# Hidden Links: Analyzing Secret Families of VPN Apps

Benjamin Mixon-Baca
ASU/Breakpointing Bad
bmixonba@asu.edu

Jeffrey Knockel
Citizen Lab / Bowdoin College
jeff@citizenlab.ca

Jedidiah R. Crandall
Arizona State University
jedimaestro@asu.edu

## ABSTRACT

Ownership transparency in the VPN ecosystem allows users to make informed decisions about who they trust with their data. Researchers have recently begun investigating the relationships between seemingly distinct providers and who operates them, but such analysis is currently limited to a small sample of providers in the VPN ecosystem. One known family of providers, Innovative Connecting, Autumn Breeze, and Lemon Clove, has been previously scrutinized by two research efforts linking them to the People's Liberation Army.

In our work, we identify and analyze three families of VPN providers. Combined, their download counts on the Google Play Store exceed 700 million. Similar to previous research, we use information from business filings and Android APKs to link distinct providers together. However, we build upon past work by introducing new methods for revealing how VPN providers are connected, showing that they even share VPN servers' cryptographic credentials, including Shadowsocks passwords that are hard-coded into their APKs. Hard-coded Shadowsocks passwords allow an attacker to decrypt the traffic of these providers' clients, compromising the security claimed by these providers. Therefore, our analysis reveals that these apps share not only common ownership but a common set of security issues. As such, these apps' providers are not merely misleading their users about their ownership but about the extent of their security properties as well.

## 1 INTRODUCTION

To ensure the security and privacy of their network communications, users must know who develops, owns, and operates their Virtual Private Network (VPN) services because VPN operators may observe all communications transmitted by or to each client. VPN providers obfuscating their ownership interferes with users' ability to make informed decisions about who to trust with their data. Furthermore, any deception on the part of the providers undermines user trust and may suggest other problems such as with privacy and security practices of the provider in question.

In this paper, we analyze the privacy and security of VPN apps whose providers intentionally disguise their ownership. We use hidden relationships between supposedly distinct VPN providers as an indirect indicator of deceptive behavior. We then search for VPN-specific security issues in the deceptive providers' apps to uncover shared flaws. These shared flaws themselves serve as a signature by which to map relationships between providers.


Free and Open Communications on the Internet 2025(2), 18–27


VPN Pro was the first to widely report on hidden provider relationships [36]. They identified seemingly obfuscated relationships between Innovative Connecting, Autumn Breeze, and Lemon Clove. According to VPN Pro's research, these providers claimed to be based in Singapore but were owned by a Chinese national and subject to Chinese law. VPN Pro linked these providers together using information collected from WHOIS records and resources in the binary the providers distributed through the app stores. Recently, the Tech Transparency Project (TTP) uncovered links between these VPN providers (and multiple other providers) and Chinese computer security firm Qihoo 360, a company the United States government sanctioned on June 2020 for its connection to the People's Liberation Army (PLA) [28]. Their findings were based on legal documents such as mergers and acquisitions filings. These reports together reveal concerning details about ownership transparency in the VPN ecosystem and motivated us to ask the following questions: To what extent do VPN providers provide inaccurate or incomplete corporate-ownership information on the Google Play Store? Using analysis of legal filings, APK artifacts, and network communications, can seemingly distinct VPN apps be clustered into common-operator families? Which forensic signals are most reliable for that clustering? Within each detected family, to what extent are code bases, server infrastructure and cryptographic materials reused, and how does such reuse contribute to exposing users to widespread security issues?

To answer these questions, we created a list of seemingly distinct VPN providers and their respective applications. We collected information about the providers from their Google Play pages, APKs, websites, domain registration information, business filings, their social media presence, and social media posts on Google, GitHub, Reddit, Twitter/X, and LinkedIn. We used this information to look for VPN providers that appear to be obfuscating their ownership information and using deceptive business practices.

We identified three distinct families of VPN providers. We analyzed each for security issues, finding that, in addition to sharing other code similarity features, each family of apps also shared problematic security properties. For two of the families we identified hard-coded Shadowsocks passwords shared by their VPN clients. We also confirmed we could decrypt their traffic when these apps were using the Shadowsocks protocol. An attacker can use these credentials to decrypt all of the traffic for all of the clients of these providers. These families alone have over 700 million Google Play Store downloads. Therefore, these issues put the traffic of large numbers of users at risk.

The remainder of this paper is structured as follows. In § 2, we provide background on VPN technologies and their previously known weaknesses. § 3 covers the methodology we used for VPN selection and our analysis. § 4 details our findings, the implications of which we discuss in § 5 as well as potential remedies.

# 2 BACKGROUND

VPN apps implement a variety of tunneling protocols, which can operate on either the network layer or the application layer of the OSI model. In this section, we provide a brief summary of the security issues we look for in each app depending on its supported protocol(s). We then conclude the section by summarizing previous research identifying hidden connections between distinct VPN providers.

## 2.1 VPN Protocols

VPNs are not defined by a single protocol. Rather, a VPN is an abstract model of networking that many unique protocols satisfy. The protocols of interest in this work operate at either the network layer or the application layer, and this fact has security implications specific to it.

### 2.1.1 Network Layer VPNs.

Network Layer VPNs are truly VPNs in that they create a network layer connection between the VPN client and VPN server. Examples of network layer VPNs include OpenVPN [27], WireGuard [8], and IPsec [13]. When a client transmits a packet, the packet is routed through a tuntap (OpenVPN, IPsec) or wg (WireGuard) interface. The packet, including all network and transport layer headers, is encrypted, encapsulated into a payload of a new packet destined for a VPN server, and then routed to it. When the VPN server receives the packet, the VPN server program decapsulates and decrypts the packet, then routes it to the original destination. The VPN server is acting, essentially, as a network address translation (NAT) layer for clients. Encryption is typically intended to protect the payload from eavesdroppers between the VPN client and VPN server, and the NAT obfuscates the client's public IP address. These are the purported security features of VPNs. The detail that makes these protocols implement truly virtual private networks is the inclusion of network and transport layer headers and the reliance on the underlying operating system's network stack, routing tables, and firewall hooks to facilitate NAT and routing.

### 2.1.2 Network Layer VPN Weaknesses.

Several security issues have been discovered that are specific to modern network layer VPNs [23, 34, 39].

Blind In/on-path Attacks. Blind in/on-path attacks are a class of attack where an in/on-path attacker, while unable to observe the traffic inside of the tunnel (i.e., blind), can still tamper with the inner connection [34]. This attack comes in two flavors, client-side and server-side. The client-side attack permits an attacker to infer active connections between a VPN client and an external server to which the client is communicating through the VPN tunnel. The server-side attack permits an attacker between the VPN client and server to infer active connections or inject packets into the connection and take it over.

