MVPNalyzer: An Investigative Framework for Auditing the Security & Privacy of Mobile VPNs

Wayne Wang*
University of Michigan
wswang@umich.edu

Aaron Ortwein*
University of Michigan
aortwein@umich.edu

Enrique Sobrados*
University of New Mexico
esobrados720@unm.edu

Robert Stanley
University of Michigan
rsta@umich.edu

Piyush Kumar Sharma
University of Michigan, IIT Delhi
piyush@iitd.ac.in

Afsah Anwar
University of New Mexico
afsah@unm.edu

Roya Ensafi
University of Michigan
ensafi@umich.edu

***Abstract*—Mobile users increasingly rely on Virtual Private Networks (VPNs) to protect themselves from tracking, surveillance, and censorship. VPN apps operate from a privileged position by requiring interception of user traffic. While this safeguards end user traffic from malicious network intermediaries (*e.g.* surveilling ISPs), it leads to a critical "transfer of trust" from such network intermediaries to VPN providers. Yet, despite the sensitivity of this role, VPN apps, especially on mobile platforms, remain insufficiently audited.**

In this work, we present *MVPNalyzer*, an extensible framework for systematically analyzing Android VPN apps. Designed to handle the unique challenges of the Android VPN ecosystem, *MVPNalyzer* enables detailed investigation of VPN applications' behavior across the network layers. We apply our framework to 281 popular VPN apps from the Google Play Store and uncover fundamental and critical issues: 61 apps transmit unencrypted data, with 5 sending sensitive VPN configuration files in cleartext, allowing an attacker to hijack the VPN tunnel connection; 29 apps leak user traffic (including DNS) outside the tunnel; 143 apps fail to obfuscate the traffic to avoid trivial blocking; 76 apps transmit Advertising ID, the device-unique ID widely used for device and user tracking; and 107 apps fail to implement the best security practices in their VPN configuration files. Collectively, these apps have hundreds of millions of installs, highlighting the scale of users being impacted. Our findings reveal a troubling pattern of developer negligence, highlighting how poor enforcement, transparency, and maintenance practices continue to undermine even fundamental security guarantees.

I. Introduction

In today's digital landscape, users navigate a minefield of privacy and security threats that follow their every click. The metadata trailing behind a single web request can reveal one's most intimate secrets. For example, a domain name by itself could silently convey one's sexual orientation, health conditions, or political allegiances to unseen watchers [1]–[4]. This digital shadow has escalated from being an abstract concern to a tangible danger; intelligence agencies and corporations weaponize metadata for mass surveillance and behavioral manipulation with frightening precision [5]–[7]. As a result, users live in constant paranoia, knowing their digital footprints are being harvested, analyzed, and exploited mostly without consent [8], [9].

Over time, Virtual Private Networks (VPNs), now fueling a growing multi-billion dollar industry [10], have found their way into everyday users' toolboxes as a crucial defense against such threats. For millions of users, they represent a simple fix to a complex security and privacy problem—recent statistics demonstrate that 82% VPN users believe that VPNs keep them anonymous [11]. However, this comes at a cost: the VPN app sits in a privileged position, viewing, intercepting, and handling all user traffic. This underscores the need for rigorous evaluation of VPN apps to ensure they uphold users' trust.

Previous work has demonstrated that VPNs have *caused* numerous security and privacy issues [12]–[18]. For instance, studies have demonstrated that some VPN apps turn end-user devices into a residential proxy network and monetize them on residential IP markets [19], [20]. However, most existing studies either perform limited analysis of a specific weakness (detectability, routing security, *etc.*) [15]–[17] or focus on desktop platforms [18], leaving a gap for systematic evaluation of the security and privacy issues of mobile VPN apps.

In this work, we develop *MVPNalyzer*, a systematic framework for analyzing Android VPN apps. *MVPNalyzer* is designed to be modular and extensible, supporting the periodic addition of new analyses and tests as privacy and security threats evolve over time. At a high level, the framework consists of three main modules—collection, processing, and analysis—and is structured to easily incorporate new streams of data and analysis.

Developing such a framework comes with numerous difficulties where performing certain analyses, while trivial on desktop apps, become much more challenging in the mobile ecosystem. For instance, Android provides complex routing table handling as part of their `VpnService` API, where app traffic can be routed not just based on the destination IP address and firewall marks, but also based on app IDs, making it challenging to trace the VPN apps' behavior [21].

*Authors contributed equally to this work.

Network and Distributed System Security (NDSS) Symposium 2026
23 - 27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.231573
www.ndss-symposium.org

MVPNalyzer handles this challenge, along with many others related to traffic decryption and the inherent restrictions of the Android platform in performing specific analyses.

We employ *MVPNalyzer* on a dataset of 281 most popular VPN apps and ask the question: *can users expect VPN providers to uphold fundamental security and privacy guarantees?* To that end, we conduct a detailed investigation geared towards five specific analyses: (1) whether apps properly tunnel all user traffic without leakage, (2) whether apps use secure and robust communication channels, (3) whether the apps use hardened security configurations, (4) whether apps exfiltrate sensitive user or device information to third parties, and (5) whether apps provide any kind of protection against detection, especially when they make strong claims of unblockability.

Our findings reveal alarming trends: VPN apps that appear as top results in user searches are not following even the bare minimum of security and privacy practices. Surprisingly, 61 VPNs still transfer unencrypted content, leaving them vulnerable to a range of injection attacks [22], [23]. In some cases, even crucial VPN configuration files are transmitted in cleartext, allowing an on-path adversary to trivially modify or replace them and redirect the client to unwittingly connect to an adversary-controlled VPN server. Additionally, despite the well-documented risks and long-standing awareness of DNS and traffic leaks, 29 VPNs still fail to provide such protection, defeating the main purpose of VPNs [13], [18]. Investigating the reasons behind leaks reveals that they often arise from developer negligence and misconfiguration (we found four apps that encapsulate user traffic, but do not encrypt it). Worse still, VPN configuration files directly reflect providers' security readiness and intentions. Among the apps where we found such files, more than half fail to implement basic security hardening measures to protect users against known threats. Overall, the VPN apps with uncovered security and privacy lapses have been installed over 2.2 billion times, putting a vast number of users at risk. We have disclosed the most exploitable findings to all VPN providers.

Our study highlights the need for a systematic solution to continuously evaluate mobile VPN apps, where we take a first step by developing a robust framework called *MVPNalyzer*. We envision the framework to be the *de-facto* system for analyzing mobile VPN apps going forward and provide better transparency for stakeholders to take appropriate actions.

II. Background and Related Work

VPNs have become widely adopted by users seeking to bypass geoblocking, circumvent network censorship, and protect their online activity from network observers such as ISPs [24], [25]. While these services advertise enhancing users' online security and privacy, their actual behavior remains opaque. In this section, we first describe how VPN applications operate on Android, followed by a review of prior research on the security and privacy analyses of VPN services.

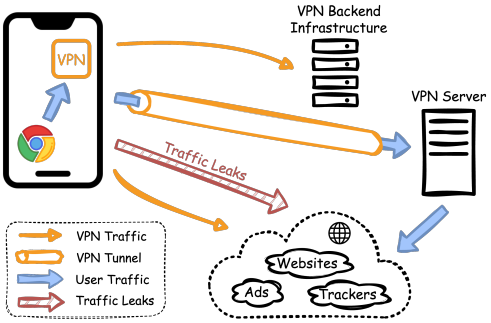


Fig. 1: Overview of traffic flows in a typical mobile VPN app. The VPN app 1) communicates with its own backend infrastructure, 2) contacts third-party services (e.g., ads and trackers), and 3) captures user traffic and tunnels it to a remote VPN server. Meanwhile, leakage might happen, exposing user traffic to observation and manipulation by on-path adversaries.

A. VPNs on Android

Mobile VPNs account for a substantial portion of the VPN ecosystem, with recent reports estimating that over 60% of VPN users access these services on mobile platforms [26], [27]. In contrast to desktop environments—where clients typically receive elevated privileges to configure routing tables and tunnel interfaces directly—Android restricts this access. To operate as a VPN, an Android app must declare the `BIND_VPN_SERVICE` permission and implement the `VpnService` API, which grants access to a virtual tunnel interface [28]. Users are notified of this access via a one-time approval dialog and a persistent system notification while the VPN is active [13]. Figure 1 illustrates the high-level structure of Android VPN traffic. In the bootstrapping phase before the tunnel is established, VPN apps generate outbound traffic to their backend infrastructure (e.g., to fetch server lists or deliver analytics), as well as to third-party services such as advertising or tracking platforms. Once the tunnel is established, traffic from other applications is redirected to the tunnel interface, allowing the VPN app to encapsulate it and send it through the tunnel; once the VPN server receives the packet inside the tunnel, it decapsulates it before forwarding to the destined service. While user traffic is expected to be encrypted and sent through the tunnel, leaks may occur, either unintentionally or by design, causing traffic to bypass the tunnel and reach its destination directly. All three categories of traffic—VPN-generated traffic, the tunnel connection itself, and any leaked user traffic—remain observable to a network adversary.

B. VPN Analysis

Despite VPNs' growing popularity, prior research has revealed vulnerabilities in client software, protocols, and server infrastructure [13], [16], [18], [29]–[32]. Ensuring full security of VPN applications is complex; the app must utilize secure communications with third party servers or its backend server for bootstrapping, securely configure routing tables to avoid

leakage, use an encrypted and securely configured tunnel, and, in many cases, obfuscate its traffic to avoid being detected.

Studies in the past have uncovered significant security and privacy concerns. Perta et al. [33] identified IPv6 and DNS leaks in their analysis of 14 popular VPNs. Khan et al. [14] revealed alarming practices such as leaking user traffic in various ways, and misrepresenting VPN server locations in their evaluation of 62 commercial VPN providers. Ramesh et al. [18] systematically analyzed popular desktop VPN providers and reported findings including traffic leaks and DNS insecurity. Prior work also identified specific vulnerabilities at the protocol and routing levels. Xue et al. [17] demonstrated widespread vulnerabilities caused by VPN client misconfigurations in routing tables, enabling traffic leaks outside the secure tunnel across various platforms and VPN services. In an investigation by Xue et al. [30], despite obfuscation claims, OpenVPN was shown to be vulnerable to fingerprinting attacks through observable packet characteristics, enabling targeted blocking.

Few studies [13], [31], [34] have explicitly focused on mobile VPNs. Ikram et al. [13] conducted the first analysis of Android VPNs in 2016, analyzing 283 VPN-enabled apps with runtime analysis on 150 of them, focusing primarily on privacy violations through third-party tracking and TLS interception using static analysis combined with limited dynamic testing. Wilson et al. [31] extended mobile VPN research to iOS, primarily investigating the unencrypted network traffic of 57 VPNs. Their work highlighted that many apps fail to encrypt sensitive information such as passwords and VPN configuration files. Zhang et al. [34] took a protocol-focused approach, analyzing OpenVPN configuration files used by Android VPN apps and uncovering systematic misconfigurations.

Overall, prior studies often focus on isolated components, smaller VPN datasets, or desktop platforms. In this study, we develop *MVPNalyzer*, a systematic, modular, and extensible framework for conducting large-scale and continuous evaluation of mobile VPN applications.

III. *MVPNalyzer* Design

We design the *MVPNalyzer* framework to systematically and comprehensively analyze the security and privacy posture of mobile VPN apps. The overall design of *MVPNalyzer* is depicted in Figure 2. Our primary emphasis is towards making the framework modular and extensible; as poor security and privacy practices evolve over time, new analyses and tests must be added periodically. Thus, the framework is structured to facilitate the addition of new streams of data collection and data analysis.

At a high-level, *MVPNalyzer* consists of three modules: (1) Data Collection, (2) Data Processing, and (3) Data Analysis. The Data Collection module is responsible for collecting different types of attributes and data associated with the app, including app metadata, network traffic (both encrypted and decrypted), local storage, *etc.* The Data Processing module converts the raw data into an analyzable format. For instance, the collected network traffic includes all activity from the mobile device; for a finer analysis, we developed a special attribution pipeline to identify the traffic generated specifically by the VPN app. Lastly, the Data Analysis module works on the raw and processed data to extract meaningful information and security violations by the VPN apps. Each module works independently and can be easily extended to add newer data pipelines and/or analyses. In this work, we show the usefulness of the framework in identifying various security and privacy violations and the potential for it to be the *de-facto* system for analyzing mobile VPN apps going forward.

