Bypassing Tunnels: Leaking VPN Client Traffic by Abusing Routing Tables

Nian Xue
New York University

Yashaswi Malla, Zihang Xia, Christina Pöpper
New York University Abu Dhabi

Mathy Vanhoef
imec-DistriNet, KU Leuven

Abstract

Virtual Private Networks (VPNs) authenticate and encrypt network traffic to protect users' security and privacy, and are used in professional and personal settings to defend against malicious actors, circumvent censorship, remotely work from home, etc. It is therefore essential that VPNs are secure.

In this paper, we present two novel attacks that cause VPN clients to leak traffic outside the protected VPN tunnel. The root cause of both attacks is a widespread design flaw in how clients configure the Operating System (OS) to route all traffic through the VPN tunnel. This is typically done by updating the system's IP routing tables such that all traffic will first pass through the VPN client. However, some routing exceptions are added to ensure the system keeps functioning properly, namely that traffic to the local network, and to the VPN server itself, is sent outside the VPN tunnel. We show that by setting up a Wi-Fi access point or by spoofing DNS responses, an adversary can manipulate these exceptions to make the victim send arbitrary traffic in plaintext outside the VPN tunnel. We confirm our findings in practice by conducting 248 experiments against 67 of the most representative VPN providers on Windows, macOS, iOS, Linux, and Android. Our experimental results reveal that a significant number (126 and 39) and proportion (64.6% and 73.6%) of free, paid, open-source, corporate, and built-in VPN clients are vulnerable to (variants of) our two attacks respectively, suffering from leaky traffic. We discuss countermeasures to mitigate the vulnerabilities and confirm the effectiveness of selected defenses in practice.

1 Introduction

Virtual Private Networks (VPNs) were created to securely connect different physical branches of a company over existing public networks such as the Internet [54]. Without a VPN, companies would have to use private wired connections between branches. A more cost-effective solution is using a VPN, allowing one to establish a tunnel to securely send data, and connect branches, over an untrusted network. Another advantage is that VPNs allow employees to securely connect from their homes to the private network of a business or organization. These days, VPNs are also used by Internet users to send *all* traffic over the VPN tunnel. This is done to protect online identities, hide IP addresses, circumvent censorship, access blocked content, and perhaps most important, prevent attackers from inspecting and manipulating the user's traffic.

The security of open tunneling and VPN protocols has been widely studied in various works. One of the first discoveries was by Schneier and Mudge in 1998, who found cryptographic flaws in the Point-to-Point Tunneling Protocol (PPTP) [53]. Researchers also discovered a host of other cryptographic flaws in VPN protocols, with IPsec being a common target [3, 8, 9, 16, 17, 23]. However, the integration of these protocols into real-world clients and platforms has not been widely studied. Motivated by this, we found techniques to bypass cryptographic protections and leak traffic in plaintext. As a result, our attacks are independent of the cryptographic protocol being used. Moreover, in contrast to the already-known issue that badly configured VPN clients may leak DNS or IPv6 traffic outside the VPN tunnel [30, 35, 47, 49], our novel attacks allow an adversary to leak arbitrary IP traffic.

Recently there has also been a growing research focus on the VPN ecosystem as a whole [35, 49], including how VPNs are advertised and what users think about them [4, 20, 43]. One complexity is that the security of VPN software is highly dependent on the implementation and configuration used by specific vendors, making it non-trivial to gain a complete picture of the ecosystem. For instance, out of necessity, several works focus on a specific platform [30, 56, 64, 65] or focus on a specific protocol such as IPsec [3, 8, 9, 16, 17, 23]. To gain a proper understanding of how widespread the vulnerabilities that we discovered are, a large-scale evaluation was performed. We evaluated different versions of Windows, iOS, macOS, Android, and Linux for a wide range of VPN software, including free and paid clients, open-source clients, and commercial clients. Our results indicate that none of the VPN providers or clients are secure on all platforms.

The idea behind our novel attacks is to manipulate the client into sending (selected) traffic outside the VPN tunnel, meaning it will be leaked in plaintext. Under normal conditions, the client's routing table assures that all traffic generated by the client will be sent through the VPN tunnel. However, most clients add exceptions to their routing table to send the following two types of traffic outside the VPN tunnel:

(1) Traffic sent *to and from the local network*; and

(2) Traffic sent *to and from the VPN server*.