This client-side attack assumes the attacker is on the local network, such as a shared WiFi network at a conference venue. The attacker spoofs packets to the client, setting the destination IP to the IP of client's tuntap/wg interface's and source IP to the web server's IP. If the source and destination ports match and the TCP sequence number is in-window, the target responds with a challenge ACK through the VPN tunnel. The attacker will infer the ACK based on its size, implying an active connection despite the tunnel encryption. Source address validation (i.e., strict `rp_filter` in Linux) mitigates this threat, although mobile phones default to loose source address validation.

The server-side attack works even when the attacker is not on the local network, such as a router/ISP between the VPN server and client. Instead of spoofing a packet to the client directly the attacker spoofs packets to the VPN server. If the spoofed packet matches the connection, then it will reach the VPN client. The client will respond with a challenge ACK or process the attacker's packet as if it is a legitimate packet. There is no known mitigation for the server-side attack because the root cause is fundamental to VPN architecture (IP sharing).

Port Shadow Vulnerabilities. One limitation of blind in/on-path attacks is the on-path requirement because it is a privileged location. The port shadow vulnerability permits attackers without that access to achieve it by allowing the attacker to escalate from adjacent (i.e., the attacker and client connect to the same VPN server) to in-path [23]. The port shadow vulnerability affects VPN servers specifically and exploits the operating system's connection tracking framework to cause the VPN connection request to be routed to the attacker instead of being handled by the VPN server. In this work, however, we did not find any VPNs vulnerable to this attack, primarily due to the VPNs' use of network layer protocols that happen to not be vulnerable, such as IPsec, or application layer protocols, which are never vulnerable, such as Shadowsocks.

## 2.2 Application Layer VPNs

The only application layer VPN we consider in this work is Shadowsocks. Shadowsocks operates similarly to network layer VPNs in that a client sends a packet to an intermediary (the VPN/proxy server) that then forwards the packet to the client's intended destination. The major difference is that application layer VPNs do not operate on the layer 3 and layer 4 network layers the way network layer VPNs do.

### 2.2.1 Shadowsocks.

Shadowsocks is an application layer proxy designed explicitly to circumvent the Great Firewall of China [5]. Its use is offered by many VPN providers, including those featured in this study. Shadowsocks is a Fully Encrypted Protocol [10] (FEP) meaning it attempts to look completely random. Shadowsocks operates similarly to SOCKS5 proxies and adds symmetric encryption to implement its FEP. When a user visits a website in their browser, the application layer data is sent through the proxy using the SOCKS5 protocol.

Application layer VPNs must take additional steps to route all TCP, UDP, and ICMP traffic through the proxy. To facilitate this, libraries such as `REDSOCKS` [9] and `tun2socks` [21] intercept outgoing packets and route them to a SOCKS proxy that can then operate on the packets as required. This operation is similar to network layer VPNs in that the operating system's connection tracking framework intercepts and routes outgoing packets through the proxy. A service listens for and completes connections at layer 4 with the service sending the packets to the Shadowsocks proxy. The application layer data is then encapsulated using the SOCKS5 protocol and sent to the proxy server.

2.2.2 Shadowsocks Weaknesses. Providers who claim that a Shadowsocks-based VPN offers confidentiality or integrity are misleading their users because, although Shadowsocks uses (symmetric) encryption, it was not designed specifically to satisfy these or any other security properties [11]. Therefore, we are concerned with two weaknesses, decryption oracle attacks when using deprecated ciphers and hard-coded passwords. In addition to these problems, a Shadowsocks client that uses `REDSOCKS` or `tun2socks` is susceptible to the client-side blind in/on-path attack due to interaction with the connection tracking framework.

Decryption Oracles. Shadowsocks originally offered a "stream ciphers" suite for encryption. In 2020, Zhiniang Peng discovered that the packets lacked an integrity check, leading to a decryption oracle. An attacker could use this attack to decrypt client traffic when those ciphers are used [11], resulting in their deprecation. Among the ciphers in the suite, only the AEAD [6] one is still recommended. For example, if the VPN application contains a static configuration file, then this weakness can be identified by locating this file and identifying the cipher that the file specifies.

Hard-coded Passwords. Symmetric encryption uses the same key to encrypt and to decrypt. Therefore, after having extracted a hard-coded Shadowsocks password in a VPN app, from which a symmetric key is deterministically derived, a network eavesdropper can decrypt all traffic for all clients of the affected application. Identifying this weakness is the same as the decryption oracle — find the static information and record the key. An implementation facilitating private connections would use non-hard-coded keys.

## 2.3 Identifying Hidden VPN Provider Relationships, Ownership, and Deception

Before summarizing the previous research on uncovering obfuscated ownership and hidden relationships between providers, we define what we mean by "link", "provider linkage", and "deception".

2.3.1 Provider Linkage. By distinct provider, we mean the name a user sees when in the developer details of an application's Google Play page or in the terms of service or privacy policy documents served on the provider's website. "Link" means, given two distinct providers, e.g., Innovative Connecting and Lemon Clove, a single entity operates both services. The process of "linking" or "provider linkage" is using information, such as business records, information within APKs, or shared VPN servers, to deduce that a single entity operates both providers.

2.3.2 Deception. It is not uncommon for VPN providers to appear to be distinct but actually be operated by a single entity. For example, Kape Technologies operates multiple VPN services including ExpressVPN and Private Internet Access. Kape discloses this fact on their website so as to not be deceptive. When we speak of "deception", we mean that the relationship between providers requires uncovering the relationship by reviewing legal documents, domain registration information, decompiled APK files, and other sources beyond where an average user would look to find it, e.g., the provider/app's website or Google Play page.

2.3.3 Previous Research. The most convincing reports linking providers together and uncovering deceptive practices have leveraged business records. Research by Tech Transparency Project linked Innovative Connecting, Autumn Breeze, Lemon Clove, and several other providers to Qihoo 360 [28]. Their investigation linked these providers together and to Qihoo 360 by comparing multiple sources of data, including merges and acquisitions records. Qihoo 360 is based in mainland China, meaning that the operator is subject to the laws and regulations of that country which require censorship and surveillance of the Internet. Another concerning detail, revealed by VPN Pro and corroborated by TTP, was the apparent lengths the providers went to hide the fact that they were owned by a Chinese national. This detail is not apparent when viewing the Google Play Store pages of any of these VPNs, nor is it on any website. Instead, these apps state they are Singapore-based.

Much of the work linking providers depends on business filings, and, while researchers have identified and localized various types of hosts and services (e.g., proxies) [2, 4, 15, 37, 38, 40], similar methods have not been applied to linked VPN providers. Researchers have linked providers together by identifying potentially shared infrastructure [1, 16, 29], identifying similarities between provider VPN clients, or performing text comparisons with privacy policies [36], but definitive proof of such relationships typically remains elusive.

