Features overview

GrapheneOS is a private and secure mobile operating system with great functionality and usability. It starts from the strong baseline of the Android Open Source Project (AOSP) and takes great care to avoid increasing attack surface or hurting the strong security model. GrapheneOS makes substantial improvements to both privacy and security through many carefully designed features built to function against real adversaries. The project cares a lot about usability and app compatibility so those are taken into account for all of our features.

GrapheneOS is focused on substance rather than branding and marketing. It doesn't take the typical approach of piling on a bunch of insecure features depending on the adversaries not knowing about them and regressing actual privacy/security. It's a very technical project building privacy and security into the OS rather than including assorted unhelpful frills or bundling subjective third party apps choices.

GrapheneOS is also hard at work on filling in gaps from not bundling Google apps and services into the OS. We aren't against users using Google services but it doesn't belong integrated into the OS in an invasive way. GrapheneOS won't take the shortcut of simply bundling a very incomplete and poorly secured third party reimplementation of Google services into the OS. That wouldn't ever be something users could rely upon. It will also always be chasing a moving target while offering poorer security than the real thing if the focus is on simply getting things working without great care for doing it robustly and securely.

This page provides an overview of currently implemented features differentiating GrapheneOS from AOSP. It doesn't document our many historical features that are no longer included for one reason or another. Many of our features were implemented in AOSP, Linux, LLVM and other projects GrapheneOS is based on and those aren't listed here. In many cases, we've been involved in getting those features implemented in core infrastructure projects.

Table of contents

LTE-only mode
Wi-Fi privacy
Private screenshots
Closed device identifier leaks
PIN scrambling
Privacy by default
Supports longer passwords
Auto reboot
Duress PIN/Password
More secure fingerprint unlock
Improved user profiles
More user profiles
End session
Disabling app installation
Install available apps
Notification forwarding
GrapheneOS app repository
Vanadium: hardened WebView and default browser
Auditor app and attestation service
GrapheneOS Camera
GrapheneOS PDF Viewer
Encrypted backups
Location data access indicator
User installed apps can be disabled
Other features
Services
Project

GrapheneOS

These are the features of GrapheneOS beyond what's provided by version 14 of the Android Open Source Project. It only covers our improvements to AOSP and not baseline features. This section doesn't list features like the standard app sandbox, verified boot, exploit mitigations (ASLR, SSP, Shadow Call Stack, Control Flow Integrity, etc.), permission system (foreground-only and one-time permission grants, scoped file access control, etc.) and so on but rather only our improvements to modern Android. We plan on providing a separate page listing the improvements we've contributed to Android since those features aren't listed here despite being a substantial portion of our overall historical work.

Defending against exploitation of unknown vulnerabilities

GrapheneOS is heavily focused on protecting users against attackers exploiting unknown (0 day) vulnerabilities. Patching vulnerabilities doesn't protect users before the vulnerability is known to the vendor and has a patch developed and shipped.

Unknown (0 day) vulnerabilities are much more widely used than most realize to exploit users not just in targeted attacks but in broad deployments. Project Zero maintains a spreadsheet tracking zero day exploitation detected in the wild. This is only a peek into what's happening since it only documents cases where the attackers were caught exploiting users, often because the attacks are not targeted but rather deployed on public websites, etc.

The first line of defense is attack surface reduction. Removing unnecessary code or exposed attack surface eliminates many vulnerabilities completely. GrapheneOS avoids removing any useful functionality for end users, but we can still disable lots of functionality by default and require that users opt-in to using it to eliminate it for most of them. An example we landed upstream in Android is disallowing using the kernel's profiling support by default, since it was and still is a major source of Linux kernel vulnerabilities. Profiling is now only exposed to apps for developers who enable developer tools, enable the Android Debug Bridge (ADB) and then use profiling tools via ADB. It's also only enabled until the next boot. This isn't listed below since it's one of the features we got implemented in Android itself.

The next line of defense is preventing an attacker from exploiting a vulnerability, either by making it impossible, unreliable or at least meaningfully harder to develop. The vast majority of vulnerabilities are well understood classes of bugs and exploitation can be prevented by avoiding the bugs via languages/tooling or preventing exploitation with strong exploit mitigations. In many cases, vulnerability classes can be completely wiped out while in many others they can at least be made meaningfully harder to exploit. Android does a lot of work in this area, and GrapheneOS has helped to advance this in Android and the Linux kernel. It takes an enormous amount of resources to develop fundamental fixes for these problems and there's often a high performance, memory or compatibility cost to deploying them. Mainstream operating systems usually don't prioritize security over other areas. GrapheneOS is willing to go further, thus we offer toggles for users to choose the compromises they prefer instead of forcing it on them. In the meantime, weaker less complete exploit mitigations can still provide meaningful barriers against attacks as long as they're developed with a clear threat model. GrapheneOS is heavily invested in many areas of developing these protections: developing/deploying memory safe languages/libraries, static/dynamic analysis tooling and many kinds of mitigations.

