Android NAT-T Keepalive Offload Bypasses VPN Lockdown: Device-Class Exposure Across Most Android 12+ Devices

Version 1.0.0

Armin Šupuk

July 29, 2026

1. Abstract

Android's Always-on VPN and "Block connections without VPN" settings create a user-visible expectation that traffic attributable to covered applications will not leave through a non-VPN path. A normal application can violate that boundary through Android's public NAT-T socket-keepalive API, causing clear, fixed-format UDP/4500 packets to reach the physical router outside the VPN path. The runtime evidence has three levels. A controlled access-point capture on a Pixel 8 Pro running Android 16 build `CP1A.260505.005` recorded the packets at the public minimum 10-second interval while Always-on VPN and lockdown were enabled. A Samsung SM-F966B running Android 16 exposed one active Wi-Fi slot through the same public path; VPN Leak Guard selected the physical IPv4 default gateway, observed the active callback, and recorded a continuous router-directed active-slot lease for 24 h 32 min. On a Nothing A059 (`Asteroids`) running Android 16, the same implementation selected the physical gateway and recorded one active Wi-Fi slot. The Nothing result confirms public-path admission and the active callback on a third OEM. No independent packet capture or duration measurement was collected for that device. The Pixel lifecycle matrix covered backgrounding, lock, Doze, battery saver, restricted standby bucket, Binder freezer, and the observed force-stop, uninstall, network-loss, and reboot boundaries.

Source history traces the failure to a collapsed trust model in `startNattKeepaliveWithFd(...)`: a privileged raw-fd API evolved into a public `UdpEncapsulationSocket` path, resource validation was added and reverted, and admission no longer authenticates the fd/resource pair or enforces the original caller UID's current VPN policy before offload. In an F-Droid/IzzyOnDroid study of 4,679 distinct stored Git origins, the scanner detected no Android framework IPsec, IKE, or NAT-T API use; manual audit found 73 Android `VpnService` apps. Runtime confirmation across three OEMs and two confirmed WLAN families, the shared Android 12+ framework path, and firmware coverage across seven WLAN families representing 91.24% of estimated Android-derived shipments establish device-class exposure affecting most Android 12+ devices. The remaining 8.76% is unresolved.

2. Introduction

VPN lockdown governs routing and confinement in addition to encryption. Users and administrators expect covered applications to fail closed when the VPN is unavailable and to withhold their real network identity from destinations outside the tunnel. Prior VPN-leak research has studied routing exceptions, IPv6 and DNS leaks, WebRTC address exposure, VPN client ecosystems, and shared VPN infrastructure failures [1; 2; 3; 4; 5; 6; 7]. Android also delegates some application-triggered packet emission to `system_server`, a NetworkAgent, a hardware abstraction layer, or firmware, beyond the application's ordinary socket send path.

A normal application can cross that boundary through the public Android-managed `IpSecManager.UdpEncapsulationSocket` and ask `ConnectivityManager.createSocketKeepalive(...)` to maintain a NAT-T mapping. The framework routes the request through `startNattKeepaliveWithFd(...)`, accepts the duplicated fd and resource ID without proving current caller-owned IpSec resource identity, and hands a completed NAT-T keepalive packet to the Wi-Fi keepalive offload path without first enforcing the caller UID's effective VPN-lockdown policy. A controlled Pixel 8 Pro capture recorded the resulting UDP/4500 packet on the physical access-point interface. Active-slot observations on two additional OEMs exercised the same public physical-gateway path on Qualcomm hardware [8; 9; 10; 11; 12; 13; 14; 15; 16; 17].

Recent Android Automotive access-control work identified `ConnectivityService.startNattKeepaliveWithFd` in a broad sweep of framework permission anomalies because a related keepalive API enforced `PACKET_KEEPALIVE_OFFLOAD` and the fd-based path did not [18; 19; 20]. That work reported the permission inconsistency. It did not trace the public `UdpEncapsulationSocket` trust split, the reverted IpSec resource validation, or physical Wi-Fi emission under VPN lockdown.

The platform fixes the packet shape, but the caller chooses the destination within the API and routing constraints. Repeated packets disclose the physical network's source address and timing to that destination after the user has enabled blocking without the VPN. This violates lockdown's identity-confinement property without requiring arbitrary payload control.

3. Background: Android VPN Lockdown and NAT-T Keepalive Offload

3.1 Android VPN Lockdown

Android's VPN model can route covered application traffic into a VPN app's TUN interface. Always-on VPN keeps the selected VPN active, and the user-facing "Block connections without VPN" setting is intended to prevent covered traffic from using non-secure networks outside that VPN path [21; 22]. In the normal case, an application write traverses the socket layer, per-UID network policy, fwmark and netd routing state, VPN UID-range routing, firewall/prohibit rules, and eventually the VPN TUN interface when the UID is covered by the VPN.

The normal VPN-protected path is:

```
covered app UID
  -> socket connect/write
  -> fwmark/netd policy and VPN UID range checks
  -> lockdown prohibit/fail-closed decision when needed
  -> VPN app TUN interface
  -> encrypted VPN tunnel over an allowed underlay
```

The relevant security property covers traffic and delegated packet emission attributable to a covered non-owner application. Such emission must stay off non-VPN interfaces unless the platform defines an explicit exemption. VPN-owner underlay traffic, configured split tunnels, documented platform probes, and privileged system functions may follow separate policy.

Android separately records which package is *prepared* to act as the VPN for each user. `VpnService.prepare()` may require user consent; the service itself must be declared with `BIND_VPN_SERVICE`. In `Vpn`, the prepared package is paired with its installed owner UID so uninstall/reinstall and package-only comparisons do not preserve authority. An APK that merely declares a VPN service is not the prepared VPN, and prior consent that has been revoked is not current approval [23; 24]. The installed owner UID and current prepared package provide the authority needed for keepalive admission.

3.2 NAT-T Socket Keepalive Offload

NAT traversal for IPsec commonly uses UDP port 4500. Android exposes a public API path in which an application creates an `IpSecManager.UdpEncapsulationSocket` and asks `ConnectivityManager.createSocketKeepalive(...)` to maintain the NAT mapping [8; 9]. Internally, this path passes a duplicated file descriptor and an IpSec resource ID to `IConnectivityManager.startNattKeepaliveWithFd(...)`, which then reaches `ConnectivityService`, `KeepaliveTracker`, a NetworkAgent, and transport-specific Wi-Fi or cellular keepalive machinery [10; 11; 12].

The NAT-T keepalive offload path is:

```
app UID
  -> IpSecManager.openUdpEncapsulationSocket()
  -> ConnectivityManager.createSocketKeepalive(...)
  -> NattSocketKeepalive.startImpl()
  -> IConnectivityManager.startNattKeepaliveWithFd(...)
  -> ConnectivityService / KeepaliveTracker
```

```
-> NetworkAgent
-> Wi-Fi HAL / chipset firmware
-> UDP/4500 keepalive on physical Wi-Fi
```

The Wi-Fi offload path can emit keepalive frames without waking the application or performing a new socket write for each packet. After framework admission, the final emitter sits below the ordinary app socket path that VPN lockdown normally controls.

The public keepalive path, Wi-Fi HAL offload methods, and compatibility slot requirements are shared platform interfaces below the ordinary app socket path [25; 8; 26].

4. Threat Model and Expected Lockdown Behavior

4.1 Expected Lockdown Behavior

Under Always-on VPN and "Block connections without VPN," a covered normal application must not cause Android-managed NAT-T keepalive packets to leave over the physical network outside the VPN tunnel. If the caller's full Android UID is covered by a non-bypassable or lockdown VPN, the public `UdpEncapsulationSocket` keepalive request should fail, remain unsupported, or be stopped before Wi-Fi or cellular offload emits UDP/4500 traffic on the physical underlay.

The security goal applies to NAT-T emission attributable to a covered normal app. Android's documented policies for VPN-owner underlay traffic, configured split tunnels, and privileged platform functions remain separate. Acceptance of an unauthenticated public fd/resource pair cannot confer physical-underlay offload authority.

