatically even if it's left completely idle.

The "Notification settings" option is a shortcut to the System Updater notification settings which allows you to control notification settings from System Updater such as notification dot, lock screen, and noisy / silent notifications. These notifications include updater errors, progress, already up to date, and reboot prompts. By default all notifications are enabled.

Security

The update server isn't a trusted party since updates are signed and verified along with downgrade attacks being prevented. The update protocol doesn't send identifiable information to the update server and works well over a VPN / Tor. GrapheneOS isn't able to comply with a government order to build, sign and ship a malicious update to a specific user's device based on information like the IMEI, serial number, etc. The update server only ends up knowing the IP address used to connect to it and the version being upgraded from based on the requested incremental.

Android updates can support serialno constraints to make them validate only on a certain device but GrapheneOS rejects any update with a serialno constraint for both over-the-air updates (Updater app) and sideloaded updates (recovery).

Disabling

It's highly recommended to leave automatic updates enabled and to configure the permitted networks if the bandwidth usage is a problem on your mobile data connection. However, it's possible to turn off the update client by going to Settings > Apps, enabling Show system via the menu, selecting System Updater and disabling the app. If you do this, you'll need to remember to enable it again to start receiving updates.

Sideloading

Updates can be downloaded via the releases page and installed via recovery with adb sideloading. The zip files are signed and verified by recovery, just as they are by the update client within the OS. This includes providing downgrade protection, which prevents attempting to downgrade the version. If recovery didn't enforce these things, they would still be enforced via verified boot including downgrade protection and the attempted update would just fail to boot and be rolled back.

To install one by sideloading, first, boot into recovery. You may do this either by using `adb reboot recovery` from the operating system, or by selecting the "Recovery" option in the bootloader interface.

You should see the green Android lying on its back being repaired, with the text "No command" meaning that no command has been passed to recovery.

Next, access the recovery menu by holding down the power button and pressing the volume up button a single time. This key combination toggles between the GUI and text-based mode with the menu and log output.

Finally, select the "Apply update from ADB" option in the recovery menu and sideload the update with adb. For example:

```
adb sideload raven-ota_update-2021122018.zip
```

You do not need to have adb enabled within the OS or the host's ADB key whitelisted within the OS to sideload an update to recovery. Recovery mode does not trust the attached computer and this can be considered a production feature. Trusting a computer with ADB access within the OS is much different and exposes the device to a huge amount of attack surface and control by the trusted computer.

USB-C port and pogo pins control

Our **USB-C port and pogo pins** setting protects against attacks through USB-C or pogo pins while the OS is booted. For the majority of devices without pogo pins, the setting is labelled **USB-C port**.

The setting is available in **Settings > Security > Exploit protection**.

The setting has five modes:

- Off
- Charging-only
- Charging-only when locked
- Charging-only when locked, except before first unlock
- On

The default is **Charging-only when locked**, which significantly reduces attack surface when the device is locked. After locking, it blocks any new USB connections immediately and disables USB data once any current connections end.

For technical details on how this feature works using a combination of hardware and software protection, see the section on the features page.

Web browsing

GrapheneOS includes our Vanadium subproject providing privacy and security enhanced releases of Chromium. Vanadium is both the user-facing browser included in the OS and the provider of the WebView used by other apps to render web content. The WebView is the browser engine used by nearly all other apps embedding web content or using web technologies for other uses. It's also used by many minor web browsers not forking Chromium as a whole. These apps using the WebView benefit from a subset of the Vanadium hardening.

Vanadium was previously primarily focused on security hardening but we plan on adding assorted privacy and usability features. In the near future, we plan to add support for always incognito mode, improved state partitioning, backup/restore and many other features.

Chromium-based browsers like Vanadium provide the strongest sandbox implementation, leagues ahead of the alternatives. It is much harder to escape from the sandbox and it provides much more than acting as a barrier to compromising the rest of the OS. Site isolation enforces security boundaries around each site using the sandbox by placing each site into an isolated sandbox. It required a huge overhaul of the browser since it has to enforce these rules on all the IPC APIs. Site isolation is important even without a compromise, due to side channels. Browsers without site isolation are very vulnerable to attacks like Spectre. On mobile, due to the lack of memory available to apps, there are different modes for site isolation. Vanadium turns on strict site isolation, matching Chromium on the desktop, along with strict origin isolation.

Chromium has decent exploit mitigations, unlike the available alternatives. This is improved upon in Vanadium by enabling further mitigations, including those developed upstream but not yet fully enabled due to code size, memory usage or performance. For example, it enables type-based CFI like Chromium on the desktop, uses a stronger SSP configuration, zero initializes variables by default, etc. Some of the mitigations are inherited from the OS itself, which also applies to other browsers, at least if they don't do things to break them.