In our own study, we realized the inherent limitations of using privacy policy comparisons or code comparisons to infer links. A developer taking shortcuts might copy and paste the privacy policy of a competitor. Two providers might employ the same software developer without knowing. Therefore, while such comparison techniques are a good start, shared infrastructure is more compelling as evidence. In our study, we improve upon the previous understanding of relationships between VPN providers by using the cryptographic credentials from one provider to establish a tunnel with the servers of a different provider. These shared, hard-coded credentials will also prove to be a shared security defect.

# 3 METHODS

Our methodology consists of three steps: collecting developer and app names for 100 VPN providers, filtering this list down to 50 providers by excluding providers who were based in the USA, and identifying the subset of VPNs that appear deceptive.

## 3.1 VPN Selection

We selected the 100 most downloaded VPN apps according to SensorTower [31] and AppMagic [17], who both provide aggregated metadata about mobile applications for market research. We combined the datasets by comparing the download counts for each dataset and selecting the larger value reported when they differed. The initial list was filtered down to 50 by including only apps whose providers were based outside of the United States. We prioritized Singapore-based providers because of the previous research that observed deceptive providers incorporating in Singapore [36]. We then collected information about the providers from their Google Play page, website, domain registration information, GitHub, GitLab, Gitee, and the providers social media pages. We identified 13 potentially related and deceptive providers for security analysis and to search for more definitive evidence of deception via infrastructure connections (see Table 1 for their aggregated download counts).

Table 1: The number of Google Play Store downloads for each app analyzed (see § 4 for family identification).

| Family | Provider Name | VPN Name | # Downloads |
|---|---|---|---|
| A | Innovative Connecting | Turbo VPN | 100,000,000+ |
| | Innovative Connecting | Turbo VPN Lite | 50,000,000+ |
| | Innovative Connecting | VPN Monster | 10,000,000+ |
| | Lemon Clove | VPN Proxy Master | 100,000,000+ |
| | Lemon Clove | VPN Proxy Master - Lite | 10,000,000+ |
| | Autumn Breeze | Snap VPN | 50,000,000+ |
| | Autumn Breeze | Robot VPN | 10,000,000+ |
| | Autumn Breeze | SuperNet VPN | 1,000,000+ |
| B | MATRIX MOBILE PTE LTD | Global VPN | 10,000,000+ |
| | MATRIX MOBILE PTE LTD | XY VPN | 100,000,000+ |
| | Super Z VPN (Privacy & Proxy) | Super Z VPN | 10,000,000+ |
| | The Tool Tech | Touch VPN-Stable & Secure | 50,000,000+ |
| | Fruit Security Studios | VPN ProMaster-Secure your net | 50,000,000+ |
| | Fruit Security Studios | 3X VPN - Smooth Browsing | 100,000,000+ |
| | WILDLOOK TECH PTE. LTD. | VPN Inf | 10,000,000+ |
| | WILDLOOK TECH PTE. LTD. | Melon VPN - Secure Proxy VPN | 50,000,000+ |
| C | FreeConnectedLimited | X-VPN | 50,000,000+ |
| | Fast Potato ptd ltd | Fast Potato VPN | 10,000,000+ |
| Other | Miczon LLC | Tetra VPN | 1,000,000+ |
| | Super VPN Inc | VPN - Super Unlimited Proxy | 100,000,000+ |
| | Secure Signal Inc | Secure VPN Safer Internet | 100,000,000+ |
| Total | 13 Providers | 21 Apps | 972,000,000+ |

In the remainder of this section we summarize our collection and analysis methodology for these sources below. Overall we found that analysis of providers' websites, business filings, and apps' source code were the most useful for linking providers and identifying deception.

3.1.1 Websites. For each selected application, we collected the developer name, address/location, and website and privacy policy link from the Google Play Store and downloaded the APK onto our analysis device. We then reviewed the website, privacy policy, terms of service, and APK code and built a search terms list to search Google, Twitter/X, Reddit, LinkedIn, Telegram, Facebook/Instagram, YouTube, GitHub, GitLab, and Gitee. When we decompressed the APKs, we noted the asset and shared library names, and searched the decompiled code for email addresses, URLs, and similar terms that we added to our search term list. A sample of the list of search terms that we used is provided in Table 2 in Appendix A.

3.1.2 Business Filings. We used the provider name identified on the Google Play Store and mentioned in each provider's privacy policy and terms of service documents as search terms in OpenCorporates to find business records for each provider. OpenCorporates is a London-based NGO that aggregates business records, like tax and copyright records, of companies around the world to promote transparency [26]. An example search result is provided in Figure 1.

We found 10 providers with records in OpenCorporates and 3 that did not. We then reviewed available business records, such as copyright filings. We cross-referenced addresses in these documents with addresses listed on websites and Google Play pages and noted inconsistency. We found 10 providers had addresses listed in China but allegedly operating out of Singapore.

3.1.3 Social Media. We searched for and noted VPN providers and apps that had social media profiles on Facebook/Instagram, Twitter/X, Reddit, Telegram, Discord, and YouTube. We did not find any information useful for provider linkage or identifying deception for our study. We also searched GitHub, GitLab, and Gitee for any repositories the VPN providers might maintain, but we failed to locate any of such repositories.

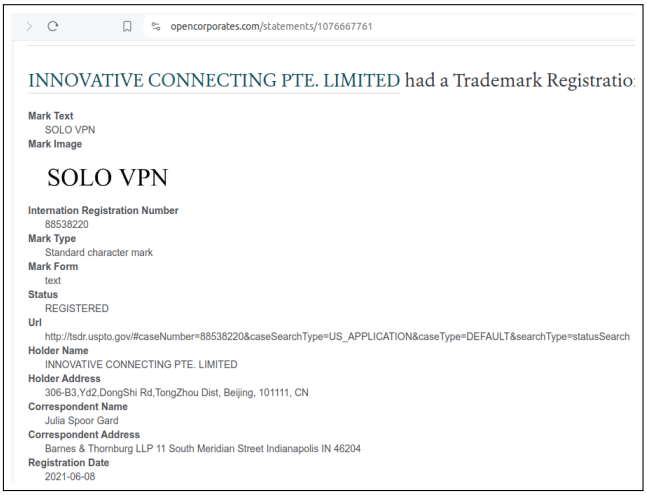


Figure 1: Innovative Connecting Pte LTD linked to Beijing through copyright filings with USPTO.

3.1.4 DNS Records. We collected DNS records using `dig` and `WHOIS`. In all cases, the VPN providers either redacted all information from their `WHOIS` records or used a domain registrar, such as Domain Protection Services or Domain by Proxy, to anonymize this information. Thus, domain-related information was largely not useful for making an association between distinct VPN providers.