4.2 Attacker

The attacker controls a normal Android application installed on the victim device and controls or observes a UDP/4500 endpoint on the Internet. The validated public-API path does not require root, ADB, hidden API access, JNI, raw Binder construction, a dangerous runtime permission prompt, or the privileged `PACKET_KEEPALIVE_OFFLOAD` permission. The local PoC variants declared ordinary networking capabilities such as `INTERNET` and `ACCESS_NETWORK_STATE`.

4.3 Victim Configuration

The victim device has Always-on VPN enabled and "Block connections without VPN" enabled for the attacker's UID. Runtime confirmation used Android 16 configurations across multiple OEMs. The Pixel 8 Pro run used a researcher-controlled Wi-Fi network and an external physical-interface capture; the Qualcomm-based devices supplied active physical-gateway slot observations [21; 22; 27; 15; 16; 17].

ADB and root were used for research instrumentation and packet collection. Neither is an exploitation precondition for the public API path.

Dimension	Public claim
App privilege	Normal third-party app.
Runtime dangerous permission	None required for the core public API behavior.
Common declared permissions	`ACCESS_NETWORK_STATE` and `INTERNET`.
Privileged keepalive permission	No `PACKET_KEEPALIVE_OFFLOAD` for the public API path.
Root / ADB	Not required for exploitation; used only for lab instrumentation.
User interaction	Initial app launch is enough for local PoC variants.
Network condition	Wi-Fi with NAT-T keepalive offload and an available unprivileged slot.

Dimension	Public claim
VPN condition	Always-on VPN plus “Block connections without VPN” in the measured run.
Data leaked	Real non-VPN source IP and timing to an attacker-chosen UDP/4500 endpoint.
Cadence	Public minimum interval on the controlled Pixel configuration.

5. Vulnerability: NAT-T Keepalive Lockdown Bypass

5.1 Public API Sequence

The practical attack uses the documented public path:

```
IpSecManager.UdpEncapsulationSocket socket =
        ipSecManager.openUdpEncapsulationSocket();

SocketKeepalive keepalive = connectivityManager.createSocketKeepalive(
        network,
        socket,
        sourceAddress,
        destinationAddress,
        executor,
        callback);

keepalive.start(10);
```

The public API converges on the fd-based Binder method also used by hidden raw-fd paths. The controlled on-wire Wi-Fi result used this documented sequence; raw Binder was unnecessary [8; 9; 10].

5.2 Packet Path and Missing Decision

`NattSocketKeepalive.startImpl()` calls `IConnectivityManager.startNattKeepaliveWithFd(...)`; `ConnectivityService` forwards the request through `KeepaliveTracker` to the NetworkAgent and Wi-Fi backend. No admission decision checks the original caller’s effective VPN policy before hardware offload. Once handed to the Wi-Fi backend, repeated emission is outside the normal app socket writes intercepted by per-UID VPN routing and lockdown firewall rules [11; 12; 26].

5.3 Scope of Packet Control

The attacker controls the destination address within the API and routing constraints and observes the real source address exposed by the physical network. The platform fixes the NAT-T payload. The primitive leaks the real IP address and timing/cadence; it does not carry arbitrary application content.

5.4 Raw Binder as Supporting Evidence

Raw Binder diagnostics show that the server does not authenticate the supplied file descriptor or claimed IpSec resource. The reviewed validation path reports that `isNattKeepaliveSocketValid(fd, resourceId)` accepts every non-null fd and does not meaningfully consult `resourceId`; bogus values such as `0`, `-2`, `Integer.MAX_VALUE`, and `Integer.MIN_VALUE` are not ownership checks. These diagnostics support the resource-authenticity finding. The VPN-lockdown result uses the documented public API and is independent of a stable Binder transaction number or hidden API construction. The source-line anchors are `KeepaliveTracker.makeNattKeepaliveInfo( ... resourceId ...)` and `isNattKeepaliveSocketValid(...)` on the reviewed AOSP Connectivity branch [12].

Supplemental fuzzing found broad IPv4 destination acceptance after admission and route-driven IPv6 behavior. Neither result is a prerequisite for the VPN-lockdown bypass.

6. Root Cause: Collapsed Trust Model and Abandoned Resource Validation

The fd-based Binder method combines two trust models. The original raw-fd NAT-T keepalive API was privileged. During API review, public `UdpEncapsulationSocket` keepalives were routed through the same method, and the unconditional permission check was removed to support public IpSec/IKE applications. Secure admission then required public callers to prove ownership of a live `IpSecService` encap-socket resource while raw-fd callers without such a resource remained privileged.

Two checks are absent. The server accepts a public NAT-T keepalive request without proving that the duplicated fd corresponds to a live caller-owned IpSec encap-socket resource. It also starts the offload path without checking whether VPN/lockdown policy for the caller UID blocks physical-underlay emission. Once admitted, the Wi-Fi offload path emits below the ordinary app socket path where lockdown would normally constrain the caller.

Source history shows that resource validation and lifetime locking were briefly added. The implementation validated caller UID ownership, pinned the encap socket for the keepalive lifetime, rejected duplicate active use, and released the resource on final stop. It was reverted because of service dependency and deadlock concerns. Per-UID/per-network quotas replaced it. Quotas limit resource exhaustion; they do not authenticate the fd/resource pair, preserve an IpSec lifetime lease, or enforce VPN policy.

The commit IDs were rechecked against the AOSP `packages/modules/Connectivity` Gitiles repository on 2026-05-30 [28].

Date	Change	Security relevance
2015	Packet keepalive offload existed as privileged platform behavior.	Baseline: not a general untrusted-app emission primitive.
2019-01	fd-based NAT-T keepalive support was introduced for privileged raw-fd callers.	The first fd-based version still enforced keepalive permission.
2019-03	Public `UdpEncapsulationSocket` keepalives were moved onto the fd-based method.	The shared method could no longer use one unconditional privileged check without breaking the public API.
2019-04	IpSec resource validation and lifetime locking were added.	The change enforced caller ownership, resource validation, duplicate-use rejection, and lifetime pinning.
2019-05	The IpSec validation stack was reverted for service dependency/deadlock concerns.	fd/resource ownership and lifetime checks were removed.
2019-05	Per-UID/per-network quota limits replaced validation.	Quotas address resource exhaustion, not confidentiality, fd/resource authenticity, or VPN routing policy.
2023+	Automatic-on/off and `underpinnedNetwork` fields were added to the same Binder method.	More caller-supplied state reaches the same admission point and needs the same validation model.

The revert removed the following security checks:

Removed check	Former control	Current consequence
Raw fd and public `UdpEncapsulationSocket` callers have distinct trust models.	Early privileged raw-fd method plus later IpSec validation stack.	Both reach `startNattKeepaliveWithFd(...)`; one unconditional permission gate is insufficient and validation is absent.
Public caller owns the supplied encap-socket resource.	`IpSecService.lockEncapSocketForNattKeepalive(...)` looked up caller-owned records.	Current validation does not use `resourceId` as an ownership proof.
Invalid or other-UID resource IDs fail before offload.	Reverted IpSecService tests covered invalid-resource and invalid-UID cases.	Bogus IDs are authenticity failures, not meaningful authorization checks.
The encap socket remains alive for the keepalive lifetime.	`NattKeepaliveRecord` pinned the `EncapSocketRecord`.	The current path holds a duplicated fd but not an IpSec resource lease.
One encap resource cannot back two active NAT-T keepalives.	Reverted KeepaliveTracker resource locking rejected duplicates.	Slot quotas may hide duplicates on a device but do not authenticate duplicate resource use.

The public API duplicates a `UdpEncapsulationSocket` fd, passes `socket.getResourceId()` through `NattSocketKeepalive`, and calls `IConnectivityManager.startNattKeepaliveWithFd(...)` with the fd, resource ID, automatic-on/off state, and `underpinnedNetwork`. `ConnectivityService` forwards those fields to `KeepaliveTracker`; the existing socket validation accepts non-null fd state without proving that the fd matches a live caller-owned IpSec resource and without enforcing effective VPN policy before the NetworkAgent or transport backend starts hardware offload [9; 10; 11; 12].