The first rule assures that the local network remains accessible while using the VPN, meaning the user can use the VPN while simultaneously printing files, performing screen casting to a local display, and so on. The second rule assures that

there is no routing loop, i. e., that already-encrypted VPN packets do not undergo encryption again. Unfortunately, we demonstrate that an adversary can manipulate these routing exceptions such that arbitrary traffic will be sent outside the VPN tunnel. First, an adversary can manipulate the IP address range of the local network, causing leakage of Internet traffic. Second, under the right conditions, an adversary can spoof the IP address of the VPN server, again causing leakage of (arbitrary) traffic due to the above routing exceptions.

The above two routing exceptions lead to two attack types, which we call *LocalNet* and *ServerIP attacks*, respectively. The main goal of these attacks is to leak traffic in plaintext, but our attacks may also lead to the targeted blocking of traffic while the VPN is being used. The LocalNet attacks assume that the adversary can control the IP range assigned to the local network where the victim situates, which is trivially feasible if the adversary is a rogue Wi-Fi Access Point (AP). The ServerIP attacks assume that the adversary can spoof DNS replies, which is possible when the adversary acts as a rogue AP or when the adversary controls a (core) router near the victim. Our attacks are not computationally expensive, meaning anyone with the appropriate network access can perform them, and they are independent of the VPN protocol being used. Overall, our attacks break the security and anonymity that VPNs are expected to provide. Even if the victim is using another layer of encryption such as HTTPS, our attacks reveal which websites a user is visiting [46], which can be a significant privacy risk. Moreover, the leaked traffic can contain sensitive data if plaintext HTTP is used, and our attack can be used as a basis to subsequently attempt to attack HTTPS.

To summarize, our primary contributions are:

- We reveal security and privacy vulnerabilities in VPN clients. These vulnerabilities are configuration-specific, affect a large number of VPN apps, and result in our LocalNet attacks and ServerIP attacks.
- We realize attack setups and evaluate which VPN clients are vulnerable to our two attacks, by conducting a large set of experiments against both desktop and mobile platforms, covering Windows, macOS, and Linux, as well as Android and iOS.
- We propose and discuss practical mitigations for both of our novel attacks, and will make public the instructions and scripts we used for conducting the attacks in order to achieve replicability of the experiments [2].

Disclosure: We reported our findings to CERT/CC and to affected VPN vendors for which we found security contact information. Further details can be found on our repository [2].

2 Background

In this section, we give an introduction to VPNs and explain how they typically route IP traffic and configure DNS servers.

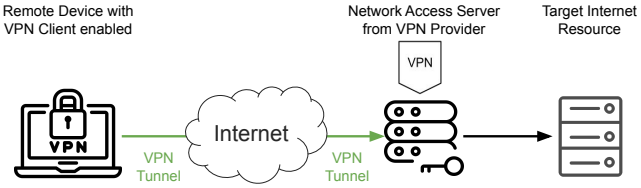

Figure 1: Typical full-tunnel VPN configuration where the client sends all traffic through the VPN server.

2.1 Virtual Private Networks

VPNs have attracted a large number of users, for instance, 31% of all internet users have at some point used a VPN [58]. The history of VPNs dates back to 1996 when the first VPN protocol, called PPTP, was introduced by Microsoft [26, 53]. As time passed, VPNs grew into essential tools for staying safe and secure online. Current features of VPNs go far beyond their original purpose: they can now secure Internet connections, hide physical locations, evade government censorship, allow to access geo-blocked media content, prevent malware and hacking, etc. VPNs can be classified in multiple ways. For our attacks, the most important difference is between split-tunnel and full-tunnel VPNs. In a *split-tunnel* VPN, only a subset of all traffic will be sent through the encrypted tunnel. For instance, when a VPN is used to remotely access the internal network of a company, the VPN client is configured to only send traffic to the company network over the VPN tunnel, while all other traffic is sent as normal over the Internet. Split tunneling is also used to specify that only selected apps will (not) use the VPN. In a *full-tunnel* VPN, on the other hand, all client traffic is sent through the tunnel, and the VPN server then forwards traffic to its final destination (see Figure 1). In this paper, we will target VPNs that are intended to be full-tunnel ones, but we will show how they can be manipulated to still send traffic outside the VPN tunnel.