3.1.5 APKs. We downloaded the APK for each application onto our analysis device (detailed in § 3.2). We used the `strings` command to search for specific file paths and email addresses. We found shared libraries were the most productive source linking VPN providers together. For example, pictured in Figure 2, VPN applications from Innovative Connecting, Autumn Breeze, and Lemon Clove each had strings in `libopvpnutils.so` explicitly linking the VPN applications for each provider together.

## 3.2 Security Analysis

Our analysis goals are twofold, to identify VPN-specific threats and to uncover deception via provider linkage. All applications were loaded onto a rooted Google Pixel 7a device for testing. We used a laptop as a WiFi hotspot and connected the mobile device to the hotspot to collect packet captures using the laptop.

3.2.1 Finding VPN-specific Security Issues . We searched for the VPN-specific security issues outlined in § 2. During dynamic analysis, we recorded packet captures to identify VPN server IP addresses and connections to APIs to which the VPN app made calls. We also configured the device to proxy traffic through mitmproxy [7] to search for API connections if the app used TLS outside the VPN tunnel for such connections.

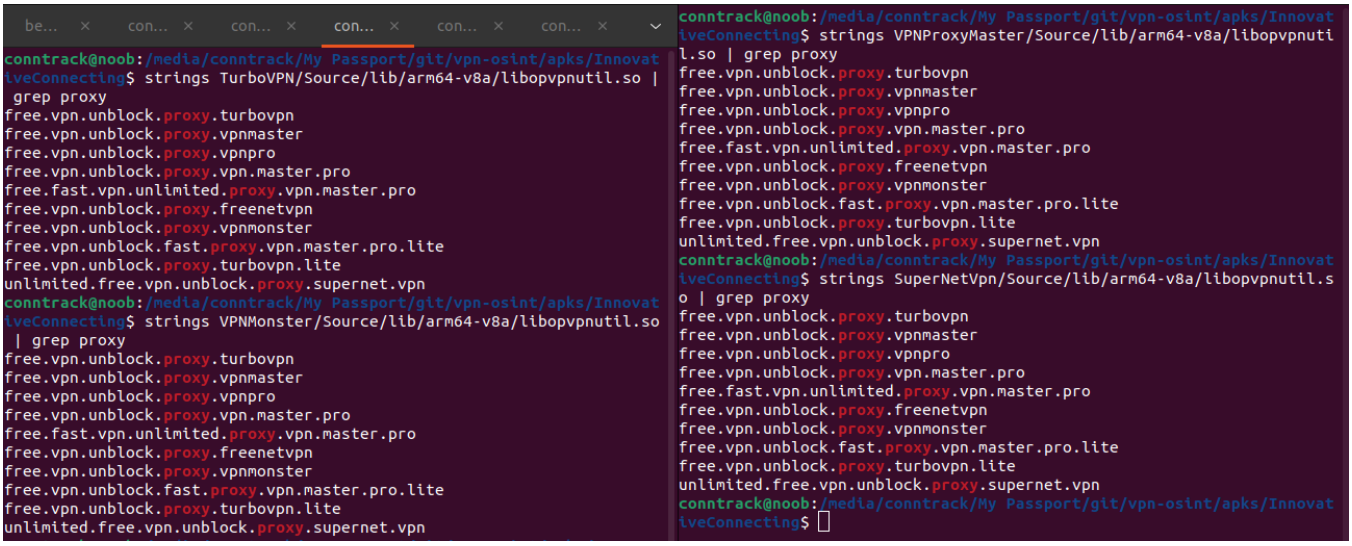


Figure 2: `strings` output showing references to the multiple VPN applications from supposedly distinct VPN providers.

Static Analysis. For our static analysis, we compared the filenames and SHA256 hashes of files in the `assets` and `lib/arm64` directories. We used the GNU `strings` command to extract strings from shared libraries. We then performed consistency checks of file types by comparing file extensions (e.g., `.png`) with the file type reported by the `file` command and recorded instances when `file` reported `data` file types and the file extension is either missing or of types for images (e.g., `png`, `jpg`, etc.) and common text files formats (e.g., `.txt`, `json`). We used jadx [32] to decompile `.dex` files and Ghidra [25] to decompile native code libraries.

Dynamic Analysis. For dynamic analysis, we focused on identifying credentials like WireGuard configuration files, OpenVPN and IPsec certificates, or Shadowsocks passwords. We used Frida [30] to trace the execution of specific libraries and functions we identified during static analysis and fridump [24] to extract objects like credentials from process memory. We noted instances when the app appears to use anti-reverse-engineering countermeasures. Finally, we used mitmproxy to view the APIs VPN apps call during execution.

## 4 RESULTS

We identified three families, which we call Families A, B, and C, containing three, five, and two providers, respectively. Three of the VPN providers that we analyzed did not appear to have any similarities with the other VPNs, which we assign to an "other" catch-all group.

In the remainder of this section, for each family identified, we summarize the signatures that led us to cluster the family's providers together. For each family, we also summarize the results of our security analysis. For Families A and B, we performed a full security analysis, and, for Family C, we performed a partial analysis that did not consist of analyzing the apps' cryptography. For these families we describe each of the security issues we identified, but leave analysis of their cryptography to future work.

### 4.1 Family A

Family A consists of the providers Innovative Connecting, Autumn Breeze, and Lemon Clove, who collectively operate eight VPN applications. Other researchers [28, 36] have linked more apps to Lemon Seed, the holding company, but we could only confirm the shared infrastructure for these three providers. Each application contained nearly identical decompiled Java code, shared libraries, and assets. The overt copy-and-pasting suggests a single developer implemented all of the apps and reused significant portions of code. While it is plausible that someone could have copied their APK and added their own code, given the signatures, weaknesses, and shared infrastructure, this seems unlikely.

#### 4.1.1 Signatures.

We found consistent patterns across the apps' protocols, software implementations, and code obfuscation methods.

Protocols. Each app supported at least the IPsec and Shadowsocks protocols. IPsec was implemented in part by `libcharon.so`, `libstrongswan.so`, and `libipsec.so`, whereas Shadowsocks was implemented in part by `libsslocal.so`, `libredsocks.so`, and `libtun2socks.so`. In the case of Robot VPN, Snap VPN, SuperNet VPN, TurboVPN, and VPN Monster, the settings menu contained a radio button for OpenVPN, but it was not selectable at the time of analysis.

Code Signatures. We found significant code overlap both at the Java and native level, and within the `assets`. Each app contains four characteristic files: `aaa_new.png`, `cert.pem`, `proxy.builtin`, and `server_offline.ser`.

Defense Mechanisms. The code appeared to be designed to deceive analysts or automated security checks. For example, in some apps (e.g., VPN Monster) the file `aaa_new.png` contains bytes that are read into a helper function that decrypts it and uses it for IPsec configuration. The key used to decrypt this file is also built dynamically in native code. These characteristics were present in all of the

apps in this family. We successfully extracted the IPsec and Shadowsocks configuration parameters using Frida during execution and from memory using fridump.