7. Evaluation

The runtime evidence has three distinct levels. On the Pixel 8 Pro Wi-Fi configuration, Android accepted a public NAT-T keepalive request from a normal application and emitted repeated UDP/4500 packets on the physical Wi-Fi interface while VPN lockdown remained enabled. The authoritative observation point for that controlled matrix is an OpenWrt AP/router tcpdump on a separate device, because packets observed there have already crossed Android's VPN policy boundary. A Samsung SM-F966B independently confirmed the same public path on Qualcomm WLAN hardware by maintaining one active, router-directed physical Wi-Fi slot through a lease exceeding one day. A Nothing A059 on Qualcomm hardware provided third-OEM public-path admission and active-callback confirmation through one physical-gateway Wi-Fi slot, without an external packet capture or duration measurement.

7.1 Controlled Pixel Packet Capture

The public NAT-T keepalive path produced on-wire UDP/4500 traffic on physical Wi-Fi while lockdown was enabled. The separate OpenWrt AP/router recorded a one-byte UDP payload every 10 seconds:

```
2026-05-28 18:36:31.709917 phy1-ap0 P   IP 192.168.1.182.38904 > 1.2.3.4.4500: UDP, length 1
2026-05-28 18:36:41.710042 phy1-ap0 P   IP 192.168.1.182.38904 > 1.2.3.4.4500: UDP, length 1
```

The public path used `IpSecManager.openUdpEncapsulationSocket()` and `ConnectivityManager.createSocketKeepalive(...)`; no root, ADB, hidden API, raw Binder construction, JNI, dangerous runtime permission, or `PACKET_KEEPALIVE_OFFLOAD` permission was required for the app-side primitive.

The caller chooses the destination within API and routing constraints. The receiver observes the device's real non-VPN source address and cadence; the payload remains the platform's fixed NAT-T keepalive format.

7.2 Independent Samsung Runtime Confirmation

VPN Leak Guard produced an independent cross-OEM runtime result on a Samsung SM-F966B running Android 16 build `BP4A.251205.006.F966BXXUABZF1` on Qualcomm hardware. Its keepalive implementation excludes VPN logical networks, selects the physical network's validated IPv4 default gateway, opens `IpSecManager.UdpEncapsulationSocket`, and calls `ConnectivityManager.createSocketKeepalive(...)` with that physical network and gateway as the UDP/4500 destination. The Samsung run received the active callback and exposed one active physical-gateway Wi-Fi slot [13; 15].

The observation snapshot records both the highest and latest Wi-Fi slot counts as one. A sanitized screenshot records the same slot still active at a lease uptime of 24 h 32 min [17]. This is independent runtime confirmation of the vulnerable public path and physical-gateway destination on a Samsung/Qualcomm stack. The Pixel matrix remains the only controlled external packet capture; Samsung supplies the active-slot and lease measurements.

7.3 Nothing Active-Slot Confirmation

VPN Leak Guard also recorded a Nothing A059, device and product `Asteroids`, board `volcano`, on Qualcomm (`qcom`) hardware. It ran Android 16, SDK 36, security patch `2026-06-01`, build ID `BQ2A.250721.001-BP2A.250605.031.A3`. The protector excludes VPN logical networks, selects the validated physical IPv4 default gateway, and marks a slot active only when `SocketKeepalive.Callback.onStarted()` runs. The observation report factory omits zero-slot maxima. The read-only snapshot records both the highest and latest active Wi-Fi slot counts as one [13; 14; 16].

The row confirms public physical-gateway admission and the active callback on a third OEM. Its evidence is limited to the active-slot result; no router capture, packet cadence, lease duration, lifecycle, persistence, or reboot result was collected.

7.4 Controls and Baselines

The ordinary UDP lockdown control recorded app-side UDP sends while the router capture recorded no matching UDP/4500 or UDP/12345 packets. Ordinary lockdown-covered UDP was therefore confined or absent from the physical capture while the NAT-T keepalive offload appeared at the AP boundary.

Three policy baselines bound the interpretation. With VPN and lockdown disabled, ordinary UDP and keepalive traffic were visible on the router. With VPN enabled and lockdown disabled, keepalive traffic was visible on the router while ordinary UDP was observed through the VPN interface. With both VPN and lockdown enabled, keepalive traffic was still visible on the router while the ordinary UDP lockdown control remained absent from the physical capture.

7.5 Lifecycle Behavior

Once armed on the Pixel, the keepalive remained active through backgrounding, screen lock, forced idle, battery saver, restricted standby bucket, Binder freezer/process observation, and GrapheneOS relock/back-to-BFU-without-reboot while the device stayed powered. On the Samsung, the single active Wi-Fi slot remained continuously leased for a measured period exceeding one day and was still active when the sanitized evidence screenshot was taken.

The observed stop boundaries were manual force stop, uninstall, network loss, and reboot. Packets were present before the force-stop, uninstall, and reboot actions; Wi-Fi loss stopped the active keepalive with a network-loss error, and a manual re-arm restored it after Wi-Fi returned.

7.6 Slot Counts and Lease

The tested Pixel 8 Pro Wi-Fi configuration exposed one unprivileged keepalive slot to the app UID after privileged reservations. One slot was accepted, later slot attempts failed with `ERROR_INSUFFICIENT_RESOURCES (-32)`. The Samsung SM-F966B independently exposed one active Wi-Fi slot and held it through the measured lease. The Nothing A059 exposed one active Wi-Fi slot; no duration was measured for that row [16].

8. Impact

The direct attacker value is repeated real-network identity disclosure. A destination controlled by the attacker can learn the source IP address as seen from the physical Wi-Fi network, the fact that the device remains online, and packet timing while the keepalive remains armed. Depending on the network, the source IP can imply ISP, organization, travel state, or correlation between a device expected to be behind a VPN and a non-VPN access network.

The value of the leak comes from the user-visible lockdown promise: covered applications are expected to fail closed rather than reveal non-VPN network identity to attacker-chosen destinations. Even without payload exfiltration, a periodic signal can support presence checks, IP correlation, and timing correlation against other observations. The measured Samsung lease exceeded one day.

8.1 Device-Class Exposure

The reviewed IEEE and Wi-Fi Alliance material contains no generic WLAN keepalive-offload mandate. Public implementation history instead points to low-power NIC/driver contracts and vendor FullMAC firmware interfaces. By 2009, Windows 7's NDIS 6.20 model supported low-power ARP, IPv6 Neighbor Solicitation, and 802.11 RSN/GTK offloads; in 2011, IEEE 802.11v standardized adjacent Wireless Network Management keep-alive and proxy mechanisms; by 2014, public Android WLAN source evidence shows Qualcomm firmware command surfaces for STA keepalive and IPsec NAT keepalive; and by Android 6 / Marshmallow in 2015, AOSP included hidden NAT-T keepalive framework support. Android compatibility requirements for app-visible Wi-Fi keepalive offload appeared later, in the Android 10 CDD in 2019 [29; 30; 31; 32; 33].

App-visible slots are part of the Android compatibility model. Android 10's 2019 CDD requires devices that expose Wi-Fi keepalive offload to support the `SocketKeepalive` API and at least three concurrent Wi-Fi keepalive slots. Slot/resource checks found no manufacturer overlay that deliberately zeroed the relevant defaults.

Runtime results cross OEM boundaries within the two confirmed WLAN families. The Pixel 8 Pro/Broadcom configuration emitted packets in the controlled AP capture. The Samsung SM-F966B/Qualcomm configuration maintained an active router-directed Wi-Fi slot through the measured lease. Another Qualcomm-based OEM admitted the public physical-gateway path and reached the active callback for one Wi-Fi slot [27; 34; 35; 13; 14; 15; 16; 17].