In the following sections we detail each module of the *MVPNalyzer* framework.

IV. Collection & Processing

Raw data collection is the most essential component of the *MVPNalyzer* framework. However, such raw data may not always be suited for extracting meaningful information and performing nuanced analysis. Thus, we perform an additional data processing step, which helps us streamline our subsequent analysis pipeline. This section details both the data collection and processing steps.

A. Data Collection

To capture VPN apps' malpractices, we require detailed visibility into their functioning as well as evidence of such practices (e.g., in network traces of such apps). *MVPNalyzer*'s Data Collection module collects different types of data to facilitate a fine-grained analysis. Specifically, for a given app, we collect app metadata from the Google Play Store, network traffic (both inside and outside the VPN tunnel), TLS session keys, opened sockets, and local storage (including caches). Overall, the data collection consists of four steps: (1) app metadata collection, (2) instrumentation and configurations, (3) app execution and tunnel establishment, and (4) user interaction with and traffic generation via the VPN app.

1) App Metadata Collection: Before running and testing a VPN application, we first obtain its metadata from the Google Play Store. Using the `google_play_scraper` library [35], we retrieve a wide array of metadata, including but not limited to the app description, developer information, ratings, and number of installs.

2) Instrumentation & Configurations: The core functionality of *MVPNalyzer* requires visibility into the encrypted traffic generated by the VPN app. Multiple communication vectors, such as backend API interactions, configuration exchanges, and contact with third-party advertisement and tracking services, are all conducted over the network.

Traffic Decryption: Gaining visibility into TLS traffic requires the capability to decrypt it. However, common approaches to TLS decryption on Android are becoming difficult due to platform and app-level hardening. One popular method is to install a self-signed root certificate and run a flow-terminating proxy (e.g., `mitmproxy`) [36]. However, since Android 7.0, apps no longer trust user-added root certificates by default [37]. Further, the system-provided (and thus trusted) certificate store is on a read-only disk partition, which cannot be remounted as read-write on production phones even with

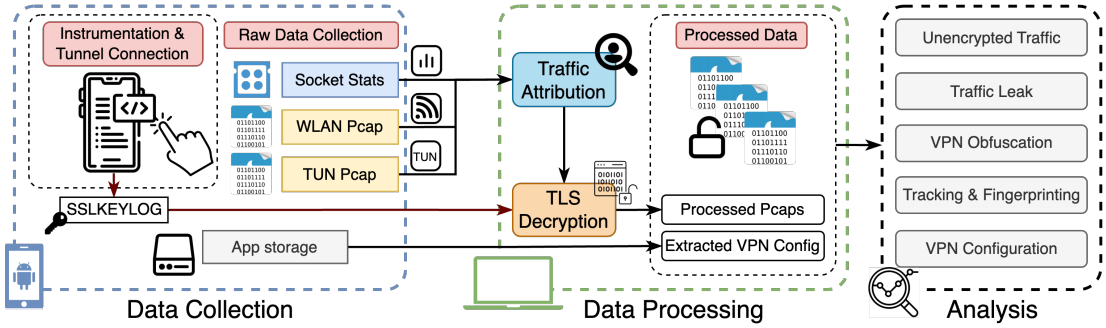


Fig. 2: High-level architecture of the *MVPNalyzer* framework, comprising of three engines: data **collection**, data **processing**, and data **analysis**. The system captures raw traffic captures, socket metadata, TLS keys, and app storage, processes it via attribution and decryption, and supports modular analysis to detect security and privacy issues in Android VPN apps.

root access. Even if we succeeded in installing a self-signed root certificate to the system store, we still may not be able to decrypt traffic due to certificate pinning [38], which causes the app to terminate TLS connections if the server presents a certificate that does not match the expected hashes. For this reason, mobile reverse engineering tools such as Frida are commonly used to bypass known implementations of certificate pinning [39], [40]. Unfortunately, Frida's popularity has motivated apps to detect its use in a number of ways, such as by detecting the `ptrace` system call [41], [42]. On top of that, Frida is known to crash under certain circumstances [43].

Thus, we take a more robust approach to traffic decryption, utilizing function hooking through the `LD_PRELOAD` environment variable [44]. The process involves compiling a shared library that redefines `SSL_new`, an OpenSSL function that is invoked when a new SSL connection is initiated and creates the data structure to store the SSL connection state [45]. We then set the `LD_PRELOAD` environment variable to point to our shared library. When `SSL_new` is called, the dynamic linker resolves the symbol to our custom implementation of `SSL_new` instead of the real implementation in legitimate OpenSSL libraries. Our hook logs TLS session keys—which can be used to decrypt packet captures—before dispatching execution back to the real `SSL_new`. This approach avoids limitations of other popular traffic decryption methods and is simple, portable, and largely transparent to the app. However, this approach also presents some implementation challenges. `LD_PRELOAD` purely uses the function name to hook and is not aware of where the original function is defined; returning program execution to the wrong library can thus cause the app to crash. To handle this, inspired by [46], we examine the call stack at runtime to determine which library the intercepted call originated from, and ensure that we dispatch execution back to the correct function.

Socket Statistics: We continuously record socket information throughout the app's execution by repeatedly invoking the `ss` utility [47]. This allows us to capture all active sockets on the device, along with their associated five-tuples `(src_ip, dst_ip, src_port, dst_port, proto)` and the owning UID identifying the app responsible for the connection. Although this polling-based approach may miss very short-lived connections, it is sufficiently reliable for packet attribution (detailed in Section IV-B1). The collected socket information is saved alongside the packet captures and used during processing in Section IV-B1 to isolate VPN-generated traffic.

3) App Execution and Tunnel Establishment: Once the instrumentation and socket metadata collection infrastructure is in place, we proceed to execute the VPN app and initiate the connection to its VPN tunnel. After launching the VPN app, we manually interact with the app to navigate its UI and complete the VPN tunnel connection process. In contrast to prior Android analysis studies that use automated tools like `Monkey` to simulate random inputs [48], [49], our approach favors manual interaction due to the complexity of UI elements (e.g., closing pop-ups, watching ads, and in one case, playing with a virtual pet[1]) prior to tunnel connection. `Monkey`, while useful for exploring app code paths, generates random input events and therefore offers no guarantee that the VPN tunnel will be successfully established. This makes it unsuitable for our purposes, where tunnel establishment is the most critical prerequisite for capturing meaningful data for subsequent analysis. Once a tunnel connection is established, which we confirm by monitoring for the existence of a tunnel interface, we start a packet capture on the tunnel interface to observe traffic going inside the tunnel. This traffic represents all the flows that the app observed and tunneled to the VPN server.

4) User Traffic Generation: With the VPN tunnel active, we generate realistic traffic to help us analyze and reveal how the app handles user traffic. We simulate a common user scenario: browsing a diverse set of websites in a browser while connected to a VPN. This traffic forms the basis for detecting issues such as traffic leaks.

We begin by launching a second `tcpdump` process on the tunnel interface to capture in-tunnel traffic that is not visible on the wireless interface due to VPN tunnel encryption. The system then visits 50 URLs, selected from the Tranco list [50] by requesting domains and following redirects until reaching a

[1]Phone Guardian VPN (com.distimo.phoneguardian) requires the user to pet a virtual husky to connect to the VPN.

live web server, using the Chrome browser. These URLs reflect websites commonly accessed by legitimate VPN users, such as social media platforms blocked by some network adversaries (e.g., Facebook, X, YouTube) [51], [52], as well as services that enforce georestrictions (e.g., Netflix) [53]–[55]. In addition to popular URLs, we attempt to visit two non-existent, app-specific domains to help identify DNS leakage. Specifically, we compute the `md5` and `sha1` hashes of the package name and construct domains of the form `www.<hash>.com`. Because these domains are highly unlikely to be contacted by background processes, their appearance in out-of-tunnel DNS traffic provides a strong signal of DNS leakage.

Each website is loaded for six seconds to allow page content and embedded resources to fully load, ensuring the total active testing period after tunnel establishment exceeds five minutes, consistent with prior mobile app research [13], [48]. Once the visits are complete, we terminate the browser and clear its cache to ensure subsequent tests issue all network requests rather than retrieving cached content.

As part of the teardown process, we also extract auxiliary data such as device identifiers (e.g., Advertisement ID, IMEI) and the app's local storage files. Although our focus is on network traffic, this additional context can expose issues not evident from traffic alone. For example, VPN configuration files recovered from local storage may reveal tunnel insecurity, as described in Section V-E. We copy all three dedicated storage locations for the app: credential-encrypted storage (private storage accessible only after the device is unlocked), device-encrypted storage (private storage accessible even when the device is locked), and external storage (shared storage accessible to any apps granted the appropriate permissions) [56], [57]. Finally, all collected data is compressed and transferred back to the host machine for further processing and analysis.

B. Data Processing

Before analysis, the raw data that *MVPNalyzer* collects is processed and transformed into formats suitable for analysis. In this section, we describe four processing modules: identifying VPN-generated traffic, decrypting TLS flows using session keys, translating protocols (e.g., converting HTTP/2 traffic into HTTP/1.1) to ensure compatibility with analysis tools, and extracting VPN configuration files from local storage.

1) Traffic Attribution: Correctly attributing network traffic is essential to avoid mischaracterizing app behavior, particularly the information it transmits. Because multiple processes run concurrently, including system services and our browser for traffic generation, packet captures will also contain traffic generated by other apps. This challenge is compounded by split tunneling, where only some traffic is routed through the VPN while other flows bypass the tunnel. Without proper attribution, a flow carrying device fingerprints to a tracking service, for instance, cannot be definitively linked to the VPN app, as it may originate from another app on the device.

Thus, using the socket information collected in IV-A2, we filter each packet capture to retain only packets sent from a socket opened by the VPN application. Packets are matched to sockets by five-tuple, and sockets are mapped to applications using UIDs. We save the resulting *VPN-attributed* packets to a separate packet capture that can be used for further analysis.

2) TLS Decryption: Next, we decrypt TLS traffic in the *VPN-attributed* packet captures. Because both the wireless and tunnel interfaces may contain encrypted traffic generated by the VPN app, we decrypt traffic on both interfaces using the TLS keys output from our `LD_PRELOAD` instrumentation (described in Section IV-A2) and `tshark`. The decrypted traffic is then saved to a separate packet capture to avoid confusion with traffic that was originally unencrypted.

3) Protocol Translation: To support analysis using existing traffic analysis frameworks such as Zeek, we translate `HTTP/2` traffic to `HTTP/1.1`. This translation preserves the semantics of the data but enables existing frameworks to parse HTTP requests and responses.

4) Extracting VPN Configuration Files: To analyze VPN configuration files (detailed in section Section V-E), we extract them from the VPN app's local storage. While there are many tunneling protocols, we focus on OpenVPN as it is the most widely used protocol in our dataset (over 50% of apps). Although OpenVPN configuration files often use the `.ovpn` extension [58], only 8 out of 108 apps with such files in our dataset follow this convention. We therefore use an alternative approach, identifying configuration files by matching file contents against a minimal set of directives required to establish an OpenVPN connection (e.g., `client`, `remote`, `dev`, *etc.*).

V. Analysis

To comprehensively examine the security and privacy posture of Android VPN apps, *MVPNalyzer* applies five analysis modules that extract structured metrics from collected data.

A. Unencrypted Traffic

Mobile VPN applications are widely perceived as security- and privacy-enhancing tools, and they occupy a privileged position on the device where they observe and handle all user traffic. As a result, failures to encrypt the apps' own communications (e.g., fetching server lists and configuration files) expose users to heightened risks. Thus, this module aims to identify unencrypted traffic generated by VPN apps and to characterize the transmitted data and its destinations, which range from backend VPN control-plane servers to advertising and tracking infrastructure.