4.1.2 Security. We confirmed multiple security deficits shared by this family of VPN providers, which we summarize below.

Blind In/on-path attacks. All eight applications were susceptible to client-side blind in/on-path attacks. We found the underlying issue to be in `libredsocks.so`, which uses the operating system's firewall (Netfilter for Android) to divert all packets through the Shadowsocks tunnel. This design enables a client-side attack that lets an adversary interfere with active connections.

Location Information. We configured the devices to proxy their communication through mitmproxy. We found that the VPNs call APIs to, for example, download VPN config files, public keys, and upload user telemetry. Even when the VPN did not request the location permission, it requested the zip code of the user's public IP from `ip-api.com`, which it subsequently uploaded to a Firebase endpoint. The apps' privacy policies all claim they do not collect user addresses (personal or business), yet we observed them to do so.

Weak Encryption. The Shadowsocks configuration files use `rc4-md5` for encryption and specifies "plain" for the obfuscation technique. `rc4-md5` is one of several deprecated ciphers supported by Shadowsocks. The Shadowsocks implementation does not discard the first 512 bytes of keystream prior to encryption, which is recommended [22, 35], making a confirmation attack possible. Fortunately, the initial MD5 hashing, use of a longer IV, and construction of key material protect against key-recovery attacks such as FMS [12], WPA style attacks [3, 33, 35].

Hard-coded Keys. The apps contain a hard-coded password for Shadowsocks configuration in the file `assets/server_offline.ser`. This file is encrypted using `AES-192-ECB`. When the app first connects to a Shadowsocks server, it attempts to download a config file from a remote server, although we never observed the app successfully download a remote config. If or when that fails, the app then calls the native function, `NativeUtils.getLocalCipherKey`, implemented in the library `libopvpnutil.so`. This function deterministically builds the secret key used to decrypt the configuration file. The key built depends on which VPN app is calling the function (e.g., VPN Monster and TurboVPN build the same key but VPN Monster and Snap VPN do not). We confirmed that `server_offline.ser` is the same file across different geographic locations and devices. Figure 3 depicts the Shadowsocks password shared by every user who has downloaded VPN Monster, TurboVPN, or TurboVPN Lite, and Figure 4 demonstrates how knowledge of such a password can decrypt their traffic. Another consequence of the hard-coded password is that anyone who downloads the app can "freeload" off of the VPN service by establishing a Shadowsocks tunnel from a laptop using the extracted parameters.

We used `libopvpnutil.so` as a search term in Google and GitHub. We found some instances of files with this name. However, the newly discovered instances lacked a feature present in the apps that are part of our study. The instances of `libopvpnutil.so` found in the apps in our study reference the various VPN apps distributed by these providers. This was one of the ways we connected them together. The versions on GitHub have no such signature. It appears whoever developed these apps used the version on GitHub as a template for integrating custom native code.

Server Enumeration. Another consequence of the hard-coded password is that an attacker can enumerate additional VPN servers operated by the same entity. We tested this by selecting IP addresses in the same /24 as a VPN server confirming that we could connect to other servers on that /24. We use this capability to confirm that supposedly different VPN providers share server infrastructure.

Cross-provider Infrastructure Sharing. We used the above "freeload" capability to test whether the providers share the same servers. To do this, we established tunnels to VPN servers using each app and recorded the VPN server IP to which we connected. We then connected to each of them using the same hard-coded credentials to confirm that Innovative Connecting, Autumn Breeze, and Lemon Clove share infrastructure.

## 4.2 Family B

Family B consists of the providers MATRIX MOBILE PTE LTD, ForeRaya Technology Limited, WILDLOOK TECH PTE LTD, Hong Kong Silence Technology Limited, and Yolo Mobile Technology Limited. MATRIX MOBILE PTE LTD develops Global VPN and XY VPN and ForeRaya Technology Limited develops Super Z VPN. Notably, both these apps' privacy policies explicitly reference Innovative Connecting.

4.2.1 Signatures. The patterns we saw across these apps consisted of the particular protocols they support and their implementations, identifying strings we found in the code, a particular form of code obfuscation that we observed, and the sharing of VPN server IP addresses.

Supported Protocols. All five apps appear to support only Shadowsocks, facilitated by `libsslocal.so`. They both also use `libredsocks.so` and `libtun2socks.so` to build the proxy.

Code Signatures. There is a mixture of similarities and differences with these apps. While XY VPN and Super Z VPN have similar code structures, they are different from Global VPN. Global VPN and Super Z VPN reach out to the same Shadowsocks service at `149.28.197.166:443` but XY VPN does not. None of these apps are similar to Family A. For example, they use `libsslocal.so` whereas Family A uses `libss-local.so`. All of the apps contain the file `libcore.so` which contains explicit references to the APK files for these eight VPN clients.

Defense Mechanisms. XY VPN and Super Z VPN are obfuscated at the Java level by concatenating real words together for package and function names to appear legitimate and even informative, e.g., `RingAdaptorDecrypted`, but do not reflect functionality.

4.2.2 Security. We confirmed multiple security deficits shared by this family of VPN providers, which we summarize below. Blind In/on-path attacks These applications are susceptible to connection inference attacks using client-side blind in/on-path attacks because of the `libredsocks.so` and `libtun2socks.so` dependencies. Weak Encryption The VPN clients contain obfuscated passwords in the shared library `libcore.so`. Similarly to Family A, the app decrypts a Shadowsocks config based on which APK is executing and uses a hard-coded password to connect to Shadowsocks servers. Each app offers 33 built-in servers. Each server has 19 open ports,

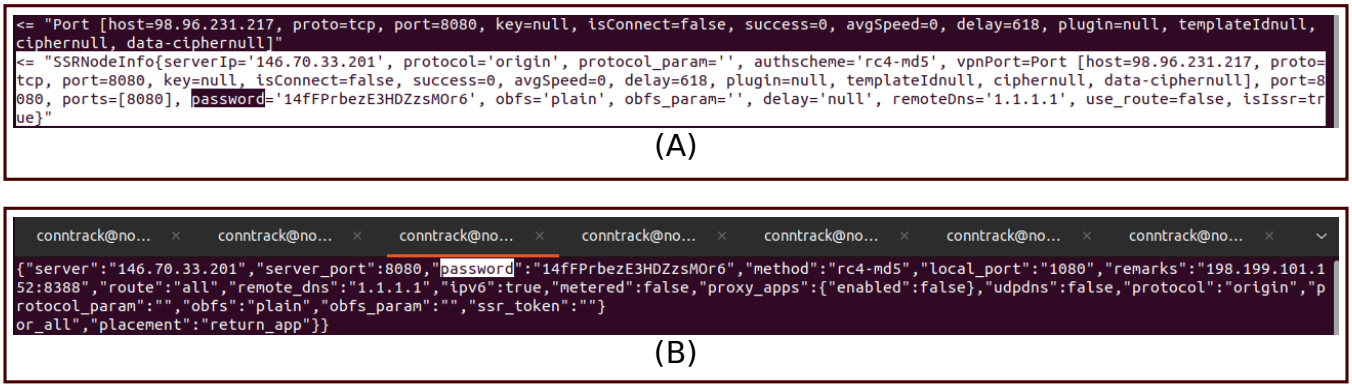


Figure 3: VPNMonster: Shadowsocks passwords are present in both the frida-trace (top) taken during execution and the memory dump for the VPNMonster process (bottom).