The vulnerable admission path is shared Android 12+ framework behavior, making the relevant platform window start with Android 12's 2021 release. The firmware/source census finds keepalive or offloaded-packet support surfaces across all seven tracked Android WLAN stack families: Qualcomm QCA/CLD3/FastConnect, Qualcomm WLAN/QDSP6 WCNSS, Broadcom/Cypress bcmdhd/DHD, MediaTek CONSYS/Connac, Unisoc/Spreadtrum SPRDWL, Samsung S.LSI/Exynos Wi-Fi, and Huawei/HiSilicon Hi11xx. The first-observed public support evidence across those families spans 2014 through 2020 [33; 36; 37; 38].

Runtime confirmation across three OEMs and two confirmed WLAN families, the shared Android 12+ implementation, slot defaults, and the seven-family firmware census establish device-class exposure affecting most Android 12+ devices. The mapped WLAN families represent 91.24% of estimated Android/AOSP-derived shipments from 2021Q4 through 2026Q1. The remaining 8.76% is unresolved mixed/long-tail SKU coverage [39; 36].

9. Application Ecosystem Study

Public API availability does not establish compatibility demand. A static F-Droid/IzzyOnDroid study measured use of Android's framework IPsec, IKE, and NAT-T machinery. Across the scanned origins, the scanner found zero framework API uses and zero method-specific raw Binder invocations of the corresponding transactions [40; 41].

9.1 Corpus and Method

The collection run downloaded the signed F-Droid and IzzyOnDroid v1 indexes on 2026-07-05, extracted and normalized declared source URLs, and cloned available repositories. This produced 4,888 package-keyed per-checkout reports. Those reports contain 4,679 distinct stored `gitOrigin` strings; the table below de-duplicates package-keyed observations by exact stored origin and does not merge differently spelled URLs that might identify the same upstream project.

The analysis was static and lexical: it scanned source and manifest files for framework API references, method-specific raw Binder transactions, and VPN comparison candidates, while excluding generated build directories, dependency caches, Git metadata, and files above the scanner's size limit. The VPN comparison candidates were then manually audited for intentional user-facing Android `VpnService`/TUN-style behavior. Catalog entries, clone targets, or source checkouts that were unavailable to the scanner remain outside the distinct-origin denominator.

The scan result is:

Observed concept	Distinct origins	Share
Android framework IPsec, IKE, or NAT-T API use	0	0.00%
Method-specific raw Binder invocation of IPsec/IKE/NAT-T transactions	0	0.00%
Manually audited Android `VpnService` apps	73	1.56%

The manually audited `VpnService` set confirms that the scanned corpus contains ordinary Android VPN applications. Those apps use the standard `VpnService` path; none supplied a framework IPsec/IKE/NAT-T match.

9.2 Interpretation

Android exposes transform construction, SPI allocation, UDP encapsulation, framework IKE negotiation, migration, state queries, and hardware NAT-T offload to ordinary applications. The sample found no use of that framework machinery. The proposed repair targets NAT-T keepalive admission around fd/resource ownership and effective VPN-lockdown policy; ordinary Java/NDK networking and the `VpnService` path remain outside its scope.

F-Droid and IzzyOnDroid exclude much proprietary enterprise VPN software, OEM clients, carrier software, and sideloaded closed-source applications. The zero therefore describes this open-source sample and cannot establish universal absence.

9.3 Google Play Cross-Check

A Google Play and web census dated 2026-07-29 found 29 candidates: 26 were currently listed and 3 had been removed. APKs were acquired for 9 of the 26 current listings; 17 remained listing-only. The acquired set comprised 8 proprietary APKs and 1 open-source APK. Two proprietary APKs contained all 47 verified platform-SDK call sites: 37 in FortiClient VPN and 10 in SmartVPN. The other 7 acquired APKs contained none [42].

FortiClient VPN had 3,995,326 cumulative Google Play installs and SmartVPN had 139,322, totaling 4,134,648. Google Play exposes these exact cumulative-install fields behind rounded public download badges [42].

"Stale" means still listed but not updated for more than three years on 2026-07-29. Six candidates were removed or stale [42]:

App	Store state	Install/download count
VpnCilla	Removed 2025-03-27	~48,000
NCP VPN Client	Removed 2019-07-16	~22,000
VPN Taiwan – Secure Taiwan IP	Removed 2026-06-17	16,241
Secure Tactical VPN Client	Last updated 2023-06-02	18
VPNGN	Last updated 2023-07-04	21,445

App	Store state	Install/download count
Melba VPN	Last updated 2023-05-31	1,317

The DEX verifier counts an `invoke-*` instruction targeting a platform method, not an incidental string, class descriptor, method signature, or help-text reference.

10. Mitigation and Regression Tests

A repair for any retained normal-app path should preserve authorized NAT-T keepalives while making physical-underlay offload fail closed for covered normal apps. The fix point is admission to `startNattKeepaliveWithFd( ...)` and the associated keepalive lifetime: the platform must authenticate the fd/resource pair, decide whether the caller may emit on the selected physical network, and stop the record if that authorization becomes stale.

10.1 NAT-T Repair Responsibilities

- Raw-fd callers without a valid public IpSec resource remain gated by `PACKET_KEEPALIVE_OFFLOAD` and receive basic fd-shape validation.
- Public `UdpEncapsulationSocket` callers prove ownership of a live `IpSecService` encap-socket resource, prove that the duplicated fd matches the stored resource, and cannot reuse one resource for concurrent active NAT-T records.
- `ConnectivityService` captures the calling full UID before identity clearing, checks the target network and any `underpinnedNetwork` relationship, and rejects physical-underlay offload when the effective VPN/lockdown policy for that UID would block direct emission.
- `KeepaliveTracker` and the transport backend start Wi-Fi or cellular offload only after admission succeeds, and release the resource exactly once on denial, construction failure, binder death, packet-replacement failure, client stop, or final system stop.
- VPN, lockdown, owner, underlying-network, and selected-network changes revoke or revalidate active and paused NAT-T records before they can resume emission.

10.2 Regression Tests

Regression tests for this bug should prove that unauthorized keepalives fail before slot allocation, packet-filter installation, transport start, callback success, or IpSec resource mutation. The minimum negative cases are:

- a covered normal app requests a public NAT-T keepalive while lockdown is already enabled;
- a record admitted before lockdown is stopped when lockdown begins for the caller's UID range;
- running and paused records are stopped when the relevant VPN is removed, replaced by a different owner, changes bypassability, or loses its underlying network;
- stale, closed, mismatched, other-UID, and duplicate-active IpSec resources are rejected before offload;
- raw-fd use with an invalid or absent resource ID requires the privileged keepalive permission;
- denial and final-stop paths close incoming `ParcelFileDescriptor`s and release any acquired IpSec lease exactly once.

Positive coverage should show that a legitimate caller with an owned live encap-socket resource can still use NAT-T keepalive when its effective VPN policy permits the selected physical emission [11; 12].

11. Disclosure, Ethics, and Artifacts

The finding was reported to the Android Vulnerability Reward Program on 2026-05-15. Google triaged the report the same day and requested coordinated disclosure while Android Security assessed the issue. The reporter stated that the finding had not been posted publicly or shared with third parties, submitted additional validation material on 2026-05-17, and asked for disclosure guidance after Google marked the report as a duplicate of canonical issue 386376240 on 2026-05-19.

The reporter notified Google of planned public disclosure through the VRP report on 2026-06-12. The researcher-visible VRP API record then shows redacted Google update markers from two accounts, dated 2026-06-12 and

2026-06-15; neither exposes a comment body. The record contains no objection, delay request, CVE assignment, fix-status update, severity decision, reward decision, or publication clearance. Public disclosure began on 2026-07-29.

The experiments used researcher-controlled devices, VPN configurations, packet captures, and endpoints. No third-party user traffic was collected. Released artifacts are limited to sanitized evidence and reproduction material; private VRP content, local identifiers, weaponized raw traces, and patch diffs are excluded.