While any unencrypted traffic leaks data to on-path adversaries, the severity of potential attacks depends both on the content type [59] and how the endpoints handle it (e.g., integrity checks or sanitization can mitigate malicious modifications). We check for a range of content types, including web resources (HTML, JavaScript, and JSON) and specific files such as VPN configuration files. We note that HTML and JavaScript may be susceptible to code injection [22], [23], while JSON may contain sensitive information that an adversary can easily modify. Most concerning, however, is the unencrypted transmission of VPN configuration files, since

modifications can cause a client to connect to an adversary-controlled VPN server.

To identify unencrypted content types, we use Zeek [60], an open-source, event-based network traffic analysis framework widely used in security research and operations. Zeek parses the application-layer protocol and filters for HTTP traffic. For each HTTP message, we use Zeek's built-in signatures to determine the content type. We further refine the approach by searching for `<script>` tags in HTML to identify cases of embedded JavaScript, which the built-in signatures do not capture. To detect configuration files and JSON, we develop custom parsers in Spicy, a parser generator language whose parsers can be integrated into Zeek [61]. Our JSON parser implements the JSON grammar [62], and our configuration file parser is loosely modeled on the one used by the OpenVPN Android app [63]. All detected unencrypted content types are then stored for subsequent analysis.

B. Traffic Leaks

While VPN applications are expected to tunnel *all* user traffic securely, leaks remain a persistent issue in practice. Such leaks violate the core guarantee of VPNs and expose user data to network intermediaries. This module focuses on identifying three types of leaks: (1) *DNS leaks*, where DNS queries are sent outside of the tunnel; (2) *user traffic leaks*, where traffic from other applications fails to be tunneled; and (3) *unencrypted tunnels*, where user traffic is tunneled over unencrypted protocols like HTTP or SOCKS.

DNS Leaks: To detect DNS leaks, we monitor DNS requests on the wireless interface after tunnel establishment and check for requests to domains visited in Section IV-A4. On Android, DNS requests are typically issued by the system resolver rather than individual applications, which prevents our packet attribution method from identifying their source. Moreover, since *MVPNalyzer* visits popular domains from the Tranco list, incidental overlap with domains contacted by the VPN app itself is possible, so out-of-tunnel DNS queries do not always indicate a DNS leak. We therefore flag a DNS leak if either (1) a significant proportion (90%) of domains visited by *MVPNalyzer* appear in out-of-tunnel DNS traffic, or (2) any of the "unique hash" domains (as described in Section IV-A4) appear in DNS requests outside the tunnel.

User Traffic Leaks: When user traffic leaks outside the tunnel, the core security guarantee of a VPN is undermined, leaving traffic vulnerable to the same attacks as if no VPN were used. Because *MVPNalyzer* visits all websites over HTTPS, we detect user traffic leaks by inspecting the Server Name Indication (SNI) extension of TLS Client Hello messages in out-of-tunnel traffic for visited domains. The presence of these domains indicates a leak but does not reveal whether it is due to misconfigured routing rules or tunnel failure. To make this distinction, *MVPNalyzer* records whether the tunnel interface still exists at the end of testing; we classify traffic as leaked only if the tunnel interface remains active. Similar to DNS leaks, we conclude that a VPN app leaks user traffic if a large proportion (90%) of visited domains appear outside the tunnel. Finally, we note that the VPN app itself might send traffic outside the tunnel even when connected, but we do not consider this a leak since traffic originates from the VPN app rather than from the user's activities in other applications.

Unencrypted Tunnels: Tunneling user traffic through unencrypted protocols can conceal a user's IP address from an end server but leaves the traffic vulnerable to surveillance or blocking by network intermediaries. To detect unencrypted tunnels, we examine the transport-layer payloads of packets on the wireless interface. If a large proportion (90%) of visited domains appear in the payloads but not in standard protocol fields (e.g., TLS SNI or DNS query name), we infer that the app is tunneling traffic without encryption.

C. VPN Obfuscation

As VPNs are increasingly used for censorship circumvention, network adversaries, including ISPs and nation-state censors, have developed a range of techniques to detect and block VPN traffic [64]–[68]. While some VPN apps advertise resistance to blocking, such promises are only meaningful if measures are taken to avoid detection. In this module, we assess whether a VPN app implements *any* basic obfuscation to evade simple detection or blocking measures. Specifically, we test for three indicators of VPN traffic: (1) traffic over well-known VPN protocol ports, (2) protocol signatures recognizable by standard parsers, and (3) domains in DNS or TLS traffic that suggest contact with VPN servers (e.g., the presence of the string "vpn" or the package name in the domain).

Standard Ports: Traffic over standard and default ports used by circumvention tools, including VPNs, has previously been targeted for blocking [69], [70]. We therefore check for traffic on well-known VPN ports: `500/udp` and `4500/udp` for IPsec, `1194/udp` and `1194/tcp` for OpenVPN, `1701/udp` for L2TP, `1723/udp` for PPTP, and `51820/udp` for WireGuard.

Protocol Parsing: Well-known VPN protocols, when used without obfuscation, can be detected by adversaries using standard protocol parsers. To identify the use of such protocols, we leverage the protocol parsers built into nDPI [71], an open-source deep packet inspection (DPI) toolkit for traffic classification. Unlike alternative traffic analysis tools such as Wireshark, which often rely on port numbers for protocol identification, nDPI applies signature- and heuristic-based detection to identify protocols regardless of port, allowing us to detect VPN traffic even when apps use non-standard ports. Successful identification of a common tunnel protocol suggests that the VPN app makes a limited attempt at obfuscation.

VPN Domains: While tunnel traffic can be obfuscated to evade detection, VPN-related metadata may still reveal a user's VPN usage and potentially put them at risk. In particular, DNS queries and TLS Client Hello SNIs observed in out-of-tunnel traffic may indicate connections to VPN services. To detect this, we check DNS queries and TLS Client Hello SNI fields for domains containing either (1) the substring "vpn", which prior work has used to identify VPN usage [72], or (2) any portion of the application's package name. Because Android

application package names are conventionally structured as reverse domain names, as recommended in Java [73], we exclude common suffixes by removing package name segments that appear in the Public Suffix List [74].

D. Tracking and Fingerprinting

Advertising, tracking, and device fingerprinting are common monetization strategies for free VPN services [75]. However, when users transfer trust to VPN applications, they often expect increased privacy [12], [25]. In addition, VPNs have become part of the essential digital toolkit for users at risk of targeted surveillance [24]. Thus, tracking and fingerprinting by VPN applications undermine these privacy expectations. To evaluate apps' tracking and fingerprinting practices, we analyze outbound communications to advertisers and trackers, as well as the types of data transmitted.

This analysis requires examining all traffic generated by the VPN app—including cleartext and decrypted HTTPS traffic—using the attribution and decryption techniques described in Section IV-B. Because VPNs may generate their own traffic within the tunnel alongside user traffic, we inspect *VPN-attributed* packets on both the tunnel and wireless interfaces. We then extract contacted URLs and compare them against popular filter lists like EasyList and EasyPrivacy [76].

We also examine exfiltrated data to assess whether it could enable tracking or device fingerprinting. Following prior work [77], *MVPNalyzer* leverages the tendency of Android apps to transmit information in structured key-value pairs. We extract these pairs from three parts of the application-layer payload: (1) non-standard HTTP headers (i.e., excluding those defined in RFCs), (2) query string parameters in HTTP GET requests, and (3) HTTP POST request bodies. For POST requests, we restrict analysis to form-encoded or JSON data. These are identified using text patterns rather than the `Content-Type` header, allowing us to extract key-value pairs even when this header is missing or incorrect.

To characterize the type of data being exfiltrated, we categorize the extracted key–value pairs based on their semantic content. Specifically, we use regular expressions to search all extracted pairs for device attributes (make, model, IMEI, serial number, MAC address, screen size, Android version, Android API level, and Advertising ID), location information (IP address, coordinates, city, timezone, and country), and language information. Among these, the Advertising ID and IP address or geographic coordinates are particularly concerning, as they enable persistent and precise user tracking across sessions and services. The Advertising ID was introduced as a privacy-preserving alternative to hardware-bound identifiers like the IMEI and is designed to be resettable and, since 2021, deletable [78]. In practice, however, most users are unaware of its existence, let alone how or when to reset or delete it, which limits its effectiveness [79].

E. VPN Configuration

VPN apps use configuration files to define the settings of the tunnel. These files specify parameters through *directives*, which instruct the VPN client and server on how to establish or manage the tunnel (Figure 3 shows a sample configuration file). For example, the *remote* and *cipher* directives specify the server's IP address and encryption method, respectively.

```
client
dev tun
remote my.vpn.server 1194 udp
cipher AES-256-CBC
auth-user-pass
ca ca.crt
...
```

Fig. 3: OpenVPN client configuration file (`client.ovpn`). Each line contains a directive followed by zero or more parameters, which are shown in *italic*.

Each VPN protocol defines a different set of configuration directives. While some protocols have straightforward configurations, others offer more complex combinations to provide flexibility for specific use cases. However, without sufficient security awareness, this versatility can lead to misconfigurations that expose users to significant risks. We focus on analyzing OpenVPN configuration files because more than half the apps in our dataset default to OpenVPN.

We base our analysis on the latest OpenVPN version (2.6). We reviewed the reference manual [80] to identify directives with security implications, including security features, best practices, and hardening recommendations [81]. Across the configuration files in our dataset (refer to Section IV-B4), we find 18 directives with associated security risks. We use these directives to classify apps into four categories based on their compliance with best practices and recommendations.

Insecure Cryptography: It is important that VPN providers use secure ciphers that comply with OpenVPN recommendations, as weak algorithms can expose users to traffic decryption or machine-in-the-middle (MITM) attacks [34]. We identify weak cryptography through insecure cipher choices (e.g., Blowfish, 3DES) specified in the `cipher` or `data-ciphers` directives, as well as missing authentication in the `auth` directive.

Weak Authentication: Strong authentication prevents unauthorized access to VPN servers and protects other users sharing the infrastructure. OpenVPN supports username/-password and client-certificate authentication, and combining them as multi-factor authentication can significantly improve security [82]. We therefore check for both mechanisms: `auth-user-pass` for username/password authentication and `cert`/`key` for client-certificate authentication.

Deprecated Directives: OpenVPN regularly updates its directives—introducing new ones and deprecating or removing outdated ones that pose security risks—to align with state-of-the-art security practices. While OpenVPN still supports deprecated directives to maintain backward compatibility,

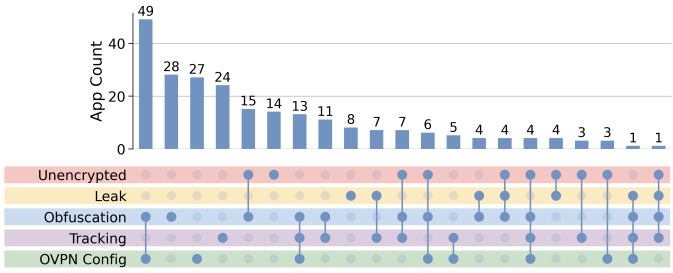


Fig. 4: Intersection of VPN app behaviors across the five analysis modules (traffic leaks, unencrypted content, VPN obfuscation, tracking/fingerprinting, and configuration issues).

adopting the latest security features should remain a priority for VPNs given their role as security- and privacy-enhancing tools. We quantify the use of directives marked as deprecated in OpenVPN documentation [83].

Hardening Options: OpenVPN provides several configuration options to mitigate known attacks against VPNs [80], [81]. While not required for functionality, their use demonstrates commitment to security. We therefore check for directives such as `tls-auth`, `tls-crypt`, and `tls-crypt-v2`, which authenticate control-channel messages, and `verify-x509-name` and `remote-cert-tls`, which verify the server's identity to prevent impersonation.

VI. Results

In this section, we present results from our systematic investigation of popular VPN apps using the *MVPNalyzer* framework. We follow the analysis outlined in Section V and report our findings across five categories. The aggregated results are shown in Figure 4.

To evaluate whether Android VPN apps uphold their advertised guarantees, we curated a dataset of 281 operational VPNs from the Google Play Store. We began by scraping search results for 40 VPN-related search terms (e.g., "fast VPN", "VPN for gamers", etc.), as well as the "VPN & Proxy Tools" app category, across all supported countries in November 2024. A complete list of the search terms is provided in Table I. Because the web interface returns at most 30 apps per query, this process yielded up to 1,200 results per country. Although searches within the Android Play Store app return more results, they remain incomplete and are significantly harder to collect given the complexity of robust Android UI automation. We then constructed a global set of unique apps across all search terms and regions. From this set, we automated downloads from the U.S. Play Store onto a physical Android 14 device. Apps that were not free to download were skipped, and successfully downloaded APKs were extracted for testing.