We recommend against trying to achieve browser privacy and security through piling on browser extensions and modifications. Most privacy features for browsers are privacy theater without a clear threat model and these features often reduce privacy by aiding fingerprinting and adding more state shared between sites. Every change you make results in you standing out from the crowd and generally provides more ways to track you. Enumerating badness via content filtering is not a viable approach to achieving decent privacy, just as AntiVirus isn't a viable way to achieving decent security. These are losing battles, and are at best a stopgap reducing exposure while waiting for real privacy and security features.

Vanadium will be following the school of thought where hiding the IP address through Tor or a trusted VPN shared between many users is the essential baseline, with the browser partitioning state based on site and mitigating fingerprinting to avoid that being trivially bypassed. The Tor Browser's approach is the only one with any real potential, however flawed the current implementation may be. This work is currently in a very early stage and it is largely being implemented upstream with the strongest available implementation of state partitioning. Chromium is using Network Isolation Keys to divide up connection pools, caches and other state based on site and this will be the foundation for privacy. Chromium itself aims to prevent tracking through mechanisms other than cookies, greatly narrowing the scope downstream work needs to cover. The focus is currently on research since we don't see much benefit in deploying bits and pieces of this before everything is ready to come together. At the moment, the only browser with any semblance of privacy is the Tor Browser but there are many ways to bypass the anti-fingerprinting and state partitioning. The Tor Browser's security is weak which makes the privacy protection weak. The need to avoid diversity (fingerprinting) creates a monoculture for the most interesting targets. This needs to change, especially since Tor itself makes people into much more of a target (both locally and by the exit nodes).

WebView-based browsers use the hardened Vanadium rendering engine, but they can't offer as much privacy and control due to being limited to the capabilities supported by the WebView widget. For example, they can't provide a setting for toggling sensors access because the feature is fairly new and the WebView WebSettings API doesn't yet include support for it as it does for JavaScript, location, cookies, DOM storage and other older features. For sensors, the Sensors app permission added by GrapheneOS can be toggled off for the browser app as a whole instead. The WebView sandbox also currently runs every instance within the same sandbox and doesn't support site isolation.

Avoid Gecko-based browsers like Firefox as they're currently much more vulnerable to exploitation and inherently add a huge amount of attack surface. Gecko doesn't have a WebView implementation (GeckoView is not a WebView implementation), so it has to be used alongside the Chromium-based WebView rather than instead of Chromium, which means having the remote attack surface of two separate browser engines instead of only one. Firefox / Gecko also bypass or cripple a fair bit of the upstream and GrapheneOS hardening work for apps. Worst of all, Firefox does not have internal sandboxing on Android. This is despite the fact that Chromium semantic sandbox layer on Android is implemented via the OS `isolatedProcess` feature, which is a very easy to use boolean property for app service processes to provide strong isolation with only the ability to communicate with the app running them via the standard service API. Even in the desktop version, Firefox's sandbox is still substantially weaker (especially on Linux) and lacks full support for isolating sites from each other rather than only containing content as a whole. The sandbox has been gradually improving on the desktop but it isn't happening for their Android browser yet.

Camera

GrapheneOS has the same camera capabilities and quality as the stock OS. It will match the stock OS when comparing the same app on each OS. GrapheneOS uses our own modern Camera app rather than the standard AOSP Camera app. GrapheneOS Camera is far better than any of the portable open source camera alternatives and even most proprietary camera apps including paid apps. On Pixels, Pixel Camera can be used as an alternative with more features. The section below has a detailed guide on using GrapheneOS Camera and the following section explains the remaining advantages of Pixel Camera on Pixels.

GrapheneOS Camera app

GrapheneOS includes our own modern camera app focused on privacy and security. It includes modes for capturing images, videos and QR / barcode scanning along with additional modes based on CameraX vendor extensions (Portrait, HDR, Night, Face Retouch and Auto) on devices where they're available (Pixels currently only have support for Night mode).

Modes are displayed as tabs at the bottom of the screen. You can switch between modes using the tab interface or by swiping left/right anywhere on the screen. The arrow button at the top of the screen opens the settings panel and you can close it by pressing anywhere outside the settings panel. You can also swipe down to open the settings and swipe up to close it. Outside of the QR scanning mode, there's a row of large buttons above the tab bar for switching between the cameras (left), capturing images and starting/stopping video recording (middle) and opening the gallery (right). The volume keys can also be used as an equivalent to pressing the capture button. While recording a video, the gallery button becomes an image capture button for capturing images.

Our Camera app provides the system media intents used by other apps to capture images / record videos via the OS provided camera implementation. These intents can only be provided by a system app since Android 11, so the quality of the system camera is quite important.

The app has an in-app gallery and video player for images/videos taken with it. It currently opens an external editor activity for the edit action. GrapheneOS comes with AOSP Gallery which provides an editor activity. You can install a nicer photo editor and the Camera app will be able to use it. We plan to replace AOSP Gallery with a standalone variant of the gallery we're developing for the Camera app in the future.

Using the default 4:3 aspect ratio for image capture is recommended since 16:9 is simply cropped output on all supported devices. A device oriented towards video recording might actually have a wider image sensor but that's not the case for Pixels or nearly any other smartphone.