Competing interest: the author sells VPN Leak Guard, the commercial Android app that produced the active-slot observations reported here.

12. Limitations

Runtime testing spans three device models and is not an exhaustive per-model inventory. The Pixel 8 Pro/Broadcom result includes a controlled packet-capture matrix. The Samsung SM-F966B/Qualcomm result includes an active physical-gateway slot and measured lease. The Nothing A059/Qualcomm result confirms public-path admission and the active callback; external packet capture and duration remain unmeasured [15; 16].

Reboot persistence is not established. The Pixel keepalive stopped at the observed reboot boundary, the Samsung lease was measured while the device remained powered, and Nothing lifecycle and reboot behavior were not measured.

Cellular packet emission was not measured. The framework admission path is transport-agnostic in source analysis, and Android 16 compatibility material says cellular keepalive offload can be exposed to third-party apps with at least one cellular slot. The tested Pixel default configuration returned insufficient resources before a cellular modem path emitted traffic, and effective normal-app execution depends on manufacturer slot overlays, hardware/HAL availability, privileged slot reservations, per-UID unprivileged limits, and transport backend behavior [25; 8; 43; 12].

APKs were acquired for 9 of 26 current Google Play listings. Store counts are cumulative across versions and VPN modes and do not identify active users or platform-path use. DEX results apply to the acquired APK versions.

No patched Android build, patched-device packet capture, or complete device-level `atest` run was performed. The repair remains source-level guidance and requires implementation-level regression testing.

13. Related Work

VPN-leak comparisons depend on attacker position, trigger, endpoint control, packet shape, cadence, enforcement layer, measured scope, and the user-visible policy being bypassed. The NAT-T case uses a normal installed app to arm repeated fixed UDP/4500 packets to an attacker-chosen Internet endpoint while Android VPN lockdown is enabled.

TunnelCrack and TunnelVision-style work shows that routing exceptions and hostile local-network conditions can defeat VPN expectations on affected platforms [1; 44; 45; 46]. Those attacks use a hostile local network; the NAT-T case uses an installed app. Both test whether packets cross the expected VPN boundary.

IPv6, DNS, WebRTC, and VPN ecosystem studies provide the broader privacy and measurement context. Perta et al., Al-Fannah, Cho and Heidemann, Khan et al., VPNalyzer/VPNInspector, Wu et al., and Yang et al. show why source-IP exposure, resolver behavior, shared VPN state, and careful measurement boundaries matter [2; 3; 6; 4; 5; 47; 48; 7].

These studies measure VPN behavior, infrastructure, privacy, or client properties. The ecosystem study measures compatibility demand for Android's framework IPsec/IKE/NAT-T machinery. F-Droid and IzzyOnDroid make source-level inspection possible [40; 41], but exclude much proprietary enterprise and business VPN software. The resulting zero is therefore evidence for a compatibility context around NAT-T keepalive admission and cannot estimate universal Android-market prevalence.

AutoAcRaptor is the closest prior signal on the same Android framework entry point. It was a broad AAOS access-control study that flagged `ConnectivityService.startNattKeepaliveWithFd` as a verified missing-permission anomaly because a related keepalive API required the signature-level `PACKET_KEEPALIVE_OFFLOAD` permission while the fd-based path did not [18; 19; 20]. That work identified the suspicious entry point but did not validate

the Android phone VPN-lockdown bypass. The NAT-T analysis connects the entry point to public `UdpEncapsulationSocket` keepalives, Wi-Fi offload, and on-wire packet emission outside phone VPN lockdown.

Android QUIC close-payload delegated UDP send is the closest Android delegated-send peer. It shows application-triggered UDP emission outside the ordinary app VPN send path [49; 50; 51]. QUIC close-payload is a one-shot or event-driven software send; NAT-T keepalive is repeated, fixed-format UDP/4500 Wi-Fi offload traffic.

Mullvad's Android connectivity-check and DNS-leak reports, GrapheneOS VPN leak-blocking work, and local-link/multicast discussions show that Android VPN enforcement has long depended on platform exceptions as well as VPN-app behavior [52; 53; 54; 55]. They differ from the NAT-T case in endpoint control, trigger, cadence, and packet shape. They also show that delegated or exempt traffic must be checked against the user's lockdown expectation.

14. Discussion

14.1 Review Failure and Fail-Closed Release Discipline

The public and privileged NAT-T paths reached the same Binder method with different trust requirements and no complete admission boundary. Stronger ownership, fd-identity, lifetime, duplicate-use, and invalid-resource checks were added in 2019, then reverted for technically legitimate service-dependency and deadlock concerns. Per-UID and per-network quotas replaced them, but quotas limit resource consumption; they do not provide equivalent fd/resource authenticity, lifetime, or VPN-policy gates [28; 12].

The remaining validation TODOs cover invalid, closed, stale, mismatched, other-UID, and duplicate resources; raw-fd privilege; lease cleanup; and authorization changes while a keepalive is active or paused. A technically motivated reversion does not itself constitute the failure. Releasing the normal-app path without equivalent controls, while those validation obligations remained open, is a review and release-governance failure. That characterization concerns the process and resulting control gap, not any individual contributor.

Unprivileged availability should have remained at zero slots until an acceptable public/private API split and the complete security checks were ready. Resource quotas cannot serve as release authorization for a physical-underlay emission primitive.

14.2 Deprecate Normal-App IPsec Access

Android should deprecate the public app-facing IPsec, IKE, and NAT-T surface and make the framework functionality system-privileged. Authenticated carrier, IWLAN, VCN, platform VPN, and other platform-internal consumers should remain; their current roles are not replaced merely by removing normal-app access [56; 57; 58].

Normal apps should receive zero exposed slots by default. If legacy access must remain, it should sit behind a default-off, reboot-required compatibility switch with a release posture analogous to radio-generation controls. Ordinary VPN apps can continue to run their own userspace protocol implementations through `VpnService` [23].

The open-source scan found 0 platform consumers across 4,679 origins; the Play study found 2 among 9 acquired APKs. FortiClient VPN and SmartVPN have 4,134,648 cumulative Google Play installs between them, while Google reports more than 3 billion active Android devices. Both apps also support VPN modes that do not use the platform path. I estimate that at most 0.01% of Android users—about one in 10,000—depend on these platform APIs. That constituency is too small to justify leaving normal-app access enabled by default. [40; 41; 42; 59]

14.3 Router-Terminated VPN Guidance

For threat models that cannot tolerate a phone-side VPN escape, a VPN-enforcing external router is the conservative community consensus among identifiable privacy and security practitioners. Mullvad reported recurring Android bypass classes in 2022 for connectivity checks, in 2024 for DNS, and in 2026 for application-triggered QUIC traffic. IVPN independently reproduced the 2026 QUIC path, and GrapheneOS community guidance recommends an external router that tunnels the phone's upstream traffic [52; 53; 60; 61; 62].

These reports cover different mechanisms and do not imply that all Android traffic always bypasses a VPN. Their recurrence shows that Android's current architecture has repeatedly exposed new phone-side bypass paths and cannot responsibly promise that no further class will appear.

The recommendation is conditional. The phone must use the router as its exclusive Internet path, with cellular and alternate networks disabled or separately blocked, and the router must fail closed if its tunnel fails. This reduces dependence on Android's VPN enforcement; it is not an unconditional guarantee against router defects, local-network exposure, or traffic over other radios.

15. Conclusion

The affected class is Android 12+ devices that expose app-visible Wi-Fi NAT-T keepalive offload with usable unprivileged slots. On such devices, a covered normal app can reach physical-underlay offload without effective VPN-lockdown admission. The available runtime, framework, slot, firmware, and shipment evidence supports exposure across most Android 12+ devices [25; 39].

The repair must separate privileged raw-fd requests from public `UdpEncapsulationSocket` requests. Raw-fd callers require `PACKET_KEEPALIVE_OFFLOAD`; public callers require caller-owned resource validation, fd identity checks, lifetime pinning, and duplicate-use rejection. Both paths require effective VPN-policy authorization before NetworkAgent or HAL admission and revalidation when relevant network or VPN state changes. Until those checks are complete, unprivileged NAT-T offload should fail closed [11; 12].