Despite using VPN-related keywords in our searches, not all apps requested the `BIND_VPN_SERVICE` permission; we examined only those that did. We excluded apps that required an account or in-app payment to use the service, provided only VPN client functionality [2], or required updates at the time of testing (to keep download dates consistent). We then tested the remaining apps, removing those that failed to launch or could not establish a VPN connection. Among the functioning VPN apps, our traffic-leak analysis revealed that some failed to tunnel browser traffic. We manually examined these apps and filtered out those explicitly claiming not to be general-purpose VPNs—that is, while they use Android's VPN interface, they either do not tunnel all user traffic (e.g., tunnel only to streaming services) or repurpose the interface for other network-management tasks (e.g., anti-virus apps that block connections to malicious websites or DNS changer apps that modify the default resolver).

Google Play Store Search Terms		
VPN	VPN app	Secure VPN
Fast VPN	Free VPN	Unlimited VPN
VPN proxy	VPN for privacy	VPN with no logs
VPN with split tunneling	VPN for streaming	VPN for gaming
VPN with kill switch	VPN with multiple servers	VPN with encryption
VPN for Android	VPN for public Wi-Fi	VPN for travel
VPN for torrenting	VPN for school	VPN for work
High-speed VPN	Low-latency VPN	Reliable VPN
Best free VPN	Premium VPN	Cheap VPN
Trial VPN	VPN for Netflix	VPN for sports streaming
VPN for YouTube	Private VPN	Anonymous VPN
VPN with strong encryption	Fast free VPN	Secure private VPN
Best VPN for streaming	Unlimited free VPN	No-logs secure VPN
VPN and proxy tools		

TABLE I: Complete list of VPN-related search terms queried across all supported countries on Google Play Store during our evaluation dataset construction.

We ran *MVPNalyzer*'s Data Collection module on OnePlus Nord 5G (CPH2513) devices running Android 14, the latest production version at the time of experimentation. Data Processing and Analysis were subsequently performed on a Linux machine running Ubuntu 22.04.

A. Unencrypted Traffic

In this subsection, we analyze the network traffic of mobile VPN apps that transmit unencrypted data outside the tunnel. We present our findings on the presence of such traffic, the types of data exposed, and their security implications.

Unencrypted Traffic Volume: Overall, we observed 10,552 unencrypted flows across 61 VPN apps. Figure 5 shows the number of unique unencrypted flows generated by each app, categorized by content type (i.e., HTML, JavaScript, JSON, and VPN configuration files). Notably, 32 apps generated at least five unencrypted flows, with `com.kylovpn` sending the most (2,075). Alarmingly, most apps transmitting unencrypted content are highly popular, averaging 11.16 million installs.

Categorizing Requested URLs: To further analyze the cleartext traffic sent by VPN apps, we examined the endpoints contacted in unencrypted web requests. We categorized these URLs using VirusTotal, selecting the ForcePoint engine as

[2]Apps that do not provide a VPN service but function solely as clients, requiring users to supply their own configuration files for external VPN servers not managed by the app.

Category	Count	Example
Information Technology	120	http://ip-api.com/json
Web Infrastructure	114	http://gstatic.com
Suspicious Content	47	http://txcdn-res.lifegram.cc/.../source-filter-draw.zip
Web Images	28	http://103.255.209.161/img/circle_news.png
Dynamic Content	27	http://api.dundundun.online/getconf?uid=...&...
Elevated Exposure	19	http://ip3.6016725.xyz/github.com/Alvin9999
Shopping	7	http://chunchuan.kejixiaoqi666.store/
Business & Economy	7	http://creators.trueid.net
Advertisements	6	http://tpc.googlesyndication.com/sodar/....html
Travel	5	http://us-host.non-vpn.com/world_v65/file
Proxy Avoidance	3	http://www.vpngate.net/api/iphone/
Entertainment	3	http://txcdn1-file-m.mvbox.cn/....jpg.mm.webp
VPN Infrastructure	105	http://download.tikvpn.in/.../America@3x.png

TABLE II: Categorization of all unique URLs found in cleartext traffic by the ForcePoint classification engine. VPN Infrastructure URLs were separately identified.

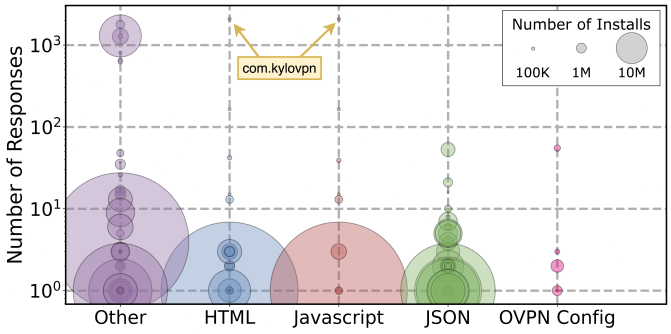


Fig. 5: Numbers of unencrypted flows for different content categories. Each dot represents an app.

the most effective for our dataset; it successfully categorized 391 of 604 URLs, summarized in Table II. Notably, 47 domains were classified as suspicious and 19 as elevated exposure (sites that "camouflage" their identity), based on URL reputation [84]. The most frequently contacted domain was `ip-api.com`, which is used by 18 applications for client IP geolocation. The free version of this API—intended for non-commercial use—is only available over HTTP [85], with HTTPS restricted to paid tiers. We further categorized URLs using the same string-matching technique described in Section V-C to identify those associated with VPN backend infrastructure, finding 105 such URLs across 21 apps. These findings show that some app developers, despite likely controlling their backend services, still opt for insecure communication.

Transparency: After observing such a large volume of unencrypted requests, we sought to determine whether VPN providers explicitly declare their intention to transmit cleartext traffic, or more importantly, whether they attempt to conceal it. To investigate this, we examined app-level declarations.

Android 6 introduced the `usesCleartextTraffic` flag, which is specified in the Android Manifest and allows developers to declare an intention to transmit cleartext (e.g., HTTP, WebSocket) traffic [86]. Since Android 7, developers have also been able to define a Network Security Configuration file [87], which provides finer-grained control over networking. Its `cleartextTrafficPermitted` flag likewise specifies whether cleartext connections are allowed, but unlike `usesCleartextTraffic`, it also permits exceptions for specific domains. Starting with Android 9, this flag is disabled by default, preventing cleartext connections [88]. Defining the Network Security Configuration is optional, but if present, `cleartextTrafficPermitted` takes precedence over `usesCleartextTraffic`.

We analyzed the Android Manifest and Network Security Configuration files of all apps to identify their cleartext settings. We found 203 apps that set either the `usesCleartextTraffic` or `cleartextTrafficPermitted` flag for *at least one* domain. Of these, 66 apps allowed cleartext traffic on *all* domains via `cleartextTrafficPermitted`, and 88 apps did so via `usesCleartextTraffic`. Among the 61 apps observed transmitting cleartext traffic, 53 declared their intention to do so.

The remaining 8 disabled both flags yet still transmitted cleartext traffic. Further investigation revealed that, although apps and libraries are expected to respect these flags, some low-level network APIs cannot enforce cleartext blocking because they cannot distinguish whether content is encrypted or not [86]. Thus, using low-level socket APIs can serve as a method for bypassing cleartext protections.

In summary, the transmission of cleartext traffic by VPN apps requires developers to actively circumvent multiple network security protections built into Android—either by disabling blocking of cleartext traffic by higher-level networking APIs or intentionally using low-level networking APIs that are incapable of blocking cleartext traffic.

Vulnerability to Injection: The presence of cleartext traffic in applications not only enables surveillance but also allows network intermediaries to modify user traffic, facilitating injection attacks that can compromise device security. The severity of such attacks depends on the type of data transferred [59]. Cleartext HTML and JavaScript enable adversaries to inject advertisements or malicious code [22], [23]. Similarly, key-value pairs in JSON data can be modified or replaced to dictate app behavior. VPN apps introduce yet another injection vector when client configuration files are transmitted in cleartext.

We identified 5 apps that receive configuration files in cleartext. Consequently, they are susceptible to tunnel hijacking, where a network intermediary induces a client to connect to an attacker-controlled VPN server. We confirmed the feasibility of this attack on VPN apps installed on mobile devices under our control, demonstrating both its practicality and its serious implications for users.

Leak Type	Apps
DNS Leak (24)	Java VPN, Noon VPN, AM TUNNEL LITE VPN, AM TUNNEL PRO, MahsaNG, GoFly VPN, Ostrich VPN, NewNode VPN, Cookie, Delight VPN, Phone Guardian, RoboProxy, Kylo Vpn, LVCHA VPN, XY VPN, Take Off, Tesla Proxy Pro, Global VPN, Air Net VPN, FoxoVPN, Bolt VPN, Free VPN, Siam VPN, Nine Tail VPN
Traffic Leak (6)	Java VPN, Noon VPN, NewNode VPN, Phone Guardian, Unicorn HTTPS, Free VPN
Unencrypted Tunnel (4)	Geo Tunnel, Raytunnel, Rosa VPN, V2net

TABLE III: Types of leakage with the corresponding VPNs.

Takeaway: Despite 98% of Internet traffic being encrypted as of April 2025 [89], we identified 61 VPN apps that still transmit cleartext traffic. Particularly concerning is the exposure of configuration files, which enables a network adversary to hijack the VPN tunnel. Notably, 8 apps bypass the cleartext protections of Android's networking APIs, showing that although the platform provides mechanisms to encourage secure communication, the lack of strict enforcement leaves exploitable gaps.

B. Traffic Leaks

Traffic leaks defeat one of the core purposes of VPNs: protecting user traffic from observation or tampering by network intermediaries. This subsection quantifies such leaks, focusing on DNS leaks, user traffic leaks, and unencrypted tunnels.

MVPNalyzer's Data Collection module visits 50 URLs through the browser. Ideally, if the domains of these sites appear in out-of-tunnel traffic, we classify it as a leak. However, the VPN app itself may also generate requests to those domains. To ensure that leaks reflect user traffic, we set a threshold: if we see 90% of the visited domains appear outside the tunnel, we classify the app as leaking traffic.

DNS Leaks: We identified 24 apps that leak DNS traffic. These apps have a combined 360 million installs, highlighting the scale of users whose traffic is exposed to network adversaries despite assumptions of VPN protection. Of these apps, 20 send requests to the client's local resolver and 6 send requests to public resolvers (Google, Cloudflare, AliDNS, and DNSPod). Because some apps use multiple resolvers, the sum of these counts exceeds the total number of leaking apps. The use of well-known public resolvers suggests that developers are aware of the risks of ISP resolvers, but this does not eliminate the possibility of DNS injection by on-path adversaries or malicious behavior by the resolvers themselves.

User Traffic Leaks: We found 6 apps leaking browser traffic outside the tunnel, as indicated by TLS Client Hello SNIs. On further investigation, we manually verified that the descriptions of these apps contain no explicit mention of split tunneling, where only a subset of user traffic is tunneled. Together, these apps account for 54 million installs, indicating that a significant number of users are affected.

Unencrypted Tunnels: Lastly, we identified 4 apps that contained the domains we visited in the transport layer payload of non-DNS, non-TLS traffic. This may occur when an app encapsulates web requests in transport layer payloads (e.g., in proxying protocols such as SOCKS) without encryption. We manually confirmed that these apps use unencrypted tunnels, thereby leaking visited domains.

Takeaway: Our findings demonstrate that leakage occurs across the network layers for 29 out of 281 apps. Very few of these apps are transparent about not tunneling certain user traffic, likely leaving users unaware of the true extent of protection provided by the VPN. The use of unencrypted tunnels in some cases highlights developers' negligence of fundamental security guarantees. Ultimately, traffic leaks undermine the core purpose of VPNs: to protect user traffic.

