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

- GrapheneOS
- Services
- Project

GrapheneOS

Partial list of GrapheneOS features beyond what AOSP 12 provides:

- Hardened app runtime
- Stronger app sandbox
- 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 Whonix. 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.
    - Fully out-of-line metadata with protection from corruption, ruling out traditional allocator exploitation
    - 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
    - Deterministic detection of any invalid free
    - Zero-on-free with detection of write-after-free via checking that memory is still zeroed before handing it out again
    - Delayed reuse of address space and memory allocations through the combination of deterministic and randomized quarantines to mitigate use-after-free vulnerabilities
    - Fine-grained randomization
    - Aggressive consistency checks
    - Memory protected guard regions around allocations larger than 16k with randomization of guard region sizes for 128k and above
    - 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)
    - 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)
- Hardened compiler toolchain
- Hardened kernel
    - Support for dynamically loaded kernel modules is disabled and the minimal set of modules for the device model are built into the kernel to substantially improve the granularity of Control Flow Integrity (CFI) and reduce attack surface.
    - 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).
    - 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.).
    - 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.
    - Kernel stack allocations are zeroed to make most uninitialized data usage vulnerabilities harmless.
    - Assorted attack surface reduction through disabling features or setting up infrastructure to dynamically enable/disable them only as needed (perf, ptrace).
    - 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).
- Prevention of dynamic native code execution in-memory or via the filesystem for the base OS without going via the package manager, etc.

without going via the package manager, etc.
- Filesystem access hardening
- Enhanced verified boot with better security properties and reduced attack surface

- Enhanced hardware-based attestation with more precise version information
- Eliminates remaining holes for apps to access hardware-based identifiers
- 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, etc.), when the screen is locked (connecting new USB peripherals, camera access) and optionally after a timeout (Bluetooth, Wi-Fi)
- Option to disable native debugging (ptrace) to reduce local attack surface (still enabled by default for compatibility)
- Low-level improvements to the filesystem-based full disk encryption used on modern Android
- Support for logging out of user profiles without needing a device manager: makes them inactive so that they can't continue running code while using another profile and purges the disk encryption keys (which are per-profile) from memory and hardware registers
- Option to enable automatically rebooting the device when no profile has been unlocked for the configured time period to put the device fully at rest again.
- Improved user visibility into persistent firmware security through version and configuration verification with reporting of inconsistencies and debug features being enabled.
- Support longer passwords by default (64 characters) without a device manager
- Stricter implementation of the optional fingerprint unlock feature permitting only 5 attempts rather than 20 before permanent lockout (our recommendation is still keeping sensitive data in user profiles without fingerprint unlock)
- Support for using the fingerprint scanner only for authentication in apps and unlocking hardware keystore keys by toggling off support for unlocking.
- PIN scrambling option
- LTE-only mode to reduce cellular radio attack surface by disabling enormous amounts of legacy code
- Per-connection MAC randomization option (enabled by default) as a more private option than the standard persistent per-network random MAC.
- 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.
- Improved IPv6 privacy addresses to prevent tracking across networks
- Vanadium: hardened WebView and default browser — the WebView is what most other apps use to handle web content, so you benefit from Vanadium in many apps even if you choose another browser
- Hardware-based security verification and monitoring: the Auditor app 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 provides verification of 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 page and tutorial.
- PDF Viewer: sandboxed, hardened PDF viewer using HiDPI rendering with pinch to zoom, text selection, etc.
- Encrypted backups via integration of the Seedvault app with support for local backups and any cloud storage provider with a storage provider app
- Secure application spawning system avoiding sharing address space layout and other secrets across applications
- 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

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.
- 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. To avoid breaking compatibility with Android apps, the added permission is enabled by default.
- 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
- Connectivity checks via a first party server with the option to revert to the standard checks (to blend in) or to fully disable them
- 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
- Minor changes to 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
- 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 and poor choice in the future. Our approach will be recommending certain apps during the initial setup, not hard-wiring them into the OS.
- No Google apps and services. These can be used on GrapheneOS but only if they avoid requiring invasive OS integration. Building privileged support for Google services into the OS isn't something we're going to be doing, even if that's partially open source like microG.
- Compatibility layer for coercing user installed Google Play services into running as sandboxed apps without any special privileges.
- Fixes for multiple serious vulnerabilities not yet fixed upstream due to a flexible release cycle / process prioritizing security.

Services

Service infrastructure features:

- Strict privacy and security practices for our infrastructure
- Unnecessary logging is avoided and logs are automatically purged after 10 days
- Services are hosted entirely via our own dedicated servers and virtual machines from OVH 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 in to 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.

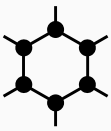
- No proprietary services
- Authenticated encryption for all of our services

- 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 experimental `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 via SameSite=strict cookies 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)

Project

Beyond the technical features of the OS:

- Collaborative, open source project with a very active community and contributors
- Can make your own builds and make desired changes, so you aren't stuck with the decisions made by the upstream project
- Non-profit project avoiding conflicts of interest by keeping commercialization at a distance. Companies support the project rather than the project serving the needs of any particular company
- Strong privacy policies across all our software and services
- Proven track record of the team standing up against attempts to compromise the integrity of the project and placing it above personal gain



GrapheneOS

Twitter GitHub Reddit LinkedIn