16. Appendix A. Evidence

Displayed pcap hashes use 12-hex SHA-256 prefixes.

A.1 Environment and Boundary Proof

- Primary measured phone: Pixel 8 Pro (`husky`), Android 16 build `CP1A.260505.005`, security patch `2026-05-05`.
- Independent measured phone: Samsung SM-F966B (`q7q`) on Qualcomm (`qcom`) hardware, Android 16 build `BP4A.251205.006.F966BXXUABZF1`, security patch `2026-06-05` [15].
- Additional active-slot phone: Nothing A059, device and product `Asteroids`, board `volcano`, on Qualcomm (`qcom`) hardware, Android 16 / SDK 36, security patch `2026-06-01`, build ID `BQ2A.250721.001-BP2A.250605.031.A3` [16].
- Later-version provenance: the same non-cumulative snapshot records a Pixel 8 Pro on Android 17 with one active Wi-Fi slot. This row establishes slot availability only; it does not replace or relabel the controlled Android 16 Pixel packet capture [16].
- VPN configuration: Mullvad package `net.mullvad.mullvadvpn`, version `2026.5`. Always-on VPN and "Block connections without VPN" are observed in VPN-management snapshots for the measured rows.
- App capability: normal app path using `IpSecManager.openUdpEncapsulationSocket()` and `ConnectivityManager.createSocketKeepalive(...)`. No root, ADB, dangerous runtime permission, JNI, hidden API, raw Binder, or `PACKET_KEEPALIVE_OFFLOAD` is required for the public-API claim.
- External observation point: separate OpenWrt AP/router capture on the physical Wi-Fi side. Packets observed there have already crossed Android's VPN policy boundary.
- Packet form: fixed NAT-T UDP/4500 keepalive with a one-byte payload. The primitive cannot carry arbitrary application payloads.
- Cadence and slots: the public minimum interval was observed. On the tested Pixel Wi-Fi configuration, one unprivileged slot was accepted and later slot attempts failed with `ERROR_INSUFFICIENT_RESOURCES (-32)`. The Samsung row records one active slot and a measured lease. The Nothing row records one active slot without a duration measurement.

A.2 On-Wire Captures, Controls, and Baselines

- Primary Wi-Fi proof: app run `20260529-023621-pid20917`, router case `slot-cadence`, 18 packets, pcap prefix `d463ea0c9ea7`. Slot 0 accepted and UDP/4500 appeared on physical Wi-Fi at 10-second cadence.
- Ordinary UDP lockdown control: app run `20260529-023925-pid21402`, router case `ordinary-udp-lockdown`, zero matching packets, pcap prefix `e3f42e268763`. App-side UDP sends to UDP/4500 and UDP/12345 had no matching router packets under lockdown.
- VPN off, lockdown off baseline: app run `20260529-031517-pid13504`, router case `baseline-vpn-off-lockdown-off`, 9 packets, pcap prefix `4bc929aeaf54`. Ordinary UDP and keepalive packets were visible

when VPN confinement was disabled.

- VPN on, lockdown off baseline: app run `20260529-032140-pid14963`, router case `baseline-vpn-on-lockdown-off`, 6 packets, pcap prefix `840d5c418933`. Keepalive packets were visible on the router while ordinary UDP was app-side on `tun0`.
- VPN on, lockdown on baseline: app run `20260529-024101-pid21674`, router case `baseline-vpn-on-lockdown-on`, 6 packets, pcap prefix `732648acd332`. Keepalive packets remained visible while the ordinary UDP lockdown control was absent from the router capture.

Device/version	WLAN path	Slot result	Evidence boundary
Samsung SM-F966B (`q7q`), Android 16	Qualcomm (`qcom`) Wi-Fi	1 active / 24 h 32 min	Physical IPv4 default gateway selected; public callback active; no independent router capture.
Nothing A059 (`Asteroids`), Android 16 / SDK 36	Qualcomm (`qcom`) Wi-Fi	Highest/latest active: 1	Physical IPv4 default gateway selected; public callback active; no external capture or duration measurement.
Pixel 8 Pro, Android 17	Wi-Fi	1 active	Later-version slot availability only; distinct from the controlled Android 16 Pixel packet capture.

The older July 28 snapshot and sanitized screenshot supply the Samsung row. The newer non-cumulative snapshot supplies the Nothing and Android 17 Pixel rows. The tracked protector defines physical-gateway selection and active-callback state; the tracked report factory publishes only maxima greater than zero. The raw snapshots remain in the read-only local mirror, and the table reproduces only the sanitized fields [13; 14; 15; 16; 17].

A.3 Lifecycle and Boundary Rows

- Nondestructive lifecycle: app runs `20260529-013941-pid14044` and `20260529-021757-pid16937`, with lifecycle rows under `active-lifecycle-*`. Background/home, screen lock, force idle, battery saver, restricted bucket, and process-observation windows kept app heartbeats alive; early router windows were workflow evidence rather than standalone on-wire persistence proof.
- Force stop: app run `20260529-024326-pid21970`, router case `active-lifecycle-force-stop`, 36 packets, pcap prefix `dc264c7d08a8`. Packets were present before the host force-stop killed the process.
- Uninstall: app run `20260529-025027-pid23321`, router case `active-lifecycle-uninstall`, 42 packets, pcap prefix `12d0a2a140e7`. Packets were present before package removal.
- Reboot: app run `20260529-025521-pid24592`, router case `active-lifecycle-reboot`, 27 packets, pcap prefix `156a67de67da`. Packets were present before reboot.
- Network loss and re-arm: app run `20260529-031051-pid12326`, router case `network-loss-rearm-manual`, 6 packets, pcap prefix `63a9bec9367f`. Wi-Fi loss caused `onError -20`; Wi-Fi return and manual re-arm restored the keepalive.
- Back-to-BFU: the Pixel lifecycle observations include GrapheneOS relock/back-to-BFU-without-reboot continuation; this is not reboot persistence.

A.4 Destination Fuzzing

Destination fuzzing tested address-class filtering after keepalive admission. IPv4 destination classes were broadly accepted and confirmed on-wire; IPv6 behavior was route-driven. These results inform consequence and hardening analysis. The VPN-lockdown bypass requires only the validated public endpoint.

IPv4 group	Destinations tested	Result
Public baseline	`1.2.3.4`	Accepted, on-wire
This network	`0.0.0.0`	Accepted, on-wire
Limited broadcast	`255.255.255.255`	Accepted, on-wire
Multicast	`224.0.0.1`, `224.0.0.251`, `239.255.255.250`	Accepted, on-wire
Loopback	`127.0.0.1`	Accepted, on-wire
Link-local	`169.254.0.1`	Accepted, on-wire
CGNAT / RFC 6598	`100.64.0.1`	Accepted, on-wire
RFC1918 private	`10.0.0.0`, `172.16.0.1`, `192.168.0.1`	Accepted, on-wire
RFC5737 TEST-NET	`192.0.2.1`, `198.51.100.1`, `203.0.113.1`	Accepted, on-wire
Class E reserved	`240.0.0.1`	Accepted, on-wire

IPv6 acceptance followed route availability and Java address-family normalization. With a matching route, especially a `::/0` default route, 41 of 48 tested IPv6 destination cases were accepted. Missing routes produced route-selection failures. IPv4-mapped `::ffff:*/96` forms were rejected with `ERROR_INVALID_IP_ADDRESS (-21)` because Java address parsing normalized them into `Inet4Address`, causing a family mismatch. IPv4-translated and NAT64-prefix forms were accepted in the route-present matrix.