To use VPNs, users have multiple options: they can use: (1) the built-in VPN functionality of the OS; (2) a standalone VPN client developed by a VPN provider; or (3) a 3rd-party client such as OpenVPN Connect [44], Tunnelblick [57], or Windscribe [63] which can load configuration files created by a VPN provider. The behavior of the VPN client will depend on exactly which software is used. Currently, most commercial VPN providers offer a standalone client, which helps users set up VPN connections in an easy manner. In addition, a few VPN clients support only one OS, while others, like Cisco AnyConnect [12], are available for multiple mainstream operating systems across both desktop and mobile platforms.

When using a standalone or commercial VPN client, the client will often pull configuration data from the remote VPN provider's server before creating the VPN tunnel. This configuration data can include a list of VPN servers from different geographic areas, the IP addresses of these servers, the sup-

```
[author@zbook ~]$ ip route
# Default: all traffic is sent over the tun0 interface. This interface represents the VPN tunnel. So by default all traffic goes through the VPN tunnel.
default via 10.8.0.1 dev tun0
# Exception 1: local network traffic is directly sent to the destination
192.168.197.0/24 dev enp0s20f0u4 proto kernel scope link
# Exception 2: traffic to the VPN server (176.126.240.111) is sent to the router (192.168.197.34) so the VPN client can still reach the VPN server
176.126.240.111 via 192.168.197.34 dev enp0s20f0u4
```

Figure 2: Example of two VPN routing exceptions.

ported VPN protocols for each server, etc. Users can generally decide which VPN protocol to adopt while creating the VPN tunnel, with common options being OpenVPN, WireGuard, or IPsec. As a result, the precise configuration of the client, and whether it is vulnerable to (variants of) our attacks, may depend on the chosen VPN server and protocol.

2.2 IP Routing Table and VPNs

When a VPN client is installed or started, the software will typically create a virtual network adapter that represents the VPN tunnel. Any packets sent over this virtual network adapter will be handled by the VPN client, which will encapsulate the packets using the selected VPN protocol. The encapsulated packets are then sent to the VPN server, where the plaintext header of the encapsulated packets always contains the IP addresses of the client and VPN server, and the real IP destination address of the original packet remains hidden.

To forward all traffic through the tunnel, a typical approach is to add a default route to the host's IP routing table with as the next hop the VPN's virtual interface.[1] Figure 2 shows an example of this: a default rule is added that causes all traffic to be sent over `tun0`, which is the VPN tunnel's interface. As a result, by default, all traffic is sent over the VPN tunnel instead of the host's physical interface so that outgoing traffic is encrypted and authenticated by the VPN protocol.

However, as we will demonstrate in this work, in real-world scenarios and on several operating systems, the IP routing tables will also contain exceptions to send select traffic outside the VPN tunnel. In particular, more specific rules are typically added such that packets to the local network and to the VPN server itself are sent outside the VPN tunnel.

2.3 Configuration of DNS Servers

After establishing the VPN tunnel, a secure VPN client will configure a trusted DNS server. This is necessary because the default DNS server provided by the local network cannot be trusted. That is, without updating the DNS server, a malicious local network, such as a rogue Wi-Fi hotspot, can trivially intercept traffic after the VPN tunnel is established [47].

[1]The routing table lists the next hop, i.e., the next network host, to send packets to in order to reach specific network destinations [21].

3 Threat Model

In both of our attacks, we assume that the encrypted VPN tunnel is not vulnerable to known attacks, e. g., it does not leak traffic during tunnel failures [49] and does not leak DNS or IPv6 traffic [47]. Additionally, we assume that the underlying cryptography used by the VPN protocol cannot be broken, that is, we assume a Dolev-Yao model where an adversary can intercept, block, and modify network packets, but cannot break cryptographic primitives [18]. We also treat the VPN protocol itself as a perfect cryptographic primitive that cannot be broken, e. g., traffic cannot be decrypted using analytical attacks. Finally, we assume that credentials are secure, e. g., passwords cannot be brute-forced. In both attacks, the adversaries share the same goal: make the VPN users send their traffic in plaintext, i. e., outside the VPN tunnel.