C. VPN Obfuscation

VPN traffic has been a target of blocking by many nation-state censors [52], [66], [67], [90]. While advanced state-of-the-art attacks exist for identifying sophisticated obfuscated VPNs [30], [91], [92], many censors block VPNs using simpler techniques such as standard port blocking or protocol parsing [64], [65], [68].

We investigate the susceptibility of VPN apps in our dataset to simple blocking measures. Specifically, we test 1) whether traffic is sent to standard VPN ports, 2) whether standard protocol parsers can identify the VPN tunnel protocol, and lastly 3) whether the DNS or TLS traffic is destined to domains containing the string "vpn". Using these techniques, we identified 143 apps as VPNs. Port checks identified 53 apps, the protocol parser identified 74 apps as using either OpenVPN, IPsec, or Wireguard, and the "vpn" string search identified 101 apps.

We corroborated our findings with the claims of these VPNs, identifying cases where VPNs advertise blocking resistance but make no effort to implement it. To this end, we analyzed the Google Play Store descriptions of the 143 apps vulnerable to trivial VPN detection. We found 38 apps making explicit claims of unblockability and 58 making implicit claims. Note that we categorize any direct mention of censorship circumvention as an explicit claim of unblockability (e.g.,"Bypass [...] internet filters and censorship"), while claims of unlimited access to blocked content are considered implicit (e.g.,"Unblock websites and restricted web content").

Takeaway: We found that more than half of the apps we analyzed were vulnerable to trivial VPN traffic detection using either standard open-source DPI solutions or simple string matching in DNS queries and SNI fields. Notably, about two-thirds of these VPNs claim, either implicitly or explicitly, to resist detection, thereby misleading end users.

D. Tracking and Fingerprinting

In this subsection, we characterize the prevalence of tracking and device fingerprinting behavior in the studied apps, focusing on the tracking URLs they contact and the device fingerprints they exfiltrate.

Category	Attribute	App Count	Example
Device	AdID	76	{“adid”: “...”}
	Make	176	{“make”: “OnePlus”}
	Model	210	{“model”: “CPH2513”}
	OS Type	209	{“os”: “android”}
	OS Version	177	{“osv”: “14”}
	Android API Level	184	{“android_api_level”: “34”}
	Display	28	{“screen_size”: “1080x2400”}
Location	Coordinates	1	{“lon”: “-00.0000000”}
	IP	38	{“ip”: “x.x.x.x”}
	City	3	{“city”: “xxxxxx”}
	Country	130	{“country”: “US”}
	Timezone	12	{“timezone”: “est”}
Language	Language	191	{“language_code”: “en-US”}

TABLE IV: Different types of device data sent to trackers.

Identifying Tracking Behavior: We extracted all URLs contacted by the VPN apps (both inside and outside the tunnel, including cleartext and decrypted HTTPS traffic) and matched them against widely used lists of advertising and tracking domains. Specifically, we used EasyList and its supplement EasyPrivacy [76], as well as the Disconnect list [93], which underlies the Disconnect browser extension and Firefox's tracking protection [94]. Because these three lists are generic and cross-platform, we also included the AdGuard mobile list [95], which is tailored for mobile apps. In total, 246 apps (over 80%) contacted 3,714 unique advertising and tracking URLs according to these lists.

Identifying Exfiltrated Data: Next, we examined the types of user and device attributes exfiltrated by VPN apps that are particularly relevant for tracking and device fingerprinting. As described in Section V-D, we extracted structured key-value pairs from network traffic and searched for known attributes associated with our test device. Among these pairs—summarized in Table IV—we focus on three identifiers: the Advertising ID, the device's IP address, and geographic coordinates. The Advertising ID, uniquely identifying on its own, was transmitted by 76 apps. Although resettable and deletable, it remains the primary persistent identifier available to third-party apps, since Android 10 and above restrict access to hardware identifiers such as the IMEI and serial number [96]–[98]. Meanwhile, 38 apps exfiltrated our device's IP address, and 1 app transmitted precise geographic coordinates. While the IP address and approximate geolocation can easily be inferred by the VPN server, its presence in application-layer key-value pairs reflects intentional collection and transmission by the app, establishing a lower bound on IP-based tracking.

Significance of Exfiltrated Data: Beyond these identifiers, we also observed widespread exfiltration of other device attributes: 210 apps transmitted our device model, 184 the Android API level, and 177 the OS version. While these values are not uniquely identifying on their own, they can be combined to create distinct device fingerprints [99]. Because our devices run Android 14, we did not observe persistent hardware identifiers such as the IMEI or serial number, which have been restricted since Android 10. However, the Advertising ID, though resettable, requires manual action by the user, and is therefore rarely changed in practice. As a result, most users remain vulnerable to cross-app and persistent tracking. Despite Android's efforts to limit access to hardware identifiers, device configurations inherently differ due to manufacturing decisions and user preferences [100], [101]. Collecting enough of these seemingly benign attributes allows advertisers and trackers to distinguish individual devices and thereby circumvent Android's privacy protections.

Category	Check	Apps	Unique Apps	Combined Installs
Insecure Cryptography	Weak Cipher	20	20 (18.5%)	40M+
	Msg Auth	9		
Weak Authentication	Uname/Passwd	22	96 (89%)	728M+
	Client Cert	74		
Deprecated Directives	Compression	12	12 (11%)	513M+
	Others	6		
Hardening Options	ID Verify	38	61 (56.4%)	601M+
	HMAC TLS	56		

TABLE V: OpenVPN configuration file analysis. Numbers represent the count of configuration files that fail each check.

Takeaway: We found that 76 VPN apps exfiltrate the Advertising ID, a persistent identifier available to third-party apps. Additionally, a large number of apps (over 200 in some categories) exfiltrated device attributes, such as model, OS version, and screen size. While these attributes are not uniquely identifying on their own, they can be combined to construct a complete device fingerprint.

E. VPN Configuration

We identified 108 applications containing at least one OpenVPN configuration file, with the number of such configuration files ranging from 1 and 186 per application. These files are spread across various directories in local storage, with the *cache* directory (94%) being the most common. However, an app will use only one configuration file per connection. When an app includes multiple configuration files, we analyzed the one showing the strongest evidence of alignment with best practices, providing an optimistic upper bound on the app's security posture.

Alarmingly, only one application complies with all evaluated security best practices. In contrast, 107 applications exhibit at least one potential security issue across one or more of the following categories: Insecure Cryptography (18.5%), Weak Authentication (89%), Deprecated Directives (11%), and Hardening Options (56.4%).

Insecure Cryptography: As depicted in Table V, we identified 20 (18.5%) applications with potentially weak tunnel encryption, collectively accounting for over 40 million installs. Among these, three (2.7%) applications explicitly set the `data-ciphers` directive to `none`. In the latest OpenVPN releases (2.5-2.6), this allows negotiation of no encryption in the data channel, resulting in an unencrypted tunnel [102].

Moreover, we find eight (7.4%) applications that set the `cipher` directive to `none`. In OpenVPN versions $\leq$ 2.4, setting it to `none` will disable tunnel encryption. However, in newer versions, this directive has been replaced by *data-ciphers*, and setting *cipher* alone no longer controls encryption behavior [80]. Disabling encryption enables a network attacker to intercept all tunnel traffic, and even if apps implements custom obfuscation, prior work has found such mechanisms to be weak and not recommended [34].

The remaining 20 applications did not explicitly disable encryption but instead rely on insecure or outdated ciphers. Specifically, we find 8 applications (7.4%) that do not define the `cipher` directive, in which case OpenVPN (versions $\leq$ 2.4) defaults to Blowfish. Blowfish has been shown to be vulnerable to birthday attacks (CVE-2016-6329) [103]. Furthermore, one (0.9%) application explicitly set the cipher to DES-EDE3-CBC, which is affected by a high severity vulnerability (CVE-2016-2183) [104].

Finally, OpenVPN employs Hash-based Message Authentication Codes (HMACs) to ensure the integrity of data packets [105]. This mechanism prevents tampered packets from being accepted by the client or server. As such, disabling HMAC allows machine-in-the-middle attackers to manipulate user traffic without detection. We identify 9 (8.3%) apps that disable HMAC, making them susceptible to such attacks.

Weak Authentication: As shown in Table V, we identified 96 applications (88.8%) that rely on only a single authentication mechanism (either a username/password or a client certificate). This weakens authentication and increases the risk of unauthorized access [82]. A majority (68.5% or 74) of the apps rely solely on client certificates, which grant access if the configuration file is leaked. The risk is amplified when a server allows multiple connections using the same certificate. Such scenarios can enable attackers to exploit vulnerable peers' services, as demonstrated by the "port shadow" vulnerability [16]. Additionally, 22 apps rely solely on username/password, which significantly increases the risk of unauthorized access if the credentials are compromised. In both cases, the absence of multi-factor authentication leaves users vulnerable to impersonation attacks. Together, these apps have over 728 million installs.

Deprecated Directives: As presented in Table V, we identified 12 (11%) apps, with over 513 million cumulative installs, that use at least one deprecated directive. The most common are `comp-lzo` (10 apps or 9%) and `compress` (2 apps or 1.8%), which were deprecated in OpenVPN 2.4 and 2.5, respectively. OpenVPN strongly recommends avoiding compression unless it is strictly necessary, due to susceptibility to the VORACLE attack [106]–[108]. Additionally, 4 apps (3.7%) include the deprecated `ns-cert-type` directive, and two (1.85%) use `tun-ipv6`. We argue that the presence of obsolete directives reflects poor security readiness, highlighting a lack of maintenance and disregard for security guidelines.

Hardening Practices: Based on the data in Table V, we find 61 (57.4%) apps that do not implement directives designed to prioritize security. In particular, 56 (51.8%) apps do not use `tls-auth`, `tls-crypt`, or `tls-crypt-v2`. These directives protect against DoS attacks, port scanning, buffer overflows, and SSL/TLS handshakes from unauthorized machines by implementing an additional HMAC signature over the control channel packets [81]. We then examined whether clients verify the server identification during authentication. We find 38 (35.1%) apps that do not perform this verification, making them vulnerable to server impersonation attacks. As such, an MITM adversary can force a user to connect to a malicious server by impersonating the legitimate one. To protect users against this threat, OpenVPN recommends including at least one of the following directives: `remote-cert-tls`, `verify-x509-name`, `peer-fingerprint`, or `verify-tls` [80].

However, we found no usage of the `peer-fingerprint` or `verify-tls` directives in any app in our dataset; 84 (77.7%) apps do not verify the server name using `verify-x509-name`, and 38 (34.2%) do not verify the server certificate using `remote-cert-tls`. Collectively, apps lacking these hardening options account for over 601 million installs.

Insecure Parameter Usage in Practice: While we found clear evidence of client configurations specifying weak parameters, we did not uncover whether they were used in practice. The client-side configurations may request specific parameters that the server-side configurations can override during a negotiation phase. That being said, the client-side configurations should not have such weak options to start with, and they should not be left to the server side to enforce stricter checks, especially when the client-side code is controlled by the same developers. The vulnerability to man-in-the-middle attacks that we uncovered is an example of a case where such weak configurations can have severe consequences for end users. We hope developers will eventually discontinue such practices and ship hardened client-side configurations.

Takeaway: Overall, only one app (out of 108 analyzed) follows all the criteria for following best practices. About 18.5% of apps expose users to insecure cryptography and 88.8% do not use recommended authentication methods. Furthermore, 11% of apps expose users to severe attacks by retaining deprecated directives, reflecting poor maintenance practices and limited security readiness, all while 56.4% of apps fail to implement security hardening directives.

VII. Discussion

Limitations of Play Store Disclosures: The Google Play Store attempts to communicate security and privacy information to users through multiple disclosure mechanisms: the "Contains ads" field, the developer-submitted Data Safety Section, and—specific to VPN apps—a recently introduced "Verified" badge indicating compliance with Google's security guidelines and MASA Level 2 validation [109]–[111]. While these disclosures are a step toward transparency, they are neither comprehensive nor necessarily accurate. Prior work has shown discrepancies in developer-reported data safety

claims [112], and the complexity and variability of these disclosures may leave users confused rather than informed. In practice, they risk functioning more as marketing signals than meaningful security guarantees. More work is needed to evaluate the effectiveness and interpretability of these labels for mobile VPN applications, using frameworks such as *MVPNalyzer*.