The final line of defense is containment through sandboxing at various levels: fine-grained sandboxes around a specific context like per site browser renderers, sandboxes around a specific component like Android's media codec sandbox and app/workspace sandboxes like the Android app sandbox used to sandbox each app which is also the basis for user/work profiles. GrapheneOS improves all of these sandboxes through fortifying the kernel and other base OS components along with improving the sandboxing policies.

Preventing an attacker from persisting their control of a component or the OS/firmware through verified boot and avoiding trust in persistent state also helps to mitigate the damage after a compromise has occurred.

Remote code execution vulnerabilities are the most serious and allow an attacker to gain a foothold on the device or even substantial control over it remotely. Local code execution vulnerabilities allow breaking out of a sandbox including the app sandbox or browser renderer sandbox after either compromising an app/browser renderer remotely, compromising an app's supply chain or getting the user to install a malicious app. Many other kinds of vulnerabilities exist but most of what we're protecting against falls into these two broad categories.

The vast majority of local and remote code execution vulnerabilities are memory corruption bugs caused by memory unsafe languages or rare low-level unsafe code in an otherwise memory safe language. Most of the remaining issues are caused by dynamic code execution/loading features. Our main focus is on preventing or raising the difficulty of exploiting memory corruption bugs followed by restricting dynamic code execution both to make escalation from a memory corruption bug harder and to directly mitigate bugs caused by dynamic code loading/generation/execution such as a JIT compiler bug or a plugin loading vulnerability.

Attack surface reduction

- Greatly reduced remote, local and proximity-based attack surface by stripping out unnecessary code, making more features optional and disabling optional features by default (NFC, Bluetooth, UWB, etc.), when the screen is locked (USB, USB-C, pogo pins, camera access) and optionally after a timeout (Bluetooth, Wi-Fi).
- Native debugging (ptrace) access is blocked for all bundled apps to reduce local attack surface. ptrace access is allowed by default for user installed apps for compatibility, with an option to block it by default. In both cases, ptrace access can be blocked or allowed manually on a per-app basis. When an app that is blocked from accessing ptrace tries to use it, GrapheneOS shows a notification that links the per-app native debugging settings screen.

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. We have a less advanced version of this feature on devices launched prior to October 2021 (Pixel 5a and earlier) since it requires a hardware-specific implementation with driver changes.

The feature 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 through either USB-C and pogo pins at both the hardware level via configuring the USB controller and also at

the OS level in the kernel to provide a second layer of defense. It disables the data lines at a hardware level as soon as the existing connections end which happens right away if there were no USB connections. It also disables USB-C alternate modes including DisplayPort at both the OS and hardware level.

Our implementation is far more secure than Android's standard USB HAL toggle available to device admin apps. The standard feature only disables high level USB handling in the OS. It doesn't block new USB connections or disable the data lines at a hardware level. It also leaves the handling of the USB-C and pogo pins protocols enabled in the OS, and it doesn't disable the USB-C alternate modes. The standard feature is also either blocking or not blocking USB at a high level, without the ability to block new connections and disable USB only once the existing connections end. Other operating systems trying to implement a similar feature via the standard toggle end up continuing to allow new USB connections in the OS until all connections end instead of the two-phase approach we use for our two Charging-only when locked modes.

Exploit mitigations

- Hardened app runtime
- Secure application spawning system avoiding sharing address space layout and other secrets across applications
- Hardened libc providing defenses against the most common classes of vulnerabilities (memory corruption)
- Our own hardened malloc (memory allocator) leveraging modern hardware capabilities to provide substantial defenses against the most common classes of vulnerabilities (heap memory corruption) along with reducing the lifetime of sensitive data in memory. The hardened_malloc README has extensive documentation on it. The hardened_malloc project is portable to other Linux-based operating systems and is being adopted by other security-focused operating systems like secureblue. Our allocator also heavily influenced the design of the next-generation musl malloc implementation which offers substantially better security than musl's previous malloc while still having minimal memory usage and code size.
    1. Fully out-of-line metadata with protection from corruption, ruling out traditional allocator exploitation
    2. Separate memory regions for metadata, large allocations and each slab allocation size class with high entropy random bases and no address space reuse between the different regions
    3. Deterministic detection of any invalid free
    4. Zero-on-free with detection of write-after-free via checking that memory is still zeroed before handing it out again
    5. Delayed reuse of address space and memory allocations through the combination of deterministic and randomized quarantines to mitigate use-after-free vulnerabilities
    6. Fine-grained randomization
    7. Aggressive consistency checks