IPv6 category	Examples / notes	Result in route-present setup
Link-local unicast	`fe80::1`, `fe80::2`, longer link-local examples	Accepted via on-link route
ULA	`fc00::1`, `fd00::1`, `fd12:...`, `fdff:...`	Accepted
Global unicast	Google and Cloudflare resolver examples	Accepted
Documentation	`2001:db8::1`, `2001:db8:ffff:ffff::1`	Accepted
Teredo / 6to4 / ORCHID-v2	`2001::1`, `2002::1`, `2001:20::1`	Accepted
Loopback	`::1`	Accepted
Unspecified	`::`	Accepted
Discard	`100::1`	Accepted
IPv4-translated	`::ffff:0:1.2.3.4`	Accepted
NAT64 well-known prefix	`64:ff9b::1.2.3.4`, `64:ff9b::127.0.0.1`, `64:ff9b::0.0.0.0`, `64:ff9b::255.255.255.255`	Accepted
NAT64 alternate prefix	`64:ff9b:1::1.2.3.4`	Accepted
Multicast	Interface-local, link-local, site-local, organization-local, and global examples	Accepted
IPv4-mapped	`::ffff:1.2.3.4` and related IPv4-mapped forms	Rejected with `-21` due Java normalization/family mismatch

A.5 Diagnostic and Precondition Runs

Android callback and error codes establish preconditions. Only the external AP/router capture establishes on-wire emission.

Code	Interpretation
`0`	Accepted callback; external AP/router capture is still required for on-wire proof.
`-20`	Network lost or keepalive stopped because the active network changed.
`-21`	Invalid IP, missing route, or address-family mismatch.
`-22`	Invalid or unbound source port.
`-24`	Invalid keepalive interval.
`-30`	Unsupported network type or supported keepalive count zero.
`-32`	Insufficient resources or slot gate.
`-994`	Local socket create/bind failure in fuzz app diagnostics.

The second-round fuzz run had 41 `trial_finished` rows, 40 errors, one skip, 26 `-32` slot-gate results, 9 `-24` interval results, 2 `-22` source-port results, and 3 local `-994` socket failures. These counts characterize negative and precondition runs; they do not establish Wi-Fi destination breadth.

Raw Binder/resource diagnostics used non-null fds with resource IDs including `0`, `-2`, `Integer.MAX_VALUE`, and `Integer.MIN_VALUE`. Those values were not treated as ownership proofs. The diagnostics support the authenticity finding; the VPN-lockdown result uses the documented public API.

17. References

1. Nian Xue, Yashaswi Malla, Zihang Xia, Christina Poepper, and Mathy Vanhoef (2023). Bypassing Tunnels: Leaking VPN Client Traffic by Abusing Routing Tables. Proceedings of the 32nd USENIX Security Symposium. USENIX Association. Source.
2. Vasile C. Perta, Marco V. Barbera, Gareth Tyson, Hamed Haddadi, and Alessandro Mei (2015). A Glance through the VPN Looking Glass: IPv6 Leakage and DNS Hijacking in Commercial VPN Clients. Proceedings on Privacy Enhancing Technologies, 2015(1), pp. 77–91. DOI: 10.1515/popets-2015-0006. Source.
3. Nasser Mohammed Al-Fannah (2017). One Leak Will Sink A Ship: WebRTC IP Address Leaks. Source.
4. Mohammad Taha Khan, Joe DeBlasio, Geoffrey M. Voelker, Alex C. Snoeren, Chris Kanich, and Narseo Vallina-Rodriguez (2018). An Empirical Analysis of the Commercial VPN Ecosystem. Proceedings of the Internet Measurement Conference. Association for Computing Machinery. DOI: 10.1145/3278532.3278570. Source.
5. Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi (2022). VPNalyzer: Systematic Investigation of the VPN Ecosystem. Proceedings of the Network and Distributed System Security Symposium. Internet Society. Source.
6. Yejin Cho and John Heidemann (2025). Smoothing Rough Edges of IPv6 in VPNs. Source.
7. Yuxiang Yang, Ao Wang, Xuewei Feng, Qi Li, and Ke Xu (2026). Invisible Adversaries: A Systematic Study of Session Manipulation Attacks on VPNs. Source.
8. Android Developers (2026). SocketKeepalive. Android API reference. Source. Accessed 2026-05-30.
9. Android Open Source Project (2026). ConnectivityManager.java. AOSP source, packages/modules/Connectivity, commit 2519a78731526d2eb20ae8812acdcab6ef7a09b6. Source. Commit-pinned; accessed/rechecked 2026-06-07.
10. Android Open Source Project (2026). NattSocketKeepalive.java. AOSP source, packages/modules/Connectivity, commit 2519a78731526d2eb20ae8812acdcab6ef7a09b6. Source. Commit-pinned; accessed/rechecked 2026-06-07.

11. Android Open Source Project (2026). ConnectivityService.java. AOSP source, packages/modules/Connectivity, commit 2519a78731526d2eb20ae8812acdcab6ef7a09b6. Source. Commit-pinned; accessed/rechecked 2026-06-07.
12. Android Open Source Project (2026). KeepaliveTracker.java. AOSP source, packages/modules/Connectivity, commit 2519a78731526d2eb20ae8812acdcab6ef7a09b6. Source. Commit-pinned; accessed/rechecked 2026-06-07.
13. Armin Šupuk (2026). VPN Leak Guard NAT-T Keepalive Protector. Unpublished Android source implementation, NattKeepaliveProtector.kt; SHA-256 ab7a14c28c441ef36029afb6e0eaf45ca34d66e86fe5a18aeb722817fcca2443; inspected 2026-07-28. On file with the author; available on request.
14. Armin Šupuk (2026). VPN Leak Guard NAT-T Observation Report Factory. Unpublished Android source implementation, NattContribution.kt; SHA-256 b8942b159ebe27ae178f16f464280b804ae54c04420f589c4b458dae23e12d9f; inspected 2026-07-28. On file with the author; available on request.
15. Local NAT-T keepalive observation database (2026). Samsung SM-F966B Wi-Fi Keepalive Observation Snapshot. Unpublished read-only observation-database snapshot, 2026-07-28T09:46:16Z; SHA-256 56e15294ba602df454f09f597850364b8ef7bd355b0ad8e7f6038c75766d290f; only sanitized device, build, transport, and slot-count facts are reproduced. On file with the author; available on request.
16. Local NAT-T keepalive observation database (2026). Nothing A059 and Pixel 8 Pro Wi-Fi Keepalive Observation Snapshot. Unpublished read-only observation-database snapshot, 2026-07-28T15:06:09Z; SHA-256 0465dfdff9104f6a4836c299e2c96203b0935417d5564edef852c15eafc1fda3; only the sanitized Nothing A059 runtime fields and Pixel 8 Pro Android 17 single-slot provenance row described in Appendix A are reproduced. On file with the author; available on request.
17. VPN Leak Guard (2026). Sanitized Samsung NAT-T Active-Slot Lease Screenshot. Published evidence image. Source. SHA-256 fd1466ff3c897190708d7cf9f35f4d02532e48e5d3846cab250b0a3667b8bb57; records 24 h 32 min lease uptime.
18. Jumana, Parjanya Vyas, and Yousra Aafer (2025). Red Light for Security: Uncovering Auto Feature Check and Access Control Gaps in AAOS. Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 147–166. Springer Nature Switzerland. DOI: 10.1007/978-3-031-97623-0_9. Source.
19. Jumana (2025). Analyzing Access Control Logic in the Android Automotive Framework. University of Waterloo. Source.
20. Parjanya Vyas (2025). Cues, Clones, and Cars: Access Control Issues in Customized Android. University of Waterloo. Source.
21. Android Developers (2026). VPN. Android Developers documentation. Source. Accessed 2026-05-30.
22. Android Developers (2026). DevicePolicyManager. Android API reference. Source. Accessed 2026-05-30.
23. Android Developers (2026). VpnService. Android API reference. Source. Preparation flow and BIND_VPN_SERVICE declaration; accessed 2026-07-11.
24. Android Open Source Project (2026). Android 17 Vpn.java. AOSP source, frameworks/base, commit 94b4c163b7dfe5ce3607f7bb8456f9573f7de57d. Source. Commit-pinned; accessed 2026-07-11.
25. Android Open Source Project (2026). Android 16 Compatibility Definition. Android compatibility documentation. Source. Accessed 2026-05-30.
26. Android Open Source Project (2026). IWifiStaIface.aidl. AOSP source, hardware/interfaces, commit 1a56e38edc2f2f6189ef405ee1edce554e15cbc0. Source. Commit-pinned; accessed/rechecked 2026-06-07.
27. Google Pixel Help (2026). Pixel Phone Hardware Tech Specs. Google support documentation. Source. Accessed 2026-05-30.
28. Android Open Source Project (2026). packages/modules/Connectivity Gitiles Repository. AOSP source repository. Source. Commit IDs in the paper source-history table returned 200 from Gitiles on 2026-05-30.
29. Microsoft Learn (2023). Protocol Offloads for NDIS Power Management. Windows driver documentation. Source. Documents Windows 7 / NDIS 6.20 protocol offloads; accessed 2026-07-03.
30. IEEE Standards Association (2011). IEEE 802.11v-2011: IEEE Standard for Information Technology–Wireless LAN Medium Access Control and Physical Layer Specifications Amendment: Wireless Network Management. IEEE standards page. Source. Published 2011-02-09; accessed 2026-07-03.
31. Motorola Mobility LLC (2014). Qualcomm qcacld-2.0 WMI Keepalive and IPsec NAT Keepalive Definitions. GitHub source snapshot. Source. Commit e6075b10e7ce876ba4fd86601fe58aa5791831bd, 2014-10-25; accessed 2026-07-03.
32. Android Open Source Project (2015). ConnectivityManager.java, Android 6-era NAT-T Keepalive Source. AOSP source, frameworks/base. Source. Hidden PacketKeepalive and startNattKeepalive source; accessed 2026-07-03.