VPN Routing on Android: Correctly configuring VPN routing behavior on Android is non-trivial. VPN apps must coordinate multiple mechanisms to ensure that all user traffic is properly tunneled: `addDnsServer()` to specify DNS server used, `addRoute()` to add IP-based routing rules, `add(dis)AllowedApplication()` to specify application-based routing rules using `fwmark` and app `UID`, and `protect()` on sockets to prevent the tunnel connection itself going into the tunnel resulting in routing loop [21]. This complexity leaves significant room for misconfiguration, and we observed several apps in our dataset leaking DNS or user traffic as a result.

We discovered cases such as `com.v2cross.foxo`, which, instead of adding a default "catch-all" route to the tunnel interface, selectively routes large subnets through the tunnel, leaving gaps that leak traffic. This illustrates *MVPNalyzer*'s capacity for supporting deeper, targeted analyses of mobile VPNs. More broadly, our findings underscore the need for Android to offer more centralized or simplified routing interfaces for VPN apps, reducing the likelihood of developer error in critical privacy-preserving infrastructure.

Extensibility to iOS: While our current framework focuses on Android, the core analytical principles and methodologies we present are largely platform-agnostic and can be extended to iOS with appropriate adaptations. Because *MVPNalyzer* relies on root capabilities such as taking a packet capture, listing sockets opened by all applications, and setting LD_PRELOAD, the most straightforward implementation of *MVPNalyzer* on iOS would also require a jailbroken device. In this case, the primary adaptation required is to use the DYLD_INSERT_LIBRARIES environment variable to inject a custom library for TLS decryption, rather than LD_PRELOAD. However, iOS is less amenable to jailbreaking than Android, and may require older devices or operating system versions [113] to achieve it. If jailbreaking is not possible, it is still possible to intercept and decrypt TLS traffic using tools such as `mitmproxy`, albeit with certain limitations (e.g., certificate pinning).

End User Impact: *MVPNalyzer* reveals pervasive insecurity across VPN services. These issues translate into tangible risks for users, evidenced by multiple documented past incidents of exploitation and harm. For example, the Daixin ransomware group specifically targeted poorly protected VPN servers and used them to exfiltrate health record data, resulting in an advisory by the FBI and Cybersecurity and Infrastructure Security Agency (CISA) [114]. Similarly, a construction company suffered the exfiltration of more than 60,000 potentially sensitive documents, along with financial losses, after attackers exploited weak VPNs used for remote network access [115]. In another escalatory attack, adversaries exploited a VPN appliance used for remote administration of modems and routers on a satellite network, subsequently compromising thousands of consumer modems across Europe and disrupting communications in Ukraine and other countries [116]. Such real-world consequences validate the importance of *MVPNalyzer* in identifying and mitigating risks posed by insecure or otherwise untrustworthy VPNs.

Responsible Disclosure: We disclosed our findings to all VPN providers, except in cases where the issues are related more to transparency for end users than for exploitable vulnerabilities, including a lack of obfuscation (relevant to users in regions where using VPN apps can put users at risk) and the presence of tracking (relevant to users who want to avoid third-party tracking by using a VPN).

However, the most critical issue we identified is tunnel hijacking, which we disclosed on a priority basis. We received acknowledgement from 2 out of the 5 providers, both of which promised to transfer VPN configuration files over HTTPS. One of the providers responded to our disclosure as follows: "We've reviewed your findings and will prioritize implementing a fix to ensure OpenVPN configuration files are transmitted securely using HTTPS with proper certificate validation." In our disclosures, we provided all relevant details of the weaknesses identified, along with recommendations for remediation, and we have retained the corresponding packet captures.

Availability: We envision *MVPNalyzer* as a public, extensible tool that enables continuous auditing of security and privacy practices in the VPN ecosystem by a range of stakeholders. To this end, we are fully committed to open-sourcing the framework for use by the broader community. Our project page is available at https://censoredplanet.org/#/research/securing-pets. Furthermore, we list all tested apps for which *MVPNalyzer* uncovered issues in Table VI.

VIII. Conclusion

Mobile VPN applications occupy a uniquely privileged position, as they intercept all network traffic from a user's device, often containing sensitive personal data. To evaluate whether the trust is justified, we developed *MVPNalyzer*, the first extensible analysis framework for systematically investigating Android VPN applications. Our system addresses the unique challenges of mobile traffic collection and attribution, allowing for modular analysis across a broad range of behaviors. We applied *MVPNalyzer* to 281 operational free VPN apps from the Google Play Store and uncovered alarming issues in security and privacy practices—including the transmission of unencrypted traffic, traffic leakage, lack of obfuscation, device and user tracking, and insecure VPN tunnel configurations. Many of these apps found with issues have tens of millions of installs and appear among top Play Store search results.

IX. Ethical Considerations

Our study adheres to the principles of the Menlo Report [117]. No human subjects or user data were involved; all

experiments were conducted on devices controlled and owned by the authors. Identified vulnerabilities, such as unencrypted transmission of VPN configuration files and traffic leaks, affected only the VPN client apps on our test devices and were disclosed responsibly to affected providers. Further, no server-side systems were impacted.

X. Acknowledgement

The authors are grateful to the anonymous reviewers for their constructive feedback. This research was supported by the National Science Foundation under Grant Numbers CNS-2452883 and CNS-2452884.

References

[1] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, "I Know Why You Went to the Clinic: Risks and Realization of HTTPS Traffic Analysis," in *Privacy Enhancing Technologies*, 2014.

[2] R. Gonzalez, C. Soriente, and N. Laoutaris, "User profiling in the time of https," in *Proceedings of the 2016 Internet Measurement Conference*, 2016.

[3] T. Libert, "Privacy implications of health information seeking on the web," *Communications of the ACM*, 2015.

[4] Federal Trade Commission, "A Look At What ISPs Know About You: Examining the Privacy Practices of Six Major Internet Service Providers," Federal Trade Commission, Staff Report, 2021. [Online]. Available: https://www.ftc.gov/system/files/documents/reports/look-what-isps-know-about-you-examining-privacy-practices-six-major-internet-service-providers/p195402_isp_6b_staff_report.pdf

[5] D. Litvinova, "The cyber gulag: How Russia tracks, censors and controls its citizens," *AP News*. [Online]. Available: https://apnews.com/article/russia-crackdown-surveillance-censorship-war-ukraine-internet-dab3663774feb666d6d0025bcd082fba

[6] P. Mozur, A. Satariano, A. Krolik, and A. Aufrichtig, "'They Are Watching': Inside Russia's Vast Surveillance State," *The New York Times*. [Online]. Available: https://www.nytimes.com/interactive/2022/09/22/technology/russia-putin-surveillance-spying.html

[7] J. Zhang, K. Psounis, M. Haroon, and Z. Shafiq, "Harpo: Learning to subvert online behavioral advertising," in *Network and Distributed Systems Security Symposium 2022 (NDSS'22)*, 2021.

[8] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten, "Cookies That Give You Away: The Surveillance Implications of Web Tracking," in *Proceedings of the 24th International Conference on World Wide Web*, 2015.

[9] H. J. Smith, T. Dinev, and H. Xu, "Information privacy research: an interdisciplinary review," *MIS quarterly*, 2011.

[10] Namecheap, "Google Trends Reveals Surge in Demand for VPN," https://www.namecheap.com/blog/vpn-surge-in-demand/, 2020.

[11] J. Koebert and K. Lane, "2025 VPN Statistics and Consumer Report: 61% of Americans Remain Unprotected Online," https://allaboutcookies.org/vpn-usage-survey, 2025.

[12] R. Ramesh, A. Vyas, and R. Ensafi, ""All of them claim to be the best": Multi-perspective study of VPN users and VPN providers," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[13] M. Ikram, N. Vallina-Rodriguez, S. Seneviratne, M. A. Kaafar, and V. Paxson, "An analysis of the privacy and security risks of android vpn permission-enabled apps," in *Proceedings of the 2016 internet measurement conference*, 2016.

[14] M. T. Khan, J. DeBlasio, G. M. Voelker, A. C. Snoeren, C. Kanich, and N. Vallina-Rodriguez, "An empirical analysis of the commercial vpn ecosystem," in *Proceedings of the Internet Measurement Conference 2018*, 2018.

[15] A. Maghsoudlou, L. Vermeulen, I. Poese, and O. Gasser, "Characterizing the VPN Ecosystem in the Wild," in *Passive and Active Measurement: 24th International Conference, PAM 2023, Virtual Event, March 21–23, 2023, Proceedings*, 2023.

[16] B. Mixon-Baca, J. Knockel, D. Xue, T. Ayyagari, D. Kapur, R. Ensafi, and J. R. Crandall, "Attacking connection tracking frameworks as used by virtual private networks," *Proceedings on Privacy Enhancing Technologies*, 2024.

[17] N. Xue, Y. Malla, Z. Xia, C. Pöpper, and M. Vanhoef, "Bypassing Tunnels: Leaking VPN Client Traffic by Abusing Routing Tables," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[18] R. Ramesh, L. Evdokimov, D. Xue, and R. Ensafi, "VPNalyzer: Systematic Investigation of the VPN Ecosystem," in *Network and Distributed System Security*. The Internet Society, 2022.

[19] X. Mi, S. Tang, Z. Li, X. Liao, F. Qian, and X. Wang, "Your phone is my proxy: Detecting and understanding mobile proxy networks," in *Proceeding of ISOC Network and Distributed System Security Symposium (NDSS), 2021*, 2021.

[20] G. Cirlig, M. Elizen, L. Kaye, J. Marques, V. Parthasarathy, J. Santos, A. Sell, and I. Vasilyeva, "Satori threat intelligence alert: Proxylib and LumiApps transform mobile devices into proxy nodes," https://www.humansecurity.com/learn/blog/satori-threat-intelligence-alert-proxylib-and-lumiapps-transform-mobile-devices-into-proxy-nodes/, Mar 2025.

[21] "VPN | Connectivity | Android Developers," https://developer.android.com/develop/connectivity/vpn, 2025.

[22] K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. Mccoy, A. Nappa, V. Paxson, P. Pearce, N. Provos, and M. A. Rajab, "Ad Injection at Scale: Assessing Deceptive Advertisement Modifications," in *2015 IEEE Symposium on Security and Privacy*, 2015.

[23] B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson, "An analysis of China's "Great Cannon"," in *5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 15)*. USENIX Association, Aug. 2015.

[24] M. Namara, D. Wilkinson, K. Caine, and B. P. Knijnenburg, "Emotional and practical considerations towards the adoption and abandonment of vpns as a privacy-enhancing technology," *Proceedings on Privacy Enhancing Technologies*, 2020.

[25] A. Dutkowska-Zuk, A. Hounsel, A. Morrill, A. Xiong, M. Chetty, and N. Feamster, "How and why people use virtual private networks," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[26] C. Crail and L. Holznienkemper, "Top VPN Statistics And Trends," https://www.forbes.com/advisor/business/vpn-statistics/, 2025.

[27] A. Vigderman and G. Turner, "2025 VPN Usage Statistics," https://www.security.org/vpn/statistics/, 2025.

[28] "VpnService | Android Developers," https://developer.android.com/reference/android/net/VpnService, 2025.

[29] K. L. Wu, M. H. Hue, N. M. Poon, K. M. Leung, W. Y. Po, K. T. Wong, S. H. Hui, and S. Y. Chau, "Back to School: On the (In)Security of Academic VPNs," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[30] D. Xue, R. Ramesh, A. Jain, M. Kallitsis, J. A. Halderman, J. R. Crandall, and R. Ensafi, "OpenVPN is open to VPN fingerprinting," in *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 2022.

[31] J. Wilson, D. McLuskie, and E. Bayne, "Investigation into the security and privacy of iOS VPN applications," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020.

[32] D. Felsch, M. Grothe, J. Schwenk, A. Czubak, and M. Szymanek, "The Dangers of Key Reuse: Practical Attacks on IPsec IKE," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[33] V. C. Perta, M. Barbera, G. Tyson, H. Haddadi, A. Mei *et al.*, "A Glance through the VPN Looking Glass: IPv6 Leakage and DNS Hijacking in Commercial VPN clients," *Proceedings on Privacy Enhancing Technologies*, 2015.