8. Memory protected guard regions around allocations larger than 16k with randomization of guard region sizes for 128k and above
9. Allocations smaller than 16k have guard regions around each of the slabs containing allocations (for example, 16 byte allocations are in 4096 byte slabs with 4096 byte guard regions before and after)
10. Random canaries with a leading zero are added to these smaller allocations to block C string overflows, absorb small overflows and detect linear overflows or other heap corruption when the canary value is checked (primarily on free)
11. Hardware memory tagging for slab allocations (128k and below) providing probabilistic detection of all use-after-free and inter-object overflows along with deterministic detection of all small/linear overflows and use-after-free until it has been reused once and gone through the quarantines twice

- On ARMv9, Branch Target Identification (BTI) and Pointer Authentication Code (PAC) return address protection are enabled for userspace OS code we build instead of only specific apps
- Signed integer overflow is made well defined in C and C++ for code where automatic overflow checking is disabled
- Hardened kernel
    1. 4-level page tables are enabled on arm64 to provide a much larger address space (48-bit instead of 39-bit) with significantly higher entropy Address Space Layout Randomization (33-bit instead of 24-bit).
    2. Random canaries with a leading zero are added to the kernel heap (slub) to block C string overflows, absorb small overflows and detect linear overflows or other heap corruption when the canary value is checked (on free, copies to/from userspace, etc.).
    3. Memory is wiped (zeroed) as soon as it's released in both the low-level kernel page allocator and higher level kernel heap allocator (slub). This substantially reduces the lifetime of sensitive data in memory, mitigates use-after-free vulnerabilities and makes most uninitialized data usage vulnerabilities harmless. Without our changes, memory that's released retains data indefinitely until the memory is handed out for other uses and gets partially or fully overwritten by new data.
    4. Kernel stack allocations are zeroed to make most uninitialized data usage vulnerabilities harmless.
    5. Assorted attack surface reduction through disabling features or setting up infrastructure to dynamically enable/disable them only as needed (perf, ptrace).
    6. Assorted upstream hardening features are enabled, including many which we played a part in developing and landing upstream as part of our linux-hardened project (which we intend to revive as a more active project again).
    7. Forced kernel module signing with per-build RSA 4096 / SHA256 keys and lockdown mode set to forced confidentiality mode help to enforce a low-level boundary between the kernel and userspace even if mistakes are made in SELinux policy or there's a deep userspace compromise.

8. Additional consistency/integrity checks are enabled for frequently targeted kernel data structures.
9. On ARMv9, Branch Target Identification (BTI) is enabled in addition to Clang type-based Control Flow Integrity (CFI) to cover functions excluded from type-based CFI
10. On ARMv9, ShadowCallStack (SCS) is enabled in addition to Pointer Authentication Code (PAC) return address protection instead of only enabling SCS when PAC is unavailable

- Android Runtime Just-In-Time (JIT) compilation/profiling is fully disabled and replaced with full ahead-of-time (AOT) compilation. The only JIT compilation in the base OS is the v8 JavaScript JIT which is disabled by default for the Vanadium browser with per-site exception support.
- Prevention of dynamic native code execution via either memory or storage for the base OS including nearly all the base OS apps. For the OS itself, only the processes involved in the OS package management system can write data to storage that can be executed and only the media DRM sandbox can do in-memory dynamic native code execution. The Vanadium browser and WebView are excluded in order to support the JS JIT compiler.
- Filesystem access hardening

Improved sandboxing

GrapheneOS improves the app sandbox through hardening SELinux policy and seccomp-bpf policy along with all the hardening to components like kernel implementing the app sandbox and providing a path for the attacker to escape it if they can exploit those components. We primarily focus on the app sandbox, but we also improve the other sandboxes including making direct improvements to the web browser renderer sandbox used for both the default browser and WebView rendering engine provided by the OS and used by a huge number of other apps from dedicated browsers to messaging apps.

Anti-persistence / detection

- Enhanced verified boot with better security properties and reduced attack surface
- GrapheneOS finishes the incomplete implementation of verified boot for out-of-band updates to packages (APKs) in the OS. We enforce this by requiring fs-verity metadata signed with a trusted key for system app updates both at install time and boot time. This provides continuous verification where every read from an out-of-band APK update is verified similarly to every read from a firmware, OS image or APEX update being verified. The signing key and version are enforced to prevent downgrades or other attacks such as replacing a package with a variant of the same one from a different GrapheneOS supported device. We disable the persistent package parsing cache to prevent bypassing the metadata checks through this otherwise highly persistent state, which only has a very small negative impact on boot time from the data not being available from previous boots (typically less than 1 second).
- GrapheneOS closes a loophole where app-based system components built as part of the OS can be downgraded to an older version due to versionCode not being

incremented when system components get updated as part of changes to the OS. We enforce this both at package install time and boot time.
- Enhanced hardware-based attestation with more precise version information
- Hardware-based security verification and monitoring via our Auditor app and attestation service
- Compressed APEX module support is disabled as it's not useful for GrapheneOS, uses extra unnecessary storage space and adds more verified boot attack surface.

More complete patching

GrapheneOS includes fixes for a large number of vulnerabilities not yet fixed in Android.