Image capture uses lightweight HDR+ on all supported Pixels and HDRnet for the preview on 5th generation Pixels. Using the torch or camera flash will result in HDR+ being disabled which is why automatic flash isn't enabled by default. The lightweight HDR+ doesn't use as many frames as the more aggressive Pixel Camera HDR+. CameraX extensions will eventually provide support for an HDR mode with more aggressive HDR+ taking/combining more than only around 3 frames. It currently supports a Night mode providing the Night Sight variant of HDR+ inflating the light of the scene through combining the frames. Other fancy features like Portrait mode will also depend on CameraX extensions being provided in the future. There isn't a timeline for when additional CameraX extensions will be added.

Zooming via pinch to zoom or the zoom slider will automatically make use of the wide angle and telephoto cameras on Pixels. 5th and 6th generation Pixels (4a (5G), 5, 5a, 6, 6 Pro) have a wide angle camera for zooming out to under 1x to capture a much wider field of view. Images taken with the wide angle lens won't match the quality of the normal camera, especially with 6th generation Pixels. Flagship 4th generation Pixels (4, 4 XL) have a telephoto camera providing 2x optical zoom and the Pixel 6 Pro has one providing 4x optical zoom.

By default, continuous auto focus, auto exposure and auto white balance are used across the whole scene. Tapping to focus will switch to auto focus, auto exposure and auto white balance based on that location. The focus timeout setting determines the timeout before it switches back the default mode. The exposure compensation slider on the left allows manually tuning exposure and will automatically adjust shutter speed, aperture and ISO without disrupting lightweight HDR+ support. Further configuration / tuning will be provided in the future.

The QR scanning mode only scans within the scanning square marked on the screen. The QR code should be aligned with the edges of the square but can have any 90 degree orientation. Non-standard inverted QR codes are fully supported. It's a very quick and high quality QR scanner able to easily scan very high density QR codes from Pixels. Every 2 seconds, it will refresh auto focus, auto exposure and auto white balance on the scanning square. It has full support for zooming in and out. The torch can be toggled with the button at the bottom center. The auto toggle at the bottom left can be used to toggle scanning for all supported barcode types. Alternatively, you can select which barcode types it should scan via the menu at the top. It only scans QR codes by default since that provides quick and reliable scanning. Most other types of barcodes can result in false positives. Each enabled type will slow down the scanning and will make it more prone to false positives especially with difficult to scan barcodes such as a dense QR code.

Camera permission is the only one that's required. Images and videos are stored via the Media Store API so media/storage permissions aren't required. The Microphone permission is needed for video recording by default but not when including audio is disabled. Location permission is only needed if you explicitly enabling location tagging, which is an experimental feature.

By default, EXIF metadata is stripped for captured images and only includes the orientation. Stripping metadata for videos is planned but not supported yet. Orientation metadata isn't stripped since it's fully visible from how the image is displayed so it doesn't count as hidden metadata and is needed for proper display. You can toggle off stripping EXIF metadata in the More Settings menu opened from the settings dialog. Disabling metadata stripping will leave the timestamp, phone model, exposure configuration and other metadata. Location tagging is disabled by default and won't be stripped if you enable it.

Electronic Image Stabilization (EIS) is enabled by default on devices providing it via the Camera2 API and can be disabled using the video settings dialog.

Zero Shutter Lag (ZSL) is available as an opt-in toggle in More Settings and speeds up image capture for the Camera mode when flash is disabled.

Pixel Camera

Pixel Camera (previously known as Google Camera) can take full advantage of the available cameras and image processing hardware as it can on the stock OS and does not require GSF or sandboxed Google Play on GrapheneOS. Direct TPU and GXP access by Google apps including Pixel Camera is controlled by a toggle added by GrapheneOS and doesn't provide them with any additional access to data. The toggle exists for attack surface reduction. Every app can use the TPU and GXP via standard APIs including the Android Neural Networks API and Camera2 API regardless.

We aim to reduce the benefits of Pixel Camera compared to GrapheneOS Camera over time, especially on Pixels. Many features of Pixel Camera will end up being available for GrapheneOS Camera in the next year or so via CameraX extensions including more aggressive HDR+, Night Sight and Portrait. Video features such as slow motion and time lapse are likely further away than within the next year. These video features could potentially be provided via CameraX vendor extensions or could be implemented via our own post-processing of the video output. Panorama, Photo Sphere, Astrophotography, Motion Photos, Frequent Faces, Dual Exposure Controls, Google Lens, etc. aren't on the roadmap for GrapheneOS Camera. Video frame rate configuration and H.265 support should be available for GrapheneOS Camera in the near future via CameraX improvements along with DNG (RAW) support in the further future.

Exec spawning

GrapheneOS creates fresh processes (via exec) when spawning applications instead of using the traditional Zygote spawning model. This improves privacy and security at the expense of higher cold start app spawning time and higher initial memory usage. It doesn't impact runtime performance beyond the initial spawning time. It adds somewhere in the ballpark of 200ms to app spawning time on the flagship devices and is only very noticeable on lower-end devices with a weaker CPU and slower storage. The spawning time impact only applies when the app doesn't already have an app process and the OS will try to keep app processes cached in the background until memory pressure forces it to start killing them.