An important difference between our two attacks is where the adversary can be located. In line with previous works, we use the terms local-network and on-path attacker to differentiate between these two positions [5, 33, 37, 56]:

Local-Network Attacker: In our LocalNet attacks, the adversary is a local-network attacker (see Figure 3). This means the adversary operates within a local network and waits for the victim's access, e. g., by setting up an open AP. They can actively observe and manipulate the traffic between the AP and the victim. Additionally, the local-network attacker can control the local IP address configuration of any device within the local network, including the IP address and netmask assigned to the victim. In practice, if the adversary controls a public AP, they can assign IP addresses to clients using the Dynamic Host Configuration Protocol (DHCP). The adversary can then assign itself an arbitrary IP address as well. For instance, the adversary can use the subnet 1.2.3.0/24 for the rogue AP and assign itself an IP address in this subnet. In our LocalNet attacks, whenever the victim contacts an IP address within this subnet, it will send data directly (outside the VPN tunnel) to this local host instead of sending it to the real website through the VPN tunnel.

On-Path Attacker: In our ServerIP attacks, we assume the adversary is an on-path attacker that must be located anywhere on the network path between: (1) the victim and the DNS server used to look up the VPN server's IP address; and (2) the victim and the target server towards which we want to leak traffic. The adversary is able to intercept and modify network traffic originating from the VPN user and destined to the DNS server, and vice versa (see Figure 4 on page 5). Such a type of attacker could, e. g., be an ISP, a compromised core-network router, or a local attacker such as a rogue AP at public places like an airport or a hotel trying to obtain leaked traffic. We assume this position enables the adversary to intercept DNS requests and spoof DNS replies. This adversary must also be able to intercept traffic from the victim to the IP address(es) in the spoofed DNS reply.

4 IP-Routing Exception Attacks

In this section, we describe our two novel attacks. In both attacks, the adversary manipulates rules in the client's IP routing table such that selected traffic will be leaked outside the VPN tunnel. Recall that when creating the VPN tunnel, a virtual network adapter is typically created that represents this tunnel. Any traffic sent over the virtual adapter is encrypted by the VPN protocol. Additionally, the client's routing table is updated to make all traffic pass through this virtual adapter, ensuring that all outgoing traffic is encrypted. However, two common rules are added that specify that certain traffic should be sent outside the VPN tunnel, namely (see also Figure 2):

1. Traffic directed to the local network.
2. Traffic directed to the VPN server.

Unfortunately, the first exception can be manipulated by a local-network attacker, while the second exception can be manipulated by an on-path attacker. By manipulating these exceptions, an adversary can make the victim leak arbitrary traffic outside the VPN tunnel, leading to our LocalNet attacks and ServerIP attacks, respectively. The attacks are independent of the VPN protocol being used, meaning all widely-known protocols such as PPTP [26], OpenVPN [44], IPsec/IKEv2 [34], SSTP [39], and WireGuard [19], are all affected.

4.1 LocalNet Attacks

4.1.1 Leaking local traffic. Our first attack exploits a common routing exception in many VPNs, namely allowing direct access to the local network, meaning traffic to local IP addresses is not sent through the VPN tunnel. This means that a user can access other devices in this network while using the VPN. This is usually done out of convenience, for instance, so the user can connect to a local printer or cast videos to a nearby display. Excluding local traffic from the VPN tunnel is the default configuration for a large number of VPN clients, as we will demonstrate through experiments. In addition, although a number of VPN clients offer configuration options regarding local network access, they typically allow access by default. Only a few VPN clients, such as Mullvad VPN, always block traffic sent to the local network when using the VPN. Allowing local network access, however, introduces a new attack vector: the adversary can be a rogue AP that assigns arbitrary IP ranges to the local network, with serious security and privacy consequences.