We're able to quickly and safely ship the latest Linux kernel LTS point releases on devices with GKI (Generic Kernel Image) support including the 6th and 7th generation Pixel phones. At the time of writing on 2023-11-06, GrapheneOS is using the latest Linux 5.10 GKI LTS release (5.10.199) for 6th and 7th generation Pixel phones. The stock Pixel OS is on Linux 5.10.157 from 2022-12-02 with a small number of additional patches backported. This means GrapheneOS provides hundreds of relevant kernel patches including many security patches not yet included in the stock OS. It's possible for us to stay several months ahead due to their approach of moving to new LTS releases only in quarterly releases after a long freeze and testing process.

We often find new vulnerabilities ourselves and report them upstream. We've reported dozens of vulnerabilities for both the generic Android codebase and also for Pixels specifically. We also often find missed patches which were supposed to be included but were missed, especially when there are device specific components with partially shared but separate codebases for different devices.

Our overall approach is to focus on systemic privacy and security improvements but fixing individual vulnerabilities is still very important.

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.

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.

See the usage guide section on sandboxed Google Play for instructions.

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.

For more details, see the usage guide section on Android Auto.

Network permission toggle

GrapheneOS adds a Network permission toggle for disallowing both direct and indirect access to any of the available networks. The device-local network (localhost) is also guarded by this permission, which is important for preventing apps from using it to communicate between profiles. Unlike a firewall-based implementation, the Network permission toggle prevents apps from using the network via APIs provided by the OS or other apps in the same profile as long as they're marked appropriately.

The standard INTERNET permission used as the basis for the Network permission toggle is enhanced with a second layer of enforcement and proper support for granting/revoking it on a per-profile basis.

To avoid breaking compatibility with Android apps, the added permission toggle is enabled by default. However, the OS app installation UI has been extended to show the toggle as part of the installation confirmation page so users can disable it when installing an app.

When the Network permission is disabled, GrapheneOS pretends the network is down. It shows the network as down in various APIs, returns errors showing a network connectivity issue rather than a revoked permission and avoids running scheduled jobs depending on the network. This results in apps handling it as if the network is down rather than crashing or showing errors from trying to use the network and being unable to do it.

Sensors permission toggle

Sensors permission toggle: disallow access to all other sensors not covered by existing Android permissions (Camera, Microphone, Body Sensors, Activity Recognition) including an accelerometer, gyroscope, compass, barometer, thermometer and any other sensors present on a given device. When access is disabled, apps receive zeroed data when they check for sensor values and don't receive events. GrapheneOS creates an easy-to-disable notification when apps try to access sensors blocked by the permission being denied. This makes the feature more usable since users can tell if the app is trying to access this functionality.

To avoid breaking compatibility with Android apps, the added permission is enabled by default. When an app attempts to access sensors and receives zeroed data due to being denied, GrapheneOS creates a notification that can be easily disabled. The Sensors permission can be set to be disabled by default for user installed apps in Settings > Privacy.

Storage Scopes

GrapheneOS provides Storage Scopes as a fully compatible alternative to the standard Android storage permissions. Instead of granting storage permissions, users can enable Storage Scopes to make the app assume that it has all storage permissions that it asked for. On Android, an app that doesn't have any storage permissions is still allowed to create files and directories, and is allowed to access the files that it created. Users can optionally add files and directories as storage scopes to permit the app to access files created by other apps.

For more details, see the usage guide section on storage access.

Contact Scopes

GrapheneOS provides Contact Scopes as an alternative to granting the Contacts permission. By default, it acts as if the contacts list is empty and users can grant different kinds of access to specific contacts or groups of contacts.

For more details, see the usage guide section on Contact Scopes.

Broad carrier support without invasive carrier access

GrapheneOS has much broader carrier support than AOSP and mostly matches the stock OS on Pixels without making the same sacrifices. We convert their APN, carrier configuration, MMS and visual voicemail databases to the formats used by AOSP with our CarrierConfig2 project and scripts. We strip out anti-user configuration requiring provisioning for tethering, forbidding disabling 2G, etc. We don't include the invasive carrier-specific apps and support for Open Mobile Alliance Device Management (OMA DM) so we also strip out configuration depending on those.

See our usage guide section on carrier functionality for more details.

LTE-only mode

LTE-only mode to reduce cellular radio attack surface by disabling enormous amounts of both legacy code (2G, 3G) and bleeding edge code (5G).

Wi-Fi privacy

GrapheneOS supports per-connection MAC randomization and enables it by default. This is a more private approach than the standard persistent per-network random MAC used by modern Android.

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 applies fixes for serious flaws with the Linux kernel IPv6 privacy address implementation which allow using it as an identifier not just for connections to the same network but also across different networks. We don't need to apply these changes for the Pixel 6 and later since this was fixed in the Linux kernel upstream, but hasn't been backported to earlier kernel LTS branches so we still need to take care of it there.

See our usage guide section on Wi-Fi privacy for more general information rather than only our improvements to the standard Wi-Fi privacy approach.