[34] Q. Zhang, J. Li, Y. Zhang, H. Wang, and D. Gu, "Oh-Pwn-VPN! Security Analysis of OpenVPN-Based Android Apps," in *Cryptology and Network Security*, 2018.

[35] JoMingyu, "google-play-scraper." [Online]. Available: https://pypi.org/project/google-play-scraper/

[36] "Install System CA Certificate on Android Emulator," https://docs.mitmproxy.org/stable/howto-install-system-trusted-ca-android/.

[37] C. Brubake, "Changes to Trusted Certificate Authorities in Android Nougat," https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate.html, 2016.

[38] "Network security configuration | Security | Android Developers," https://developer.android.com/privacy-and-security/security-config#CertificatePinning, 2025.

[39] Frida Project, "Frida: A world-class dynamic instrumentation toolkit," https://frida.re/, 2025.

[40] @Q0120S, "Bypass SSL Pinning," https://codeshare.frida.re/@Q0120S/bypass-ssl-pinning/, 2025.

[41] "ptrace(2) - Linux manual page," https://man7.org/linux/man-pages/man2/ptrace.2.html, 2025.

[42] A. Druffel and K. Heid, "Davinci: Android app analysis beyond frida via dynamic system call instrumentation," in *Applied Cryptography and Network Security Workshops: ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19–22, 2020, Proceedings 18*, 2020.

[43] A. Tchana, L. Wapet, and Y.-D. Bromberg, "Odile: A scalable tracing tool for non-rooted and on-device Android phones," in *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*, 2022.

[44] "ld.so(8) - Linux manual page," https://man7.org/linux/man-pages/man8/ld.so.8.html, 2024.

[45] "SSL_new - OpenSSL Documentation," https://docs.openssl.org/master/man3/SSL_new/, 2025.

[46] A. Shuba and A. Markopoulou, "NoMoATS: Towards Automatic Detection of Mobile Tracking," *Proceedings on Privacy Enhancing Technologies*, 2020.

[47] "ss(8) - Linux manual page," https://man7.org/linux/man-pages/man8/ss.8.html, 2025.

[48] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019.

[49] "UI/Application Exerciser Monkey | Android Studio | Android Developers," https://developer.android.com/studio/test/other-testing-tools/monkey, 2023.

[50] "A research-oriented top sites ranking hardened against manipulation - Tranco," https://tranco-list.eu/, 2025.

[51] D. Milmo, "Russia blocks access to Facebook and Twitter," *The Guardian*. [Online]. Available: https://www.theguardian.com/world/2022/mar/04/russia-completely-blocks-access-to-facebook-and-twitter

[52] G. Peck, "Myanmar's embattled military government cracks down on free flow of news by blocking VPNs," *AP News*. [Online]. Available: https://apnews.com/article/myanmar-censorship-virtual-private-network-facebook-79fb4cc0c3c4317844d0c00b0be1d9d1

[53] A. McDonald, M. Bernhard, L. Valenta, B. VanderSloot, W. Scott, N. Sullivan, J. A. Halderman, and R. Ensafi, "403 Forbidden: A Global View of CDN Geoblocking," in *Proceedings of the Internet Measurement Conference 2018*, 2018.

[54] A. Ablove, S. Chandrashekaran, H. Le, R. S. Raman, R. Ramesh, H. Oppenheimer, and R. Ensafi, "Digital Discrimination of Users in Sanctioned States: The Case of the Cuba Embargo," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[55] S. Afroz, M. C. Tschantz, S. Sajid, S. A. Qazi, M. Javed, and V. Paxson, "Exploring Server-side Blocking of Regions," 2018. [Online]. Available: https://arxiv.org/abs/1805.11606

[56] "Data and file storage overview | App data and files | Android Developers," https://developer.android.com/training/data-storage, 2025.

[57] "File-based encryption | Android Open Source Project," https://source.android.com/docs/security/features/encryption/file-based, 2025.

[58] OpenVPN, "OpenVPN Connect - VPN For Your Operating System," https://openvpn.net/client/.

[59] A. Possemato and Y. Fratantonio, "Towards HTTPS Everywhere on Android: We Are Not There Yet," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[60] "The Zeek Network Security Monitor," https://zeek.org, 2025.

[61] "Spicy — Generating Robust Parsers for Protocols & File Formats — Spicy v1.13.0-dev.191," https://docs.zeek.org/projects/spicy/en/latest/, 2025.

[62] "JSON," https://www.json.org/json-en.html, 2025.

[63] schwabe, "GitHub - schwabe/ics-openvpn: OpenVPN for Android," https://github.com/schwabe/ics-openvpn/blob/85a015c610e79dedf18c8c89b0a9eb92901cf896/main/src/main/java/de/blinkt/openvpn/core/ConfigParser.java, 2025.

[64] J. L. Hall, M. D. Aaron, A. Andersdotter, B. Jones, N. Feamster, and M. Knodel, "A Survey of Worldwide Censorship Techniques," RFC 9505, 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9505

[65] P. Winter and S. Lindskog, "How the Great Firewall of China is Blocking Tor," in *2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI 12)*. USENIX Association, 2012.

[66] M. Wu, J. Sippe, D. Sivakumar, J. Burg, P. Anderson, X. Wang, K. Bock, A. Houmansadr, D. Levin, and E. Wustrow, "How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[67] D. Xue, B. Mixon-Baca, ValdikSS, A. Ablove, B. Kujath, J. R. Crandall, and R. Ensafi, "TSPU: Russia's decentralized censorship system," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022.

[68] ValdikSS, "Blocking VPN protocols on TSPU (08/05/2023 - xx.xx.202x)," https://ntc.party/t/vpn-05082023-xxxx202x/5124/7, 2023.

[69] R. Ensafi, P. Winter, A. Mueen, and J. Crandall, "Analyzing the Great Firewall of China Over Space and Time," *Proceedings on Privacy Enhancing Technologies*, 2015.

[70] P. Winter and S. Lindskog, "How the Great Firewall of China is Blocking Tor," in *2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI 12)*. USENIX Association, Aug. 2012.

[71] "nDPI – ntop," https://www.ntop.org/products/deep-packet-inspection/ndpi/, 2025.

[72] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, and G. Smaragdakis, "The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic," in *Proceedings of the ACM Internet Measurement Conference*, 2020.

[73] Oracle, "Chapter 6. names," https://docs.oracle.com/javase/specs/jls/se8/html/jls-6.html, 2025.

[74] P. S. List, "Public Suffix List," https://publicsuffix.org/, 2025.

[75] Z. Whittaker, "Privacy group accuses Hotspot Shield of snooping on web traffic," *ZDNET*. [Online]. Available: https://www.zdnet.com/article/privacy-group-accuses-hotspot-shield-of-snooping-on-web-traffic/

[76] "EasyList - Overview," https://easylist.to/, 2025.

[77] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 2016.

[78] "Advertising ID - Play Console Help," https://support.google.com/googleplay/android-developer/answer/6048248, 2025.

[79] J. Koetsier, "5 billion ad events show that fewer than 1% of Android users opt out of personalized ads," https://easylist.to/, 2023.

[80] OpenVPN, "Reference Manual For OpenVPN 2.6," https://openvpn.net/community-resources/reference-manual-for-openvpn-2-6/.

[81] ——, "Hardening OpenVPN Security," https://openvpn.net/community-resources/hardening-openvpn-security/.

[82] E. Crist, "Multi-Factor Authentication with OpenVPN | Community Edition," https://blog.openvpn.net/multi-factor-authentication-with-openvpn-community-edition/, 2020.

[83] OpenVPN, "Deprecated Options in OpenVPN," https://community.openvpn.net/openvpn/wiki/DeprecatedOptions.

[84] Forcepoint, "ThreatSeeker URL Categories and Description," https://help.forcepoint.com/fpone/migration/webtothreatseekerurl/guid-b8ac3928-a64c-4468-bf16-4a6d4932c2c7.html, 2025.

[85] "IP-API.com - Geolocation API - Documentation - JSON," https://ip-api.com/docs/api:json, 2025.

[86] "<application> | App Architecture | Android Developer," https://developer.android.com/guide/topics/manifest/application-element#usesCleartextTraffic, 2025.

[87] "Android 7.0 for Developers | Android Developers," https://developer.android.com/about/versions/nougat/android-7.0#network_security_config, 2024.

[88] "Network security configuration | Security | Android Developers," https://developer.android.com/privacy-and-security/security-config#CleartextTrafficPermitted, 2025.

[89] "Adoption & Usage Worldwide | Cloudflare Radar," https://radar.cloudflare.com/adoption-and-usage?dateRange=28d, 2025.

[90] G. Baker, "Another Door Closes: Authoritarians Expand Restrictions on Virtual Private Networks," https://freedomhouse.org/article/another-door-closes-authoritarians-expand-restrictions-virtual-private-networks, 2024.

[91] D. Xue, M. Kallitsis, A. Houmansadr, and R. Ensafi, "Fingerprinting Obfuscated Proxy Traffic with Encapsulated TLS Handshakes," in *33st USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.
[92] D. Xue, R. Stanley, P. Kumar, and R. Ensafi, "The Discriminative Power of Cross-layer RTTs in Fingerprinting Proxy Traffic," in *Network and Distributed System Security*. The Internet Society, 2025.
[93] "Disconnect - Tracker Protection lists," https://disconnect.me/trackerprotection, 2025.
[94] Z. Yu, S. Macbeth, K. Modi, and J. M. Pujol, "Tracking the Trackers," ser. WWW '16, 2016.
[95] "AdGuard filters | AdGuard Knowledge Base," https://adguard.com/kb/general/ad-filtering/adguard-filters, 2025.
[96] S. Son, D. Kim, and V. Shmatikov, "What Mobile Ads Know About Mobile Users," in *Network and Distributed System Security*. The Internet Society, 2016.
[97] S. Zimmeck, N. Aggarwal, Z. Liu, and K. Kollnig, "From Ad Identifiers to Global Privacy Control: The Status Quo and Future of Opting Out of Ad Tracking on Android," 2025. [Online]. Available: https://arxiv.org/abs/2407.14938
[98] "Privacy changes in Android 10," https://developer.android.com/about/versions/10/privacy/changes#non-resettable-device-ids, 2025.
[99] U. Iqbal, S. Englehardt, and Z. Shafiq, "Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.
[100] A. Kurtz, H. Gascon, T. Becker, K. Rieck, and F. Freiling, "Fingerprinting mobile devices using personalized configurations," in *Proceedings on Privacy Enhancing Technologies*, 2016.
[101] K. Heid and J. Heider, "Haven't we met before? - detecting device fingerprinting activity on android apps," in *Proceedings of the 2024 European Interdisciplinary Cybersecurity Conference*, 2024.
[102] OpenVPN, "Source code for ssl_ncp.c, lines 129–133," https://github.com/OpenVPN/openvpn/blob/master/src/openvpn/ssl_ncp.c#L129-L133, 2025.
[103] NIST, "CVE-2016-6329," https://nvd.nist.gov/vuln/detail/cve-2016-6329.
[104] ——, "CVE-2016-2183," https://nvd.nist.gov/vuln/detail/CVE-2016-2183.
[105] M. Bellare, R. Canetti, and H. Krawczyk, "Message authentication using hash functions: The HMAC construction," *RSA Laboratories' CryptoBytes*, 1996.
[106] OpenVPN, "VORACLE attack and OpenVPN," https://community.openvpn.net/openvpn/wiki/VORACLE.
[107] ——, "Security Advisory: The VORACLE attack vulnerability," https://openvpn.net/security-advisory/the-voracle-attack-vulnerability/.
[108] ——, "OpenVPN versus Compression," https://community.openvpn.net/openvpn/wiki/Compression.
[109] J. Zou and S. Lin, "Android Developers Blog: Helping users find trusted apps on Google Play," https://android-developers.googleblog.com/2025/01/helping-users-find-trusted-apps-on-google-play.html, 2025.
[110] "Improve your app's security | Security | Android Developers," https://developer.android.com/privacy-and-security/security-best-practices, 2025.
[111] "AL2 - Lab Eval | App Defense Alliance," https://appdefensealliance.dev/masa/masa-al2, 2025.
[112] I. Arkalakis, M. Diamantaris, S. Moustakas, S. Ioannidis, J. Polakis, and P. Ilia, "Abandon All Hope Ye Who Enter Here: A Dynamic, Longitudinal Investigation of Android's Data Safety Section," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.
[113] "Device Selection (iPhone)," https://ios.cfw.guide/get-started/select-iphone/.
[114] CISA, "CISA Advisory- Diaxin Team," https://www.cisa.gov/news-events/cybersecurity-advisories/aa22-294a, 2022.
[115] B. Response, "Construction company avoids paying ransom after compromised VPN leads to data exfiltration," https://www.coalitioninc.com/en-ca/case-studies/construction/client-requires-strict-protocol-after-data-exfiltration, 2024.
[116] V. Inc., "KA-SAT Network cyber attack overview," https://www.viasat.com/perspectives/corporate/2022/ka-sat-network-cyber-attack-overview/, 2022.
[117] E. Kenneally and D. Dittrich, "The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research," *Available at SSRN 2445102*, 2012.