33. Android Open Source Project (2019). Android 10 Compatibility Definition. Android compatibility documentation. Source. Wi-Fi Keepalive Offload section accessed 2026-07-03.
34. TechInsights (2023). Google Pixel 8 Pro Component Analysis. TechInsights component analysis. Source. Accessed 2026-05-30.
35. Broadcom (2022). Broadcom Announces Availability of World's First Wi-Fi 7 Ecosystem Solutions. Broadcom press release. Source. Accessed 2026-05-30.
36. Local firmware census (2026). Slot Overlay Checks. Unpublished firmware-census evidence note, slot-overlay-checks.md; accessed 2026-07-03. On file with the author; available on request.
37. Local firmware census (2026). Chipset Provider Keepalive Support Timeline. Unpublished firmware-census evidence note, chipset-support-timeline.md; accessed 2026-07-03. On file with the author; available on request.
38. Android Developers Blog (2021). Android 12 Is Live in AOSP. Android Developers Blog. Source. Accessed 2026-07-03.
39. Local firmware census (2026). Android 12+ Firmware Family Coverage Report. Unpublished firmware-census evidence note, android12-market-map/coverage-report.md; generated 2026-07-02. On file with the author; available on request.
40. F-Droid Project (2026). All Our APIs: The F-Droid Repository Index. F-Droid documentation. Source. Signed v1 index documentation; accessed 2026-07-11.
41. IzzyOnDroid (2026). IzzyOnDroid F-Droid Repository. Official repository documentation. Source. F-Droid-compatible open-source Android repository; accessed 2026-07-11.
42. Armin Šupuk (2026). Play Store IPsec/IKEv2 App Census with DEX Call-Site Verification. Unpublished reproducible research report, play-ipsec-ikev2-census/report.md; commit a6c0ef59790e2ec54fbc19e201e38fbdcaa56974; SHA-256 8868346dfafd8bbb7712f524872002441056da3c1dcded6197424df0c6ae0bc5; 29 discovered candidates, 26 current listings, 3 removed apps, 9 acquired APKs, 17 listing-only candidates, 47 verified platform-SDK DEX call sites across 2 proprietary APKs, 4,134,648 cumulative installs for the 2 API-containing apps, and exact APK hashes; generated 2026-07-29. On file with the author; available on request.
43. Android Open Source Project (2026). Connectivity Service Resource Configuration. AOSP source, packages/modules/Connectivity, commit 2519a78731526d2eb20ae8812acdcab6ef7a09b6. Source. Commit-pinned; accessed/rechecked 2026-06-07.
44. Nian Xue, Yashaswi Malla, Zihang Xia, Christina Poepper, and Mathy Vanhoef (2023). TunnelCrack. Project website. Source. Accessed 2026-05-30.
45. Leviathan Security Group (2024). TunnelVision. Project website. Source. Accessed 2026-05-30.
46. Leviathan Security Group (2024). TunnelVision: Routing Traffic Without VPN Encryption Using DHCP Option 121. Technical blog. Source. Accessed 2026-05-30.
47. Reethika Ramesh, Anjali Vyas, and Roya Ensafi (2023). "All of Them Claim to Be the Best": Multi-Perspective Study of VPN Users and VPN Providers. Source.
48. Ka Lok Wu, Man Hong Hue, Ngai Man Poon, Kin Man Leung, Wai Yin Po, Kin Ting Wong, Sze Ho Hui, and Sze Yiu Chau (2023). Back to School: On the (In)Security of Academic VPNs. Proceedings of the 32nd USENIX Security Symposium. USENIX Association. Source.
49. Low Level Academy (2026). Android QUIC Close-Payload Delegated UDP Send. Technical blog. Source. Accessed 2026-05-30.
50. GrapheneOS (2026). GrapheneOS Releases: 2026050400. GrapheneOS release notes. Source. Accessed 2026-05-30.
51. GrapheneOS (2026). Disable Close-QUIC Optimization Due to VPN Bypass Concern. GrapheneOS packages/modules/Connectivity commit. Source. Accessed 2026-05-30.
52. Mullvad VPN (2022). Android Leaks Connectivity Check Traffic. Mullvad blog. Source. Accessed 2026-05-30.
53. Mullvad VPN (2024). DNS Traffic Can Leak Outside the VPN Tunnel on Android. Mullvad blog. Source. Accessed 2026-07-29.
54. GrapheneOS (2024). GrapheneOS OS Issue Tracker: VPN Leak Blocking and Multicast/Local-Link Traffic. GitHub issue. Source. Accessed 2026-05-30.
55. GrapheneOS Community (2024). GrapheneOS Fixing the Standard VPN Leak Blocking Is Nearing Completion. GrapheneOS discussion forum. Source. Accessed 2026-05-30.
56. Android Open Source Project (2026). IPsec/IKEv2 Library. Android platform documentation. Source. Platform IKEv2, IMS, IWLAN, and VPN roles; accessed 2026-07-29.

57. Android Open Source Project (2026). EpdgTunnelManager.java. AOSP source, packages/services/Iwlan, commit b9ec687fee0110ee303921431a1eca2b1d35d500. Source. Carrier IWLAN IKE session and IPsec tunnel-interface use; accessed 2026-07-29.
58. Android Open Source Project (2026). VcnGatewayConnection.java. AOSP source, packages/modules/Connectivity, commit 347fbd34b368d19f0d87e908ea101eed3601a731. Source. VCN IPsec tunnel transforms and IKE session migration; accessed 2026-07-29.
59. Google (2025). The Android Show: I/O Edition. Google blog. Source. Published 2025-05-13; reports more than 3 billion active Android devices in over 190 countries; accessed 2026-07-29.
60. Mullvad VPN (2026). Any App on Recent Android Versions Can Leak Certain Traffic. Mullvad blog. Source. Accessed 2026-07-29.
61. IVPN (2026). Android VPN Leak via QUIC. IVPN knowledge base. Source. Independent reproduction; accessed 2026-07-29.
62. GrapheneOS Community (2024). Are VPN Leaks a Ghost of the Past from Now On? GrapheneOS discussion forum, response 6. Source. External-router upstream-tunnel guidance; accessed 2026-07-29.