Private screenshots

GrapheneOS disables the inclusion of sensitive metadata in screenshots.

On Android, each screenshot includes an EXIF Software tag with detailed OS build/version information (android.os.Build.DISPLAY). It's the same value shown at Settings > About device > Build number. This leaks the OS, OS version and also usually the device family/model since builds are specific to a family of devices. GrapheneOS completely disables this tag.

On Android, each screenshot also includes EXIF tags with the local date, time and timezone offset. GrapheneOS disables this by default to avoid leaking the time and quasi-location

information through metadata that isn't visible to the user. The date and time are already included in the file name of the screenshot which is fully visible to the user and can be easily modified by them without a third-party tool. GrapheneOS includes a toggle for turning this metadata back on in Settings > Privacy since some users may find it to be useful.

Closed device identifier leaks

GrapheneOS fixes several prominent device identifier leaks bypassing Android's intention of apps not being able to uniquely identify a device. See our FAQ sections on hardware identifiers and non-hardware identifiers for more general information.

Our secure application spawning system primarily exists to significantly improve protection against exploitation. However, it also improves privacy. On a device without our secure application spawning system, the secrets used for probabilistic exploit mitigations such as ASLR are usable as device identifiers persisting until reboot. This is an easy way to identify the device from apps in different profiles. It's a minor bonus of the feature and there are still plenty of side channels to identify devices across apps, but it fixes most of the known direct identifier leaks.

We also eliminate several holes in preventing apps from accessing hardware identifiers including tightening up the restrictions for apps targeting legacy Android platform versions.

PIN scrambling

GrapheneOS adds a toggle for enabling PIN scrambling to raise the difficulty of figuring out the PIN being entered by a user either due to physical proximity or a side channel. PIN scrambling is applied to both the lock screen and SIM PIN/PUK.

Privacy by default

GrapheneOS doesn't include or use Google apps and services by default and avoids including any other apps/services not aligned with our privacy and security focus. Google apps and services can be used on GrapheneOS as regular sandboxed apps without any special access or privileges through our sandboxed Google Play feature, but we don't include those apps by default to give users an explicit choice on whether they want to use those apps and which profiles they want to use it in.

We change the default settings to prefer privacy over small conveniences: personalized keyboard suggestions based on gathering input history are disabled by default, sensitive notifications are hidden on the lockscreen by default and passwords are hidden during entry by default.

Some of our changes for attack surface reduction can also improve privacy by default by not exposing unnecessary radios, etc. by default and avoiding the impact of potential privacy bugs with the hardware.

By default, we also use GrapheneOS servers for the following services instead of Google servers:

- Connectivity checks
- Attestation key provisioning
- GNSS almanac downloads (PSDS) for Broadcom and Qualcomm (XTRA)
- Secure User Plane Location (SUPL)
- Network time
- Vanadium (Chromium) component updates

We provide a toggle to switch back to Google's servers for connectivity checks, attestation key provisioning and GNSS almanac downloads along with adding proper support for disabling network time connections. This combines with other toggles to allow making a GrapheneOS device appear to be an AOSP device. This is only particularly important for connectivity checks since the other connections get routed through a VPN which is needed to blend in on a local network in practice.

In addition to our SUPL privacy improvements, we override the SUPL server to our proxy by default. We also add a toggle for users to switch to the standard SUPL server for their carrier (usually supl.google.com) or disable it entirely.

See our default connections FAQ entry for much more detailed information.

Supports longer passwords

GrapheneOS supports setting longer passwords by default: 128 characters instead of 16 characters. This avoids the need to use a device manager to enable this functionality.

This feature allows users to make use of diceware passwords if they don't want to depend on the security of the secure element which provides very aggressive throttling and offers a high level of security even for a random 6 digit PIN.

Auto reboot

GrapheneOS provides an auto-reboot feature which reboots locked devices after a set period of time to put data at rest. A countdown timer is started each time the device is locked, and the device will reboot if a successful unlock doesn't occur before the timer reaches zero. Unlocking any profile cancels the timer, not just the Owner profile.

The timer is set to 18 hours by default, but can be set to values between 10 minutes and 72 hours, or turned off.

This feature doesn't apply when the device is in "Before First Unlock" state, meaning that it will not lead to the device continuously rebooting, as data is already at rest.

The feature is implemented in the init process, preventing it from being bypassed through system process crashes since an init crash causes a kernel panic which leads to a reboot.

Duress PIN/Password

GrapheneOS provides users with the ability to set a duress PIN/Password that will irreversibly wipe the device (along with any installed eSIMs) once entered anywhere where the device credentials are requested (on the lockscreen, along with any such prompt in the OS).

The wipe does not require a reboot and cannot be interrupted. It can be set up at Settings > Security > Duress Password in the owner profile. Both a duress PIN and password will need to be set to account for different profiles that may have different unlock methods.