In the typical Zygote model, a template app process is created during boot and every app is spawned as a clone of it. This results in every app sharing the same initial memory content and layout, including sharing secrets that are meant to be randomized for each process. It saves time by reusing the initialization work. The initial memory usage is reduced due to copy-on-write semantics resulting in memory written only during initialization being shared between app processes.

The Zygote model weakens the security provided by features based on random secrets including Address Space Layout Randomization (ASLR), stack canaries, heap canaries, randomized heap layout and memory tags. It cripples these security features since every app has the values for every other app and the values don't change for fresh app processes until reboot. Much of the OS itself is implemented via non-user-facing apps with privileges reserved for OS components. The Zygote template is reused across user profiles, so it also provides a temporary set of device identifiers across profiles for each boot via the shared randomized values.

This feature can be disabled via Settings > Security & privacy > Exploit protection > Secure app spawning if you prefer to have faster cold start app spawning time and lower app process memory usage instead of the substantial security benefits and the removal of the only known remaining direct device identifiers across profiles (i.e. not depending on fingerprinting global configuration, available storage space, etc. or using side channels).

Bugs uncovered by security features

GrapheneOS substantially expands the standard mitigations for memory corruption vulnerabilities. Some of these features are designed to directly catch the memory corruption bugs either via an explicit check or memory protection and abort the program in order to prevent them from being exploited. Other features mitigate issues a bit less directly such as zeroing data immediately upon free, isolated memory regions, heap randomization, etc. and can also lead to latent memory corruption bugs crashing instead of the program continuing onwards with corrupted memory. This means that many latent memory corruption bugs in apps are caught along with some in the OS itself. These bugs are not caused by GrapheneOS, but rather already existed and are uncovered by the features. The features are aimed at preventing or hindering exploits, not finding bugs, but they do that as part of doing their actual job.

Similarly, some of the other privacy and security improvements reduce the access available to applications and they may crash. Some of these features are always enabled under the hood, while others like the Network and Sensors toggles are controlled by users via opt-in or opt-out toggles. Apps may not handle having access taken away like this, although it generally doesn't cause any issues as it's all designed to be friendly to apps and fully compatible rather than killing the application when it violates the rules.

You can enable our exploit protection compatibility mode via **Settings > Apps > *APP* > Exploit protection compatibility mode**. The exploit protection compatibility mode toggle will:

- Switch from hardened_malloc to Android's standard allocator (Scudo)
- Reduce address space size from 48 bit to Android's standard 39 bit
- Disable memory tagging, unless the app has opted-in to it (only on compatible devices)
- Allow native debugging (ptrace) access

All of these changes apply only to the selected app and can be toggled separately.

If you run into an application aborting, try to come up with a process for reproducing the issue and then capture a bug report via the 'Take bug report' feature in Developer options. Report an issue to the GrapheneOS OS issue tracker and email the bug report capture zip to contact@grapheneos.org with the issue tracker number in the subject like "Bug report capture for issue #104". The bug report capture includes plain text 'tombstones' with logs, tracebacks, address space layout, register content and a tiny bit of context from memory from areas that are interesting for debugging. This may contain some sensitive data. Feel free to provide only the tombstone for the relevant crash and filter out information you don't want to send. However, it will be more difficult to debug if you provide less of the information. If the app doesn't work with sensitive information, just send the whole tombstone.

Wi-Fi privacy

Wi-Fi on GrapheneOS is very privacy-friendly and is essentially anonymous as long as apps do not leak uniquely identifying information to the network. GrapheneOS avoids allowing itself to be fingerprinted as GrapheneOS, other than connections which are documented (see the default connections FAQ) and can be easily disabled or forced through a VPN service.

Scanning

MAC randomization is always performed for Wi-Fi scanning. Pixel phones have firmware support for scanning MAC randomization going significantly beyond a naive implementation. On many other devices, there are identifiers exposed by Wi-Fi scanning beyond the MAC address such as the packet sequence number and assorted identifying information in the probe requests.

Avoid using hidden APs (i.e. APs not broadcasting their SSID) since all known hidden SSIDs end up being broadcast as part of scanning for networks to find them again. SSIDs are not broadcast for standard non-hidden APs. Hidden APs are only hidden when no devices are connected. It makes little sense as a privacy feature, especially for a non-mobile AP where knowing the AP exists can't be used for tracking it since it doesn't move. The feature reduces your privacy rather than increasing it. If you need to use a hidden AP, make sure to delete the saved network afterwards.