Figure 4: Hard-coded keys in VPN Proxy Master enable a network eavesdropper decrypt traffic. Upper: the original, encrypted traffic; lower, encircled in red: decrypted traffic.

each running a Shadowsocks process. Depending on the app's package name, one of these ports and one of 14 hard-coded passwords are selected.

Server Enumeration. This family of providers appears to use a smaller set of servers compared to Family A, and we were unable to enumerate servers beyond those specified in the apps' config files. All of Family B's VPN servers are hosted by a single company, GlobalTeleHost Corp. (gthost).

Cross-provider Infrastructure Sharing. We collected IP addresses for each VPN and were able to confirm that the VPN apps connected to the same set of IP addresses, though as we note earlier they do so on different ports.

## 4.3 Family C

Family C consists of the providers Fast Potato Pte. Ltd and Free Connected Limited, who distribute Fast Potato VPN and X-VPN, respectively. Fast Potato has no business filings according to OpenCorporates. Free Connected Limited is based in Hong Kong and has business records.

4.3.1 Signatures. We found multiple patterns across these providers, including a shared, proprietary protocol implementation, identifying file names and strings, and in their code organization.

Protocols. Based on their packet captures, the apps appear to use a custom tunneling protocol and make connections to servers on port 53 (DNS). None of the content in the application layer payloads was valid DNS and was instead obfuscated. This is possibly to bypass firewall rules permitting DNS on port 53.

Code Signatures. The decompiled code of these two VPNs is structurally and functionally similar. Both contain the same hard-coded, pre-shared key value in the variable `psk`. Both import the same shared library, `libxjp6xdkbew.so`. This library is the source of a second `psk` value. Both load the library in the same class named `wcyybbcujkCs`. Both utilize identically named resource files, `assets/jsd5xcyjr5w587dk5usn` though their exact contents vary. Neither the name of the shared library nor the asset file yield results when searching VirusTotal, Google, Brave, GitHub, Reddit, Twitter, or other sources, indicating that these names are specific to these VPN clients.

Defense Mechanisms. Both apps implement the same obfuscation and anti-reverse engineering countermeasures. At the Java level, there is minimal code outside of imports for advertising and analytics. Most of the functionality is implemented in native code. At the native code level, each contains the single, shared library mentioned above. The tunnel functionality is contained within this library along with anti-reversing countermeasures.

4.3.2 Security. Both applications are susceptible to connection inference attacks using the client-side blind in/on-path attacks. We localized the issue to the applications' dependency on Netfilter to

reroute packets through the tun interface. The cause for these apps is their libredsocks.so dependency. We tested each application to confirm by using the method outlined in § 2.

### 4.4 Other VPNs

The "Other" group of providers consists of VPN Super Inc., Miczon LLC, Secure Signal Inc. which distribute VPN Super Inc, TetraVPN, and Secure VPN - Safer Internet, respectively. They appear to have no links to other VPNs. We briefly summarize their most notable characteristics below.

VPN Super Inc. The application does not appear to employ much obfuscation beyond ProGuard, and there are no anti-reverse engineering counter-measures present. The application supports OpenVPN and IPsec (IKEv2). We tested both and were able to extract memory dumps confirming there appear to be public keys and configuration parameters consistent with these protocols.

TetraVPN. TetraVPN has no anti-reversing counter-measures except for ProGuard obfuscation and contains no assets. Only the WireGuard protocol is supported and it contains configuration files, which are shared by anyone who downloads the app, at res/raw/uk_client.conf and res/raw/japan_client.conf.

Secure VPN - Safer Internet. This application uses ProGuard obfuscation. The core functionality is implemented in native code in libchannel.so, which is obfuscated.

## 5 DISCUSSION

We identified three classes of problematic security and privacy issues with varying impacts on users. The undisclosed location collection issue is a major violation of user trust and privacy given the provider explicitly stated they did not collect such information. The client-side blind in/on-path attacks allow an attacker to infer with whom a VPN client is communicating. Most critically, on many of the VPNs we analyzed, a network eavesdropper between the VPN client and VPN server can use the hard-coded Shadowsocks password to decrypt all communications for all clients using the apps. These weaknesses nullify the privacy and security guarantees the providers claim to offer. These issues are even more concerning when accounting for the fact that the providers appear to be owned and operated by a Chinese company and have gone to great lengths to hide this fact from their 700+ million combined user bases.

The issues we identified affect users, providers, and app stores. At a minimum, VPN users who value privacy should avoid using Shadowsocks, including the apps from these developers, as Shadowsocks was not designed to facilitate privacy, merely censorship circumvention [11]. App store operators like Google face major challenges identifying and verifying ownership of apps on the Play Store, as well as ensuring Play Store apps are secure. Ownership identity verification and app security auditing is currently labor intensive and would require sophisticated, automated tools to achieve at scale. Google currently offers a security audit badge for VPN apps. Whether a similar badge for verified identity makes sense is debatable because there are valid reasons why a VPN provider might not want to reveal that information as it could expose them to legal or digital attack from a country or entity that opposes VPNs. Finally, VPN providers should avoid offering Shadowsocks to users or carefully explain the risks. The Shadowsocks protocol has no built-in asymmetric cryptography and requires the insecure use of hard-coded passwords, from which symmetric keys are deterministically derived, or for VPN providers to devise and implement a system for the secure distribution of these passwords. Prior work has found that home-rolled cryptographic systems commonly contain major flaws [18–20]. Thus, if not devised and maintained by experts, such a password distribution system would be liable for the introduction of additional security issues, and it may increase one's vulnerability to network censorship if not carefully implemented.

We have several unresolved questions based on our findings. First, it is unclear why a single entity would split their user base across multiple distinct providers and multiple VPN apps the way these providers appear to have done. One thought is that doing so isolates reputational damage to a single provider. Second, it is unclear why they did not also segment their code, credentials, or server infrastructure considering they went through the trouble of segmenting the legal side of their business. Simple explanations include cost, ease of execution, and/or management. It costs less to pay one developer to make one app and repackage it than to pay for multiple different apps. It is also easier and cheaper to have one app use the same credentials and infrastructure as one only needs to pay a single team to manage that infrastructure. Third, it was counterintuitive to find deprecated ciphers and hard-coded passwords in these apps, given that they are security-sensitive apps and many of their providers are owned by Qihoo 360, a major Chinese cybersecurity firm.