Note that if the duress PIN/Password is the same as the actual unlock method, the actual unlock method always takes precedence, and therefore no wipe will occur.

More secure fingerprint unlock

GrapheneOS improves the security of the fingerprint unlock feature by only permitting 5 total attempts rather than implementing a 30 second delay between every 5 failed attempts with a total of 20 attempts. This doesn't just reduce the number of potential attempts but also makes it easy to disable fingerprint unlock by intentionally failing to unlock 5 times with a different finger.

GrapheneOS also adds support for using the fingerprint scanner only for authentication in apps and unlocking hardware keystore keys by toggling off support for unlocking. This feature already existed for the standard Android face unlock feature.

Improved user profiles

Android's user profiles are isolated workspaces with their own instances of apps, app data and profile data (contacts, media store, home directory, etc.). Apps can't see the apps in other user profiles and can only communicate with apps within the same user profile (with mutual consent with the other app). Each user profile has their own encryption keys based on their lock method. They're a great fit for GrapheneOS with a lot of room for improvement.

GrapheneOS provides improvements to user profile functionality and is working on further improvements to make switching between them and monitoring other profiles much more convenient.

More user profiles

GrapheneOS raises the limit on the number of secondary user profiles to 32 (31 + guest) instead of only 4 (3 + guest) to make this feature much more flexible.

End session

GrapheneOS also enables support for logging out of user profiles without needing a device manager controlling the device to use this feature. Logging out makes profiles inactive so none of the apps installed in them can run. It also purges the disk encryption keys from memory and hardware registers, putting the user profile back at rest.

Disabling app installation

GrapheneOS adds a toggle to the user management settings for disabling secondary user app installation. You can install the apps you want to be usable in a secondary user and then disable the ability to install more apps as that user in the Owner profile. Android supports this as a standard device management feature but doesn't make it available to a user who owns their own device.

Improved install available apps

GrapheneOS enables the standard install available apps feature that's still not enabled in AOSP or the stock Pixel OS to allow the Owner user to install packages that are available in other users. This allows installing an app in a secondary user that's already installed in the Owner user without needing to download it again. This helps a lot with using the toggles added for disabling app installation by secondary users.

Notification forwarding

GrapheneOS supports forwarding notifications from users running in the background to the currently active user. Forwarding notifications to other users is disabled by default and can be enabled within each user profile where forwarding to the active profile is wanted. Notifications forwarded from other profiles are displayed by default in a standard local notification channel.

GrapheneOS app repository

GrapheneOS includes our own security, minimalism and usability-focused app repository client for using our first-party app repository. Our app repository is currently used to distribute our own apps and a mirror of Google Play for the sandboxed Google Play feature. In the future, it will be used to distribute first-party GrapheneOS builds of externally developed open source apps with hardening applied.

Vanadium: hardened WebView and default browser

GrapheneOS includes our Vanadium browser as WebView implementation provided by the OS and our default browser. Vanadium is a hardened variant of Chromium providing enhanced privacy and security, similar to how GrapheneOS compares to AOSP. The Vanadium browser currently doesn't add many features but there are a lot of enhancements planned in the long term.

Some of the features added compared to standard mobile Chromium:

- Type-based Control Flow Integrity (CFI)
- Hardware memory tagging (MTE) enabled for the main allocator
- Strict site isolation and sandboxed iframes
- JavaScript JIT disabled by default with per-site toggle via drop-down permission menu
- Native Android autofill implementation to avoid needing sandboxed Google Play for autofill support
- WebGPU disabled for attack surface reduction
- WebRTC IP handling policy toggle to control peer-to-peer WebRTC mode

- Compiler hardening: automatic variable initialization, strong stack protector, well-defined signed overflow
- High performance content filtering engine using EasyList + EasyPrivacy with per-site toggle via drop-down permission menu
- More complete state partitioning without origin trial opt-out
- High entropy client hints are replaced with the frozen user agent values to avoid leaking device/OS info
- Battery API always shows the battery as charging and at 100% capacity
- Trivial subdomain hiding disabled
- Consistent browser behavior across users without usage of feature flags and seed-based trials
- Nearly all remote services disabled by default or removed. Only connects to GrapheneOS servers by default. There are only 2 default services: component updates such as certificate authority and certificate revocation updates and DNS-over-HTTPS connectivity checks when enabled
- Web search and global search intents to replace the need for an OS search app
- Option to always open links from other apps, custom tabs, search intents and share intents in Incognito mode
- Option to reduce or disable sending cross-origin referrer information sharing where a link was opened
- Hybrid post-quantum cryptography enabled by default to match the behavior of Chromium on desktop since the devices we support are more than fast enough

Better default settings, including non-user-facing flags:

- Reduce Accept-Language header by default (only available via chrome://flags)
- Third party cookies disabled by default
- Payment support disabled by default
- Website background sync disabled by default
- Sensors access disabled by default
- Protected media (DRM) disabled by default
- Hyperlink auditing disabled by default
- Do Not Track enabled by default mainly to avoid users differentiating themselves from others by enabling it since it has no real value
- WebRTC IP handling policy set to the most private value by default instead of the least private value (turned into a user-facing option by Vanadium)

Configurable features such as JS JIT disabling and content filtering are currently exclusive to the Vanadium browser. Vanadium WebView is currently excluded from these changes until it has an app setting configuration menu similar to the standard site setting configuration menu.