The LocalNet attacks abuse the local-network-accessible feature to make the victim send arbitrary traffic outside the VPN tunnel without encrypting it. For instance, it can make the victim leak traffic towards a targeted web server. To achieve this, the local-network attacker creates a rogue AP and sets the local network's IP range to one that contains the IP address of the target server. The VPN client on the victim's device will then be tricked into believing that the website it is trying to connect to is in the local network and thus will *not* use the VPN tunnel when communicating with the website. This leads to the data of the user being sent outside the VPN tunnel without encryption, causing a serious data leak.

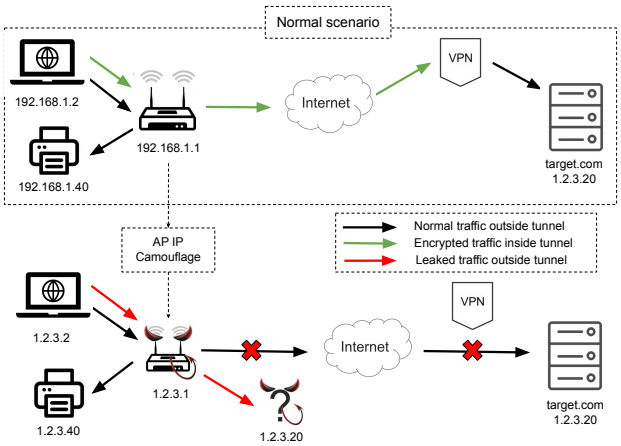

Figure 3: Network diagram of the LocalNet attacks where traffic to `target.com` is leaked outside the VPN tunnel and then redirected to a malicious server (see Section 4.1).

Figure 3 illustrates this scenario with an example. Normally, a local network will use a private IP address range such as 192.168.1.0/24 [42]. If a user in this network tries to access a web server with IP address 1.2.3.20, the VPN tunnel is used. However, if the AP uses the IP address range 1.2.3.0/24 for its local network, the target server's IP address is within this local network. Thus, the victim's VPN client will believe that the web server it is trying to connect to is in the same local network as itself. As a result, the client will try to connect to the web server directly without using the VPN tunnel.

Our attack can also be used to intercept nearly all IP traffic by assigning the IP range 0.0.0.0/1 or 128.0.0.0/1 to the local network. The adversary can then forward leaked packets to the real server and, in case the victim is not using another layer of protection such as HTTPS, monitor the resulting connection to steal user data. Alternatively, if plaintext HTTP is used, the adversary can set up a local server with the same IP address as the real server to host a malicious copy of the website and, e. g., steal user credentials or serve malware. In case HTTPS is used, the attack reveals the IP address that the victim is communicating with, which is often enough information to learn the website being visited [46]. Moreover, an adversary can also try to launch subsequent attacks against any higher-layer security protocol to steal user information (see Section 7.1).

4.1.2 Blocking traffic. We observed that with some VPNs the attack did not cause the victim to send packets outside the VPN tunnel, but instead caused the victim to block packets toward the target IP address(es). We also consider this a security risk. As a concrete example, an adversary can abuse this to block the traffic and warnings of a victim's security camera

app, or block security updates, while other apps of the victim do work properly, making the victim unaware of the attack.

4.2 ServerIP Attacks

4.2.1 Leaking server IP traffic. Our second attack exploits another common routing exception: traffic sent to the IP address of the VPN server is not (again) sent through the VPN tunnel. This prevents routing loops by not reencrypting already-encrypted traffic to the VPN server.

Leaking traffic to the VPN server's IP address can be (low) security risk on its own. In particular, it can be abused to deanonymize visitors of a website by making the website initiate a HTTP request for the VPN server's IP address [6, §5.2]. This can be achieved by abusing benign functionality of the website, such as including a remote image in a forum post, or by compromising the website's server. By then monitoring for plaintext traffic to the VPN server's IP address, the real IP address of the visitor can be learned.