Wi-Fi and Bluetooth scanning for improving location detection are disabled by default, unlike the stock OS. These can be toggled in Settings > Location > Location services > Wi-Fi and Bluetooth scanning. These features enable scanning even when Wi-Fi or Bluetooth is disabled, so these need to be kept disabled to fully disable the radios when Wi-Fi and Bluetooth are disabled. GrapheneOS itself doesn't currently include a supplementary location service based on Wi-Fi and Bluetooth scanning. These options impact whether apps such as sandboxed Google Play are able to use the functionality if you grant them the Location permission. GrapheneOS plans to eventually include an OS service based on local databases rather than a network-based service giving the user's location to a server whenever location is being used.

Associated with an Access Point (AP)

Associated MAC randomization is performed by default. This can be controlled per-network in Settings > Network & internet > Internet > *NETWORK* > Privacy.

In the stock OS, the default is to use a unique persistent random MAC address for each network. It has 2 options available: "Use randomized MAC (default)" and "Use device MAC". In GrapheneOS, the default is generating a new random MAC address when connecting to a network. It has 3 options available: "Use per-connection randomized MAC (default)", "Use per-network randomized MAC" and "Use device MAC".

In rare cases, broken routers are unable to accept new clients once their DHCP table is full instead of clearing the last recently used entry. You can work around this by manually clearing the DHCP table via the router administration page and can switch to the per-network randomized MAC mode to avoid triggering the issue again. This would prevent a router being used in any situation where many clients naturally come and go even without per-connection MAC randomization and is not generally an issue for any modern routers. Per-connection MAC randomization only makes it more likely to find a one of the rare routers with this issue.

The DHCP client uses the anonymity profile rather than sending a hostname so it doesn't compromise the privacy offered by MAC randomization. When the per-connection MAC randomization added by GrapheneOS is being used, DHCP client state is flushed before reconnecting to a network to avoid revealing that it's likely the same device as before.

GrapheneOS also disables support for stable link-local IPv6 addresses, since these have the potential to be used as identifiers. It's more sensible to use typical link-local address generation based on the (randomized) MAC address since link-local devices have access to both. As of Android 11, Android only uses stable link-local privacy addresses when MAC randomization is disabled, so we no longer need to disable the feature.

Network location

For more information on the network location feature and how it works, see the features section.

The setting can be found at **Settings > Location > Location services > Network location**. The available options are using Apple's network location service, or a GrapheneOS proxy to that service.

To use this feature, you need to make sure that you either have Wi-Fi enabled, or that Wi-Fi scanning is enabled at **Settings > Location > Location services > Wi-Fi scanning**, or you have cell reception (far less accurate). Enabling the Wi-Fi scanning setting will allow you to use Wi-Fi-based network location even if you disable the global Wi-Fi toggle.

LTE-only mode

If you have a reliable LTE connection from your carrier, you can reduce attack surface by disabling 2G, 3G and 5G connectivity in **Settings > Network & internet > SIMs >** ***SIM*** **> Preferred network type**. Traditional voice calls will only work in the LTE-only mode if you have either an LTE connection and VoLTE (Voice over LTE) support or a Wi-Fi connection and VoWi-Fi (Voice over Wi-Fi) support. VoLTE / VoWi-Fi works on GrapheneOS for most carriers unless they restrict it to carrier phones. Some carriers may be missing VoWi-Fi due to us not including their proprietary apps. Please note that AT&T users may see "5Ge" being used when LTE Only mode is enabled as AT&T intentionally mislabel LTE services as "5Ge" to mislead users.

This feature is not intended to improve the confidentiality of traditional calls and texts, but it might somewhat raise the bar for some forms of interception. It's not a substitute for end-to-end encrypted calls / texts or even transport layer encryption. LTE does provide basic network authentication / encryption, but it's for the network itself. The intention of the LTE-only feature is only hardening against remote exploitation by disabling an enormous amount of both legacy code (2G, 3G) and bleeding edge code (5G).

Sandboxed Google Play

GrapheneOS has a compatibility layer providing the option to install and use the official releases of Google Play in the standard app sandbox. Google Play receives absolutely no special access or privileges on GrapheneOS as opposed to bypassing the app sandbox and receiving a massive amount of highly privileged access. Instead, the compatibility layer teaches it how to work within the full app sandbox. It also isn't used as a backend for the OS services as it would be elsewhere since GrapheneOS doesn't use Google Play even when it's installed.

Since the Google Play apps are simply regular apps on GrapheneOS, you install them within a specific user or work profile and they're only available within that profile. Only apps within the same profile can use it and they need to explicitly choose to use it. It works the same way as any other app and has no special capabilities. As with any other app, it can't access data of other apps and requires explicit user consent to gain access to profile data or the standard permissions. Apps within the same profile can communicate with mutual consent and it's no different for sandboxed Google Play.

Sandboxed Google Play is close to being fully functional and provides near complete compatibility with the app ecosystem depending on Google Play. Only a small subset of privileged functionality which we haven't yet ported to different approaches with our compatibility layer is unavailable. Some functionality is inherently privileged and can't be provided as part of the compatibility layer.