For the time being, users should, if possible, avoid using Shadowsocks from these particular providers unless and until they implement mitigations, if not Shadowsocks operators writ large. Hard-coded Shadowsocks passwords counterproductively increase the exposure of users' communications to eavesdroppers compared with using no VPN at all. Shadowsocks was designed to be censorship resistant, not private. Therefore, VPN providers need to be upfront with their users about the risks if they use Shadowsocks with only symmetric cryptography.

App stores like the Play Store are in a challenging position given the scalability limitations around vetting developers and identifying software with misleading security properties in their store. Google offers a security audit badge for VPN apps, but making such a badge mandatory for VPN apps and offering an identity verification badge for developers who go through an identity verification process might provide users additional information and protection. Google is potentially exposing its brand to reputational damage by hosting and profiting from deceptive and insecure apps like the ones we investigated.

While increased requirements for identity verification may help to keep users safe, such requirements must be balanced with the rights of developers to anonymously distribute software. In the censorship circumvention space in particular, developers can be especially at risk of legal jeopardy and transnational repression [14]. However, while we recognize the importance of developer anonymity, we also note that anonymity is distinct from deception, and software distributors could respect authors' anonymity while still taking action against those who have misrepresented their corporate associations.

## ACKNOWLEDGMENTS

The authors would like to thank Stephanie Forrest for the initial inspiration for this work. This material is based upon work supported by the National Science Foundation under Grant Number CNS-2141547 and ICFP funding from the Open Technology Fund.

## REFERENCES

[1] Anna Ablove. 2022. VPNalyzer: Researching VPN Vulnerabilities on a Large Scale. Technical Report. University of Michigan.
[2] Genevieve Bartlett, John Heidemann, and Christos Papadopoulos. 2007. Understanding passive and active service discovery. In Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (San Diego, California, USA) (IMC '07). Association for Computing Machinery, New York, NY, USA, 57–70. https://doi.org/10.1145/1298306.1298314
[3] Nikita Borisov, Ian Goldberg, and David Wagner. 2001. Intercepting mobile communications: the insecurity of 802.11. In Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (Rome, Italy) (MobiCom '01). Association for Computing Machinery, New York, NY, USA, 180–189. https://doi.org/10.1145/381677.381695
[4] Shinyoung Cho, Zachary Weinberg, Arani Bhattacharya, Sophia Dai, and Ramsha Rauf. 2024. Selection of Landmarks for Efficient Active Geolocation. In 2024 8th Network Traffic Measurement and Analysis Conference (TMA). 1–9. https://doi.org/10.23919/TMA62044.2024.10559002
[5] Clowwindy. 2012. 发一个自用了一年多的翻墙工具 shadowsocks. https://web.archive.org/web/20120422191812/http://www.v2ex.com/t/32777
[6] Clowwindy. 2012. ShadowSocks: Stream Ciphers. https://shadowsocks.org/doc/stream.html
[7] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. 2010–. mitmproxy: A free and open source interactive HTTPS proxy. https://mitmproxy.org/ [Version 11.1].
[8] Jason A Donenfeld. 2017. WireGuard: Next Generation Kernel Network Tunnel. In NDSS. 12 pages.
[9] Leonid Evdokimov. 2011. REDSOCKS. https://darkk.net.ru/redsocks/ Accessed on: April 20, 2025.
[10] Ellis Fenske and Aaron Johnson. 2023. Security notions for fully encrypted protocols. In Free and Open Communications on the Internet. 7 pages.
[11] David Fifield. 2023. Comments on certain past cryptographic flaws affecting fully encrypted censorship circumvention protocols. Cryptology ePrint Archive, Paper 2023/1362. https://eprint.iacr.org/2023/1362
[12] Scott Fluhrer, Itsik Mantin, and Adi Shamir. 2001. Weaknesses in the Key Scheduling Algorithm of RC4. In Selected Areas in Cryptography, Serge Vaudenay and Amr M. Youssef (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 23 pages.
[13] Sheila Frankel and Suresh Krishnan. 2011. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. https://www.rfc-editor.org/info/rfc6071
[14] gfw report. 2023. Many Popular Censorship Circumvention Tools Deleted or Archived since November 2, 2023. https://github.com/net4people/bbs/issues/303 Accessed: 20 June 2025.
[15] Manaf Gharaibeh, Anant Shah, Bradley Huffaker, Han Zhang, Roya Ensafi, and Christos Papadopoulos. 2017. A look at router geolocation in public and commercial databases. In Proceedings of the 2017 Internet Measurement Conference (London, United Kingdom) (IMC '17). Association for Computing Machinery, New York, NY, USA, 463–469. https://doi.org/10.1145/3131365.3131380
[16] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar, and Vern Paxson. 2016. An Analysis of the Privacy and Security Risks of Android VPN Permission-enabled Apps. In Proceedings of the 2016 Internet Measurement Conference (Santa Monica, California, USA) (IMC '16). Association for Computing Machinery, New York, NY, USA, 349–364. https://doi.org/10.1145/2987443.2987471
[17] AppMagic Inc. 2024. AppMagic. https://appmagic.rocks/ Accessed on: June 7, 2025.
[18] Jeffrey Knockel, Adam Senft, and Ronald Deibert. 2016. Privacy and Security Issues in BAT Web Browsers. In 6th USENIX Workshop on Free and Open Communications on the Internet (FOCI 16). USENIX Association, Austin, TX, 7 pages. https://www.usenix.org/conference/foci16/workshop-program/presentation/knockel
[19] Jeffrey Knockel, Mona Wang, and Zoë Reichert. 2023. "Please do not make it public": Vulnerabilities in Sogou Keyboard encryption expose keypresses to network eavesdropping. Technical Report. The Citizen Lab.
[20] Jeffrey Knockel, Mona Wang, and Zoë Reichert. 2024. The Not-So-Silent Type: Vulnerabilities in Chinese IME Keyboards' Network Security Protocols. In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (Salt Lake City, UT, USA) (CCS '24). Association for Computing Machinery, New York, NY, USA, 1701–1715. https://doi.org/10.1145/3658644.3690302
[21] Jason Lyu. 2019. tun2socks. https://github.com/xjasonlyu/tun2socks Accessed on: 12 June 2025.
[22] Ilya Mironov. 2002. (Not So) Random Shuffles of RC4. In Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '02). Springer-Verlag, Berlin, Heidelberg, 304–319.
[23] Benjamin Mixon-Baca, Jeffrey Knockel, Diwen Xue, Tarun Ayyagari, Deepak Kapur, Roya Ensafi, and Jedidiah R Crandall. 2024. Attacking connection tracking frameworks as used by virtual private networks. Proceedings on Privacy Enhancing Technologies 2024 (2024), 109–126. Issue 3.
[24] Nightbringer21. 2016. fridump. https://github.com/Nightbringer21/fridump Accessed: 21 June 2025.
[25] National Security Agency (NSA). 2019. Ghidra - Software Reverse Engineering Framework. https://ghidra-sre.org/ Accessed: April 20, 2025.
[26] Open Corporates team. 2025. Open Corporates. https://opencorporates.com/ Accessed on April 21, 2025.
[27] Inc. OpenVPN Technologies. 2002. OpenVPN. https://openvpn.net/community-resources/ Open-source VPN software.
[28] Broken Promises. 2025. Apple Offers Apps With Ties to Chinese Military. https://www.techtransparencyproject.org/articles/apple-offers-apps-with-ties-to-chinese-military Accessed on: April 20, 2025.
[29] Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi. 2022. VPNalyzer: systematic investigation of the VPN ecosystem. In Network and Distributed System Security, Vol. 10. 16 pages.
[30] Ole André V. Ravnås and Håvard Sørbø. 2014. Frida - Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. https://frida.re/ Accessed: April 20, 2025.
[31] SensorTower. 2024. SensorTower. https://sensortower.com/ Accessed on: June 7, 2025.
[32] Skylot. 2025. JADX - Dex to Java decompiler. https://github.com/skylot/jadx Accessed: April 20, 2025.
[33] Erik Tews and Martin Beck. 2009. Practical attacks against WEP and WPA. In Proceedings of the Second ACM Conference on Wireless Network Security (Zurich, Switzerland) (WiSec '09). Association for Computing Machinery, New York, NY, USA, 79–86. https://doi.org/10.1145/1514274.1514286
[34] William J. Tolley, Beau Kujath, Mohammad Taha Khan, Narseo Vallina-Rodriguez, and Jedidiah R. Crandall. 2021. Blind In/On-Path Attacks and Applications to VPNs. In 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, 3129–3146. https://www.usenix.org/conference/usenixsecurity21/presentation/tolley
[35] Mathy Vanhoef and Frank Piessens. 2015. All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS. In 24th USENIX Security Symposium (USENIX Security 15). USENIX Association, Washington, D.C., 97–112. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/vanhoef
[36] Dovydas Vėsa and Justė Kairytė Barkauskienė. 2019. Who owns your VPN? 105 VPNs run by just 24 companies. https://vpnpro.com/blog/hidden-vpn-owners-unveiled-97-vpns-23-companies/ Accessed on: April 20, 2025.
[37] Zachary Weinberg. 2019. Toward Automated Worldwide Monitoring of Network-level Censorship. (1 2019). https://doi.org/10.1184/R1/7571876.v1
[38] Zachary Weinberg, Shinyoung Cho, Nicolas Christin, Vyas Sekar, and Phillipa Gill. 2018. How to Catch when Proxies Lie: Verifying the Physical Locations of Network Proxies with Active Geolocation. In Proceedings of the Internet Measurement Conference 2018 (Boston, MA, USA) (IMC '18). Association for Computing Machinery, New York, NY, USA, 203–217. https://doi.org/10.1145/3278532.3278551
[39] Nian Xue, Yashaswi Malla, Zihang Xia, Christina Pöpper, and Mathy Vanhoef. 2023. Bypassing Tunnels: Leaking VPN Client Traffic by Abusing Routing Tables. In 32nd USENIX Security Symposium (USENIX Security 23). USENIX Association, Anaheim, CA, 5719–5736. https://www.usenix.org/conference/usenixsecurity23/presentation/xue
[40] He Yan, Ashley Flavel, Zihui Ge, Alexandre Gerber, Dan Massey, Christos Papadopoulos, Hiren Shah, and Jennifer Yates. 2012. Argus: End-to-end service anomaly detection and localization from an ISP's point of view. In 2012 Proceedings IEEE INFOCOM. 2756–2760. https://doi.org/10.1109/INFCOM.2012.6195694