APPENDIX

A. VPN Apps Grouped by Detected Issues

Issue Type	Apps
Unencrypted Traffic (61)	Safe VPN, Nox VPN, MahsaNG, VPN Connect, Veee, Arab VPN, 4ebur.net, NewNode VPN, Delight VPN, Fast VPN (com.express.vpn.master.save.browser.fast.proxy), Follow, Proxy Master, VPN (com.free.vpn.unlimited.hotpotshield.vpnmaster), VPN 3000, japan VPN, Instabridge, India VPN, Japan Vpn Pro, KUTO VPN, Kylo Vpn, LVCHA VPN, Star VPN, NotVPN (com.notvpn), NotVPN (com.notvpn2), Now VPN, Super VPN (com.optimizer.booster.fast.speedy.phone.smooth), Pro Gamer VPN, WhitehatVPN, Andromeda VPN, A1 VPN, Super VPN (com.supervpn.vpn.free.proxy), VPN SXP, Secure VPN (com.techsphere.securevpn.ui), VPN Fast, Ultraunique VPN, Upnet, FoxoVPN, Viva VPN, Free VPN (com.vpn.unlimited.free.private.access.fast.proxy.secure), wirevpn, YoyoVPN, VPN Free, NIGERIA VPN, VPN Freely, Fast VPN (free.vpn.filter.unblock.proxy.hotspot.fastvpn), Super VPN (free.vpn.proxy.unblock.svd), Bamboo VPN, VPN Proxy Master, JAPAN VPN, Fast VPN (free_vpn_master_unblock_super.proxy_secure_vpn_new_best_vpn), Node VPN, Gozal, Kiwi VPN (kiwivpn.connectip.ipchanger.unblocksites), Unicorn HTTPS, SkyVPN, BigMama, Free VPN (org.sanctuary.freeconnect), Fast VPN (org.sanctuary.quickconnect), PLADUK VPN, Nine Tail VPN, WORLD VPN APP
Leak (29)	Geo Tunnel, Java VPN, Noon VPN, AM TUNNEL LITE VPN, AM TUNNEL PRO, MahsaNG, GoFly VPN, Ostrich VPN, NewNode VPN, Cookie, Delight VPN, Phone Guardian, Raytunnel, RoboProxy, Kylo Vpn, LVCHA VPN, XY VPN, Take Off, Tesla Proxy Pro, Rosa VPN, Global VPN, Air Net VPN, FoxoVPN, V2net, Bolt VPN, Unicorn HTTPS, Free VPN (org.sanctuary.freeconnect), Siam VPN, Nine Tail VPN
Obfuscation (143)	AM TUNNEL LITE VPN, AM TUNNEL PRO, VPN Freedom, Proton VPN, MeowPlus VPN, Nox VPN, Free VPN (com.Free.Fast.Secure.VPN), Onion VPN, Fire VPN, Gaming Master VPN, NAMO VPN, Asia Vpn, Avira Security, Avira Phantom VPN, Bangladesh Vpn, Gaming Vpn, Germany Vpn, CandyLink VPN, 4ebur.net, NewNode VPN, ColorVPN, ExVPN VPN, Delight VPN, QUICK VPN PRO, Fast VPN (com.express.vpn.master.save.browser.fast.proxy), Geo VPN, VPN (com.fast.unblock.secure.vpn.master), VPN (com.free.vpn.unlimited.hotpotshield.vpnmaster), Super VPN (com.freevpn.unlimited.free.vpn), Raid VPN, Gamers VPN, VPN 3000, Asia VPN, VPN AZ, VPN BD, GPVPN, VPN IN, VPN MY, VPN PH, VPN SA, VPN TW, Sphere VPN, Gunnar VPN, japan VPN, Instabridge, My Private VPN, IP Safe VPN, Japan Vpn Pro, Japanese Browser Vpn, VPN (com.jystudio.vpn), Kylo Vpn, CheapVPN, #WiFi Hacker, NotVPN (com.notvpn), NotVPN (com.notvpn2), 1ClickVPN, SecureNet VPN, PandaVPN (com.pandavpn.androidproxy), PandaVPN (com.pandavpnfree.androidproxy), Pawxy, PotatoVPN, PrimeX VPN, PUKANG VPN, WhitehatVPN, Rapid VPN (com.rapidconn.android), VPN UK, SecureVPN, X-VPN, Shuttle VPN, Sigma VPN, VPN Unlimited, Tryme VPN, SOSO VPN, TikVPN, Ultra VPN Proxy, VPN (com.starnest.vpnandroid), VPN Online Shield, VPN SXP, Air Net VPN, Secure VPN (com.techsphere.securevpn.ui), Tower VPN, VPN Fast, Trusted VPN, BerdVPN, VPN BRAZIL, TOP VPN, UstreamingVPN, FoxoVPN, Viva VPN, Bolt VPN, VPN GO, Free VPN (com.vpn.unlimited.free.private.access.fast.proxy.secure), 5G Global Vpn Singapore, VPN99, VPN Hero, wirevpn, Encrypto Vpn, YoyoVPN, ZIVPN, Zoog VPN, FLY TUNNEL VPN, NIGERIA VPN, KOREA VPN, VPN Freely, Fast VPN (free.vpn.filter.unblock.proxy.hotspot.fastvpn), IX VPN, Super VPN (free.vpn.proxy.unblock.svd), INDONESIA VPN, Bamboo VPN, JustVPN, VPN Proxy Master, AUSTRALIA VPN, COLOMBIA VPN, EGYPT VPN, JAPAN VPN, MALAYSIA VPN, USA VPN (free.vpnusa.fast.unlimited.free.secure.turbo), VIETNAM VPN, Fast VPN (free_vpn_master_unblock_super.proxy_secure_vpn_new_best_vpn), VPN Mate, Kiwi VPN (kiwivpn.connectip.ipchanger.unblocksites), EC Tunnel LITE, SkyVPN, Gaming VPN (mobi.bgn.gamingvpn), HighSpeedVPN, 24CLAN VPN LITE, BigMama, Octohide VPN, Fast VPN (org.sanctuary.quickconnect), Speedtest, PLADUK VPN, Kiwi VPN (secure.unblock.unlimited.proxy.snap.hotspot.shield), Siam VPN, Nine Tail VPN, SharkVPN, EC Tunnel PRO, InHouse VPN, UFO FREE VPN, Australia VPN, China VPN, Korea VPN, Singapore VPN, WORLD VPN APP
Tracking (76)	Tesla Proxy Pro, i2VPN, VPN Hero, Geo Tunnel, VPN Proxy Master, wirevpn, VPN Unlimited, VPN World, SkyVPN, Bolt VPN, FoxoVPN, Gaming VPN (mobi.bgn.gamingvpn), VPN Master (gamingvpn.unlimtedvpn.vpnforgamers), Safe VPN, VPN Fast, Ultrasurf VPN, VPN (com.free.vpn.unlimited.hotpotshield.vpnmaster), Banana VPN, RoboProxy, 24CLAN VPN LITE, VPN (com.starnest.vpnandroid), VPN Free, GreenVPN Free, VPN GO, Take Off, TravelVPN, Super VPN (com.freevpn.unlimited.free.vpn), Geo VPN, Lantern, Planet VPN, OraVPN, EC Tunnel LITE, Kiwi VPN (kiwivpn.connectip.ipchanger.unblocksites), Lion VPN, Rapid VPN (com.rapid.vpn.unlimited.hotspot.secure), mangoflutter, Voice VPN, Japanese Browser Vpn, Secure VPN (com.fast.free.unblock.secure.vpn), #WiFi Hacker, 4ebur.net, Instabridge, ColorVPN, HighSpeedVPN, Fast VPN (com.express.vpn.master.save.browser.fast.proxy), SailfishVPN, Pronto VPN, SPL VPN, Cookie, Sphere VPN, OvpnSpider, Lightning Direct VPN, Shuttle VPN, Cat Proxy, UFO FREE VPN, VPN XLock Pro, Unique VPN, QuarkVPN, Secure VPN (com.techsphere.securevpn.ui), Thunder VPN, Phone Guardian, PotatoVPN, Encrypt VPN, TikVPN, Proxy Master, Fire VPN, Kiwi VPN (secure.unblock.unlimited.proxy.snap.hotspot.shield), Trusted VPN, Zoog VPN, VPN 3000, AppVPN, GoFly VPN, Rez Tunnel, Octohide VPN, Gamers VPN, VPN Private
OVPN Config (107)	Raid VPN, VPN Brazil, VPN99, Fast VPN (com.express.vpn.master.save.browser.fast.proxy), Kiwi VPN (secure.unblock.unlimited.proxy.snap.hotspot.shield), VPN 3000, ExVPN VPN, VPN Malaysia, VPN TW, SPL VPN, VPN China, Unique VPN, VPN JP, Gunnar VPN, VPN MY, VPN Hero, VPN Freedom, VPN.lat, Pronto VPN, GPVPN, japan VPN, UstreamingVPN, Gaming Master VPN, Bolt VPN, Geo VPN, VPN Canada, Speedtest, Hamo Tunnel vpn5, VPN Indonesia, SecureVPN, Zoog VPN, #WiFi Hacker, VPN India, SharkVPN, Free VPN (com.vpn.unlimited.free.private.access.fast.proxy.secure), MR Tunnel VPN, Cafe VPN, VPN USA, Voice VPN, VPN (com.fast.vpn.secure.unblock.proxy), COLOMBIA VPN, Robust Tunnel VPN, PrimeX VPN, Free VPN (com.Free.Fast.Secure.VPN), USA VPN (free.vpnusa.fast.unlimited.free.secure.turbo), ACE VPN, Asia VPN, FLY TUNNEL VPN, VPN BRAZIL, VPN GO, Gaming VPN (mobi.bgn.gamingvpn), 5G Global Vpn Singapore, VPN (com.jystudio.vpn), VPN AZ, OvpnSpider, Bangladesh Vpn, CandyLink VPN, Trusted VPN, VPN IN, InHouse VPN, INDONESIA VPN, VPN Unblock, 4X TUNNEL, A1 VPN, VPN Online Shield, Shuttle VPN, Gaming Vpn, Smart VPN, VPN (com.free.vpn.unlimited.hotpotshield.vpnmaster), Hide My IP, JustVPN, VPN Connect, Riseup VPN, Bamboo VPN, VPN (com.starnest.vpnandroid), Sigma VPN, KOREA VPN, NIGERIA VPN, Encrypto Vpn, QUICK VPN PRO, Instabridge, Japan Vpn Pro, VPN servers in Russia, U-VPN, Super VPN (com.freevpn.unlimited.free.vpn), VPN (com.fast.unblock.secure.vpn.master), VPN UK, AUSTRALIA VPN, EGYPT VPN, Andromeda VPN, VIETNAM VPN, VPN Unlimited, VPN PH, JAPAN VPN, Fire VPN, Germany Vpn, VPN Germany, Tryme VPN, SOSO VPN, Lite VPN, Steady Fast VPN, Ultra VPN Proxy, OD VPN, VPN Australia, MALAYSIA VPN, VPN SA, VPN BD

TABLE VI: Issues are grouped by category (defined in Section VI and illustrated in Figure 4). Apps are listed by name, with package names included only when multiple apps share the same name.