Extension support isn't planned due to being at odds with site isolation and anti-fingerprinting. We plan to implement more features as part of the browser with a focus on privacy and security improvements which can be active by default rather than opt-in niche features. Improvements

will generally be opt-out on a per-site basis rather than opt-in to provide privacy and security by default and to avoid users making themselves more identifiable by opting into privacy and security features. Default-disabled JS JIT and default-enabled content filtering are early examples of this approach we plan to expand upon.

We plan to add more site settings toggles related to attack surface reduction such as site setting toggles for WebGL, WebGPU, WebRTC and other features which are normally always enabled. This will help with both security and improving the defenses against fingerprinting.

Anti-fingerprinting depends on having a large userbase with the same browser, extensions, content filters and other web-facing configuration. Once Vanadium has more features, it will be made available outside GrapheneOS to expand the userbase. Our approach to attack surface reduction eliminates fingerprinting methods in addition to attack surface for exploits and this will be a key part of how we approach preventing fingerprinting by not having features like WebGL, WebGPU and WebRTC exposed in the first place. Good defaults and avoiding having users changing web-facing configuration is an important part of this. Content filters will remain standard across users and updated together as part of the Vanadium configuration app. We'll address the need for language-focused filters by enabling them based on browser language configuration. Fingerprinting based on hardware differences will become more relevant once Vanadium is available outside of GrapheneOS which will always support a small set of highly secure devices.

State partitioning still needs to be fully completed. The main remaining hurdle is providing full cookie partitioning. Mainstream browsers with this feature rely on heuristics bypassing cookie partitioning which can be easily abused to bypass the feature. We tried deploying full cookie partitioning by default but had to roll it back and will need to consider how to approach this particularly with our goal of having most Vanadium users using nearly the same configuration.

We plan to move to a better content engine with support for more advanced filter rules and cosmetic filtering in the future. Expanding the standard filters will depend on having support for the extensions used by uBlock Origin, AdGuard and other filters.

Most browser data is currently excluded from OS backups, which will likely be changed once GrapheneOS has a better backup service included. Export/import for bookmarks and similar data export/import features are also planned. Sync beyond OS backup service support which will eventually provide per-app backup and restore including across devices and via sync services is not planned.

More information is available in the web browsing section of our usage guide.

Auditor app and attestation service

Our Auditor app and attestation service provide strong hardware-based verification of the authenticity and integrity of the firmware/software on the device. A strong pairing-based approach is used which also verifies the device's identity based on the hardware-backed key

generated for each pairing. Software-based checks are layered on top with trust securely chained from the hardware. For more details, see the About and Tutorial pages.

GrapheneOS Camera

GrapheneOS Camera is a modern camera app with a great user interface and a focus on privacy and security. More details are available in the camera section of our usage guide.

GrapheneOS PDF Viewer

GrapheneOS PDF Viewer is a sandboxed, hardened PDF viewer using HiDPI rendering with features like pinch to zoom, text selection, viewing encrypted PDFs, etc.

Encrypted backups

Encrypted backups via integration of the Seedvault app with support for local backups and any cloud storage provider with a storage provider app.

Seedvault was created by a GrapheneOS community member for inclusion in our operating system. We plan on replacing it with a new implementation since the project has been taken over by another group of people not sharing our goals or approach. For now, this is the best available option, so we're including it to give people encrypted backup support. We've made several security fixes to work around upstream issues with the project.

Location data access indicator

GrapheneOS enables the privacy indicator for location data access in addition to the standard Android camera and microphone indicators. This shows an indicator when an app the user has granted permission to access location requests location data. We also resolve various UX issues with this feature as it currently exists in AOSP to get it into a highly usable state.

Android 13 has the location privacy indicator as a developer option but it doesn't work the same way as it does in GrapheneOS. GrapheneOS shows it for all location data accesses through any APIs. Normally, the stock OS only shows it for GNSS location requests, also known as high power location requests, and doesn't normally show it for network location and other APIs gated by the Location permission / global block toggle.

The indicator works the same way as the Camera and Microphone ones, showing a bright green icon when location access occurs which then gets minimized to a small bright green dot when the quick settings tray isn't currently opened. Android 12 already includes Location with the other standard runtime permissions in the privacy dashboard for viewing the history.

User installed apps can be disabled

GrapheneOS adds support for disabling user installed apps instead of only being able to disable system apps. This allows users to completely prevent one of the apps they've installed from being able to run without being forced to uninstall it and lose their app data. This is much stricter than the standard force stop feature which only prevents an app from starting itself and the app will start running again as soon as another app tries to open an activity or service it provides.