## A SUPPLEMENTARY MATERIAL ON METHODOLOGY

Table 2 contains a curated table of search terms we entered into Google, Github, and the other services from which we collected data.

| Family | Search Term | Term Location |
|---|---|---|
| A | aaa_new.png | assets directory |
| A | b1bbceaffd6c52a2 | assets directory |
| A | cert.pem | assets directory |
| A | proxy.builtin | assets directory |
| A | server_offline.ser | assets directory |
| A | libopvpnutil.so | shared library |
| A | free.vpn.unblock.proxy.turbovpn | lib/arm64-v8a |
| A | free.vpn.unblock.proxy.vpnmaster | lib/arm64-v8a |
| A | free.vpn.unblock.proxy.vpnpro | lib/arm64-v8a |
| A | free.vpn.unblock.proxy.vpn.master.pro | lib/arm64-v8a |
| A | free.fast.vpn.unlimited.proxy.vpn.master.pro | lib/arm64-v8a |
| A | free.vpn.unblock.proxy.freenetvpn | lib/arm64-v8a |
| A | free.vpn.unblock.proxy.vpnmonster | lib/arm64-v8a |
| A | free.vpn.unblock.fast.proxy.vpn.master.pro.lite | lib/arm64-v8a |
| A | free.vpn.unblock.proxy.turbovpn.lite | lib/arm64-v8a |
| A | unlimited.free.vpn.unblock.proxy.supernet.vpn | lib/arm64-v8a |
| A | 43a41a300d06092a864886f70d01010505003 | lib/arm64-v8a |
| B | libcore.so | lib/arm64-v8a |
| B | co.infinitevpn.free.proxy | libcore.so |
| B | com.free.neo.vpn | libcore.so |
| B | com.free.turbo.unlimited.touch.vpn | libcore.so |
| B | com.free.unblock.melon.vpn | libcore.so |
| B | com.free.unlimited.lemon.vpn | libcore.so |
| B | com.smart.nord.global.vpn | libcore.so |
| B | com.vpnbottle.melon.free.unblock.fast.vpn | libcore.so |
| B | com.vpncapa.vpnmaster.free.unblock.vpn | libcore.so |
| C | libxjp6xdkbew.so | lib/arm64-v8a |
| C | jsd5xcyjr5w587dk5usn | assets directory |
| C | XgS6aaUmMB0WtOkrzrTr/5-S1jSxiFu-pCAZan12C/pvsm0pA_cUa7oAzrYvGl/uJ9H4qB7ojPP2slohjpM | libxjp6xdkbew.so |
| C | z93emquwpbdk9qvqfkfc8z552vaf52szsvzbmvd6qjdynmxm7yh6nq23c9yw4drs | libxjp6xdkbew.so |
| C | kshpqn53ps | libxjp6xdkbew.so |

Table 2: Curated list of search terms for each APK, focusing on terms that were useful for linking providers.