The vast majority of Play services functionality works perfectly including dynamically downloaded / updated modules (dynamite modules) and functionality provided by modular app components such as Google Play Games. By default, location requests are rerouted to a reimplementation of the Play geolocation service provided by GrapheneOS. You can disable rerouting and use the standard Play services geolocation service instead if you want the Google network location service and related features. This shouldn't be needed outside of testing or obscure use cases, since GrapheneOS provides its own opt-in Network Location feature.

Our compatibility layer includes full support for the Play Store. Play Store services are fully available including in-app purchases, Play Asset Delivery, Play Feature Delivery and app / content license checks. It can install, update and uninstall apps with the standard approach requiring that the user authorizes it as an app source and consents to each action. It will use the standard Android 12+ unattended update feature to do automatic updates for apps where it was the last installer.

Installation

The simplest approach is to only use the Owner user profile. Apps installed in the Owner profile are sandboxed the same way as everywhere else and don't receive any special access. If you want to choose which apps use Google Play rather than making it available to all of them, install it in a separate user or work profile for apps depending on Google Play. You could also do it the other way around, but it makes more sense to try to use as much as possible without Google Play rather than treating not using it as the exceptional case.

To install sandboxed Google Play, open our App Store, select Google Play services and install it. This will install both Google Play services and Google Play Store which are interdependent. Existing installs of sandboxed Google Play from before Android 15 will also have Google Services Framework installed and it shouldn't be removed.

Google Play services and Google Play Store are updated through the App Store.

You should give a battery optimization exception to Google Play services for features like push notifications to work properly in the background. It isn't needed for the Google Play Store app.

Signing in into a Google account is optional, unless you want to use features depending on being signed into an account. For example, some apps use Google account authentication instead of their accounts having a username and password. The Play Store requires being signed into an account in order to install apps or use in-app purchases. This is still true even for an alternate frontend to the Play Store. Aurora Store still requires an account but fetches shared account credentials from Aurora Store's service by default.

The Play Store provides many services used by apps including Play Asset Delivery, Play Feature Delivery, in-app purchases and license checks for paid apps. The Play Store app is also the most secure way to install and update apps from the Play Store.

Our compatibility layer has support for Play Games Services which you can obtain by installing Google Play Games from the Play Store. Many games on the Play Store depend on having Google Play Games installed.

Configuration

The compatibility layer has a configuration menu available at Settings > Apps > Sandboxed Google Play.

By default, apps using Google Play geolocation are redirected to our own implementation on top of the standard OS geolocation service. You don't need to grant any permissions to Google Play or change any settings for working location in apps using Google Play geolocation due to our rerouting feature.

This is not recommended unless you're testing something or have an obscure use case since our network location feature can be used instead, but if you want to use Google's network location service to provide location estimates without satellite reception, you can follow these steps:

1. Disable the "Reroute location requests to OS APIs" toggle.
2. Grant "Allow all the time" Location access to Google Play services along with the Nearby Devices permission for it to have all the access it needs.
3. Use the "Google Location Accuracy" link from the sandboxed Google Play configuration menu to access the Google Play services menu for opting into their network location service.
4. To take advantage of Wi-Fi and Bluetooth scanning, you also need to enable the scanning toggles in Settings > Location > Location services which are disabled by default and control whether apps with the required permissions can scan when Wi-Fi and Bluetooth are otherwise disabled.

Re-routing location to the OS geolocation service will use more power than using the Google Play geolocation service unless you enable our opt-in network location feature.

The Google Location Accuracy and Google settings activities would normally be integrated into the OS but we don't include any of the standard Google Play integration so there needs to be an app providing a way to access them.

The menu also provides links to this usage guide, Play services system settings, Play Store system settings and Google settings. The Play services and Play Store system settings are only included for convenience since they can be accessed the same way as any other app via Settings > Apps.

Location sharing

Location sharing in Google Maps is implemented in Google Play services. It requires enabling both the Location permission with "Allow all the time" and the Physical activity permission for Google Play services. Our location rerouting feature reimplementing the Google Play location service using the OS location service does not apply to Google Play services itself, so Google Play services won't use the OS network location service. You can use the instructions above for enabling network location via Google Play if you want this. You could continue using rerouting for other apps or disable it to avoid needing 2 separate network location services enabled.

Limitations

Our compatibility layer has to be expanded on a case-by-case basis to teach Play services to work as a regular app without any of the invasive access and integration it expects. In many cases, it doesn't truly need the access or we can teach it to use the regular approach available to a normal app. In some cases, the functionality it offers fundamentally requires privileged access and cannot be supported. For example, Android Auto cannot be supported by default as part of the baseline compatibility layer, but is supported as an extension of it with dedicated toggles for the functionality that's not possible to implement another way. The same applies to other highly invasive OS integration / control or privileged access to hardware. Our compatibility layer is a very actively developed work in progress and most of the remaining unavailable functionality is quickly becoming supported. In the future, we can also support redirection for more APIs such as FIDO2 rather than only the geolocation service.