Other features

This is an incomplete list of other GrapheneOS features.

- Per-profile encrypted file name padding increased from 16 bytes to 32 bytes to reduce the information leaked through file name lengths. See the FAQ section on the filesystem-based full disk encryption used by modern Android and GrapheneOS for more information.
- Improved user visibility into persistent firmware security through version and configuration verification with reporting of inconsistencies and debug features being enabled.
- Authenticated encryption for network time updates via a first-party server to prevent attackers from changing the time and enabling attacks based on bypassing certificate / key expiry, etc.
- Proper support for disabling network time updates rather than just not using the results
- Hardened local build / signing infrastructure
- Seamless automatic OS update system that just works and stays out of the way in the background without disrupting device usage, with full support for the standard automatic rollback if the first boot of the updated OS fails
- Require unlocking to access sensitive functionality via quick tiles
- Minimal bundled apps and services. Only essential apps are integrated into the OS. We don't make partnerships with apps and services to bundle them into the OS. An app may be the best choice today but a poor choice in the future, and vice-versa. Our approach will be recommending certain apps during the initial setup, not hard-wiring them into the OS.
- Wireless alerts are completely optional since GrapheneOS adds a toggle for the otherwise mandatory presidential alert type. This is particularly useful in Canada where the government abuses the system and sends every type of alert as a presidential alert to stop users from being able to opt out of weather and amber alerts.
- Removal of TrustCor root certificate authority as a trusted system CA.
- Secure-by-default Android 12 PendingIntent security check (FLAG_IMMUTABLE) instead of crash-by-default improving older app compatibility and security.
- Fixed UART debugging enabled warning on offical release builds.
- Engineering / Prototype ("EVT", "PVT" or "DVT") device warning as these devices typically have relaxed security controls for development, mainly the secure boot state property ro.boot.secure_boot not set to PRODUCTION.
- Enable bootloader, radio, and boot partition version / fingerprint checks.
- Remove code automatically granting the location permission to system browsers.
- Apps that don't have any storage permission aren't allowed to read the list of all user-created directories (this is allowed on Android). The list of files is hidden from such apps on both Android and GrapheneOS.
- Screenshot shutter sound is toggleable using the Tap & click sounds option in Settings > Sound & vibration while still following the standard method of putting the device on vibration/silent mode to turn off the screenshot shutter sound.

- More precise system clock via lowering the system clock time update threshold from 2000ms to 50ms and lowering the system clock drift warning from 2000ms to 250ms. This can be helpful for time-based protocols such as TOTP.
- Call recording functionality within the Dialer app using modern Android storage with recordings stored in Recordings/Call Recordings and no restrictions based on region or special cases like playing a recording tone (users are still responsible for complying with their local laws).
- Change standard Android package installer behavior to preserving packages being disabled after updating them instead of them being re-enabled.
- Enable the "Always-on VPN" and "Block connections without VPN" toggles for VPNs by default.

Services

- Strict privacy and security practices for our infrastructure
- Unnecessary logging is avoided, and logs are automatically purged after 4 days (network services used by the OS) to 10 days
- Services are hosted entirely via our own dedicated servers and virtual machines from OVH (and BuyVM for mirrors) without involving any additional parties for CDNs, SaaS platforms, mirrors or other services
- Our services are built with open technology stacks to avoid being locked into any particular hosting provider or vendor
- Open documentation on our infrastructure including listing out all of our services, guides on making similar setups, published configurations for each of our web services, etc.
- No proprietary services
- Authenticated encryption for all of our services
- Strong cipher configurations for all of our services (SSH, TLS, etc.) with only modern AEAD ciphers providing forward secrecy
- Our web sites do not include any third party content and entirely forbid it via strict Content Security Policy rules
- Our web sites disable referrer headers to maximize privacy
- Our web sites fully enable cross origin isolation and disable embedding in other content
- DNSSEC implemented for all of our domains to provide a root of trust for encryption and authentication for domain/server configuration
- DNS Certification Authority Authorization (CAA) records for all of our domains permitting only Let's Encrypt to issue certificates with fully integrated support for the accounturi and validationmethods pinning our Let's Encrypt accounts as the only ones allowed to issue certificates
- DANE TLSA records for pinning keys for all our TLS services
- Our mail server enforces DNSSEC/DANE to provide authenticated encryption when sending mail including alert messages from the attestation service
- SSHFP across all domains for pinning SSH keys
- Static key pinning for our services in apps like Auditor
- Our web services use robust OCSP stapling with Must-Staple

- No persistent cookies or similar client-side state for anything other than login sessions, which are set up securely using SameSite=Strict, Secure, HttpOnly, and Path=/ flags, prefixed with __Host and have server-side session tracking with the ability to log out of other sessions
- scrypt-based password hashing (likely Argon2 when the available implementations are more mature)