Functionality depending on the OS integrating Play services and using it as a backend is unavailable. An OS integrating Play uses it as the backend for OS services such as geolocation. GrapheneOS never uses it as the backend/provider for OS services. In cases such as text-to-speech (TTS) where the OS allows the user to choose the provider, Play services can often be used. It's on a level playing field with other apps on GrapheneOS.

eSIM support

By default, GrapheneOS has always shipped with baseline support for eSIM, where users can use any eSIMs installed previously on the device. However, in order to manage and add eSIMs, proprietary Google functionality is needed. This is fully disabled by default.

eSIM support on GrapheneOS doesn't require any dependency on Google Play, and never shares data to Google Play even when installed. It won't connect to a Google service unless the carrier uses one themselves.

eSIM support can be enabled in **Settings > Network & internet > eSIM support**. The toggle is persistent across every boot.

Note that if the eSIM installation process does not progress past the "Checking network info..." stage despite having a stable Internet connection, you may need to call the USSD code `*#*#4636#*#*` and then enable DSDS in the menu that is presented.

If an eSIM locked with a PIN is used, it is recommended to leave the eSIM support toggle enabled even after activating the eSIM. This will allow you to disable the eSIM on the lockscreen in case the PIN is forgotten. If the eSIM support toggle is disabled and the PIN is forgotten, there is no way to access the device unless the PUK is provided.

Android Auto

GrapheneOS provides an option to install and use the official releases of Android Auto.

Android Auto requires privileged access in order to work. GrapheneOS uses an extension of the sandboxed Google Play compatibility layer to make Android Auto work with a reduced level of privileges.

To install Android Auto, use the GrapheneOS App Store. Android Auto can't be installed through the Play Store or other app sources. Android Auto depends on sandboxed Google Play, you'll be prompted to install it if it's not already installed.

After installation, Android Auto has to be set up from the Settings > Apps > Sandboxed Google Play > Android Auto configuration screen, which contains permission toggles, links to related configuration screens, configuration tips, and links to optional Android Auto dependencies.

The permission toggles ask for a confirmation before turning on. The confirmation popup explains what access each permission toggle provides.

By default, Android Auto is not granted any kind of privileged access. It's treated the same way other apps are treated.

In order to work, Android Auto has to be granted baseline permissions for wired or wireless Android Auto. Wired Android Auto requires far less access than wireless Android Auto does. Baseline permissions are controlled by the "Allow permissions for wired / wireless Android Auto" toggles.

For some cars, baseline permissions for wireless Android Auto are needed even when using wired Android Auto. Therefore, if wired Android Auto is unable to connect to the car with only wired permissions granted, try granting wireless permissions instead.

To forward notifications from the device to the car, Android Auto has to be allowed notification access. The notification access settings are linked below the permission toggles.

In order to show up in the Android Auto car interface, apps have to be installed from the Play Store and include Android Auto support.

See Google's Android Auto Help pages for further Android Auto setup steps and usage instructions.

Banking apps

Banking apps are a particularly problematic class of apps for compatibility with alternate operating systems. Some of these work fine with any GrapheneOS configuration but most of them have extensive dependencies on Play services. For many of these apps, it's enough to set up the GrapheneOS sandboxed Google Play feature in the same profile. Unfortunately, there are further complications not generally encountered with non-financial apps.

Many of these apps have their own crude anti-tampering mechanisms trying to prevent inspecting or modifying the app in a weak attempt to hide their code and API from security researchers. GrapheneOS allows users to disable **Native code debugging** via a toggle in **Settings > Security & privacy > Exploit protection** to improve the app sandbox and this can interfere with apps debugging their own code to add a barrier to analyzing the app. You should try enabling this again if you've disabled it and are encountering compatibility issues with these kinds of apps.

Banking apps are increasingly using Google's SafetyNet attestation service to check the integrity and certification status of the operating system. GrapheneOS passes the `basicIntegrity` check but isn't certified by Google so it fails the `ctsProfileMatch` check. Most apps currently only enforce weak software-based attestation which can be bypassed by spoofing what it checks. GrapheneOS doesn't attempt to bypass the checks since it would be very fragile and would repeatedly break as the checks are improved. Devices launched with Android 8 or later have hardware attestation support which cannot be bypassed without leaked keys or serious vulnerabilities so the era of being able to bypass these checks by spoofing results is coming to an end regardless.

The hardware attestation feature is part of the Android Open Source Project and is fully supported by GrapheneOS. SafetyNet attestation chooses to use it to enforce using Google certified operating systems. However, app developers can use it directly and permit other properly signed operating systems upholding the security model. GrapheneOS has a detailed guide for app developers on how to support GrapheneOS with the hardware attestation API. Direct use of the hardware attestation API provides much higher assurance than using SafetyNet so these apps have nothing to lose by using a more meaningful API and supporting a more secure OS.

A 3rd party community-sourced effort containing banking app compatibility information is maintained by PrivSec.dev. GrapheneOS does not make any guarantees regarding the list's validity.

App link verification

Android apps can declare associations with domains in order to handle those URLs in the app automatically. For security reasons, app links are disabled by default to prevent apps intercepting arbitrary URLs. First party apps associated with a domain are expected to be authorized by the domain. Apps can ask for their app links to be verified by the OS by marking them with `autoVerify` in their manifest. The OS will securely confirm that the domain authorizes the app to handle the domain's URLs. Users can also manually enable an app's link associations via **Settings > Apps >** ***APP*** **> Open by default > Add link**. Apps can ask users to enable the associations and send them to this page in the Settings app.

As an example, the first party YouTube app will have the app links verified by the OS automatically while the NewPipe app requires manually enabling handling links for YouTube and other sites.

Verification of app links by the OS is done by the Intent Filter Verification Service system app. It will use an HTTPS GET request to fetch `https://example.com/.well-known/assetlinks.json` in order to process a request to verify that an app can handle `example.com` links. The app needs to have their app id and signing keys authorized by the domain in order for the verification to succeed.

These network requests by the Intent Filter Verification Service to verify app associations with domains are commonly confused for network requests made by the apps. It's simply an HTTPS GET request without identifying information and doesn't offer a communication channel with the app. Redirects won't be followed so there will be a single request for each attempt to verify a domain.

If you don't want automatic app link verification, you can disable the Network permission added by GrapheneOS for the Intent Filter Verification Service system app. In the future, we may provide a way to disable verification directly instead of stopping it from working. It will make heavily throttled attempts to verify a domain after the check failed which won't negatively impact battery life due to the conservative JobScheduler-based implementation.

For more details, see the developer documentation on app link verification.

Carrier functionality

GrapheneOS aims to work with all carriers that are officially supported by Google on the stock operating system on Pixel devices. Wi-Fi Calling, VoLTE, Visual Voicemail, MMS, SMS, Calling and 5G (SA and NSA) all are supported, however some functionality may not be usable due to Google not supporting carriers on the stock OS officially or due to GrapheneOS not shipping proprietary apps required in order for this functionality to work on some carriers. GrapheneOS extracts CarrierConfigs, APNs, modem configurations, Visual Voicemail configurations and MMS configurations from the stock operating system to ensure users get easy carrier support that "just works".

Please note that in some regions, LTE is referred to as 4G.

Generally 5G, SMS, MMS, Calls and VoLTE will work fine on GrapheneOS with officially supported carriers by Google. Wi-Fi calling may vary due to a reliance on proprietary Google apps which GrapheneOS does not ship.

If you are having issues with Visual Voicemail, please be aware that AT&T USA users are unable to use this feature currently due to a lack of AOSP support. Other carriers are done on a best effort basis. We would suggest using Google Dialer with sandboxed Google Play if you are unable to get this feature working.

If you are having problems sending or receiving SMS/MMS messages, we suggest that you perform the following steps:

- Deregister your phone number from Apple iMessage
- Deregister your phone number from Google Chat Features
- Deregister your phone number from your carrier's RCS service (Not all carriers have this)

If you continue to have problems despite following the instructions above or you have another carrier related issue, we suggest that you perform the following steps:

- Some carriers require you to explicitly opt in to use services such as Wi-Fi calling. Consult your carrier's documentation on the process for this or contact them.
- **Reset Mobile Network Settings** in **Settings > System > Reset options** and then reboot the device.
- USA users only: You may need to request your carrier to enable CDMA-less mode if you have issues.
- Follow your carrier's instructions for setting up APNs, this can be found in **Settings > Network & internet > SIMs > *SIM* > Access Point Names**
- If calls do not work and you have LTE-only mode enabled, try toggling it off. If "Allow 2G" is disabled, try toggling it back on. Your carrier may not properly support VoLTE.
- As a last resort you may need to ask your carrier for a replacement SIM card.

Some carriers may restrict functionality, such as VoLTE, on imported Pixel devices as they only whitelist the IMEI ranges of Pixel device SKUs which were sold locally. You can check your SKU on GrapheneOS by going to **Settings > About phone > Model > Hardware SKU** and using the official Google documentation. You should check if such functionality works on the stock OS to troubleshoot. It is not possible to change the IMEI on a production device and GrapheneOS cannot add support for it since the hardware doesn't support it.

Android 12 introduced support for the GSMA TS.43 standard where provisioning for VoLTE, VoNR and Wi-Fi calling may be handled by Google Firebase. Currently it is very rare for carriers to be using Firebase to handle this. It is unlikely, but technically possible, that with such carriers you may be required to install sandboxed Google Play in order to obtain the former mentioned functionality where the carrier has not implemented fallback provisioning via SMS. Currently only US Cellular, Orange France, Cspire US, and Cellcom US are using this standard.

GrapheneOS includes bypasses for carrier restrictions on APN editing, tethering via USB, Ethernet, Bluetooth and Wi-Fi and the ability to disable 2G (only on 6th generation Pixel devices onwards) actions which would not necessarily have been possible on the stock operating system.

GrapheneOS

- Forum
- Discord
- Matrix
- Hiring
- X
- Mastodon
- Bluesky
- GitHub
- Reddit
- LinkedIn