4.2.2 Leaking arbitrary traffic. If an adversary can control the IP address that the client uses for the VPN server, the above routing exception can be manipulated to leak arbitrary traffic. One common method of how clients learn the VPN server's IP address is through plaintext DNS, meaning the adversary can spoof DNS replies to change the server's address. In other words, we will exploit clients that use plaintext DNS, and we will not target clients that hardcode the VPN server's IP address(es) in their code or in their configuration files (see Section 5.6 for more details). In this scenario, an on-path attacker between the victim and its DNS server can then intercept DNS requests and spoof responses. Note that a local-network attacker would also be able to spoof DNS responses. In fact, if the adversary acts as a rogue AP, they can use DHCP to assign clients a preferred DNS server, and this server can even be controlled by the adversary itself.

After spoofing the VPN server's IP address, the victim tries to connect to the server at the spoofed IP address. The attacker redirects these packets to the real VPN server so the client still successfully establishes the VPN tunnel. This means that the on-path attacker must also be located between the client and the spoofed server IP address in order to redirect these packets. Once the client establishes the VPN tunnel, all traffic to the spoofed IP address will *not* be sent through the VPN tunnel. For instance, by making the spoofed IP address coincide with a legitimate web server, all traffic to this web server will be sent directly to the server outside the VPN tunnel.

The resulting attack is illustrated in Figure 4, where the adversary's goal is to intercept data sent to `target.com`, i.e., to 1.2.3.4. Under normal conditions in this example, the VPN server `vpn.com` has the IP address 2.2.2.2, meaning the victim will send traffic to the target website through the VPN tunnel. However, the attacker will spoof DNS responses so the victim thinks that the VPN server `vpn.com` has the IP address 1.2.3.4, and will redirect VPN traffic to the real VPN server so the

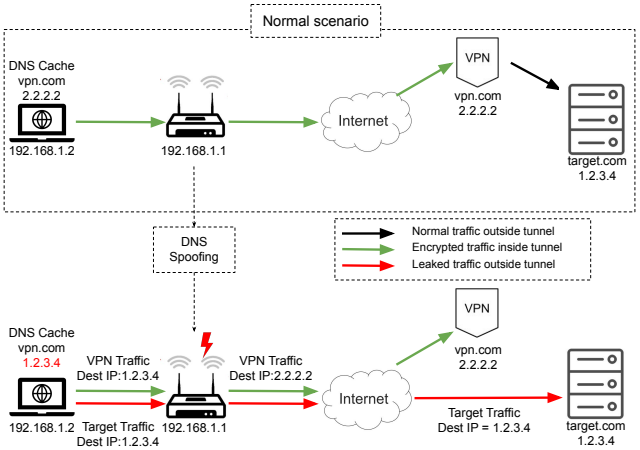

Figure 4: Network diagram of the ServerIP attacks: traffic to `target.com` is leaked outside the VPN tunnel (Section 4.2).

victim can still successfully establish a VPN connection. As a result, all traffic to 1.2.3.4, so to `target.com`, will now be sent in plaintext outside the VPN tunnel.

Our attack can also be abused to (try to) intercept all traffic to the DNS server set by the VPN client. The adversary can then intercept DNS requests and spoof replies, enabling the interception of most IP-based traffic. Note that the IP address of the DNS server set by the VPN client can be deduced from the VPN used by the victim, and the VPN being used can be deduced from the VPN server's IP address.

4.2.3 Blocking traffic. A VPN client may block traffic to the spoofed IP address of the VPN server. Some clients block traffic on all ports except the port used by the VPN protocol. Similar to the attack of Section 4.1.2, this can be abused to block selected traffic, which we consider a security risk.

5 Measurements and Experimental Validation

In this section, we evaluate our attacks against a large set of VPN clients. We describe the VPN clients and hardware used in our tests, then elaborate on the experimental setup used to test whether these clients are vulnerable, and finally, we provide an overview of results and special observations.

5.1 VPN Selection Criteria

The VPN market is huge and dynamic: there are hundreds of VPNs across the Internet with frequent version updates and new products, making it infeasible to evaluate all of them. To pick an influential and representative set of VPNs for our evaluation, we used the following selection criteria:

1. **Widely-used and influential.** We picked the most widely-used VPNs to obtain a representative sample. For Android, we can directly see the number of downloads for each app on the Google Play Store, and pick

the most downloaded ones. For other platforms such as iOS and macOS, the number of downloads per app is not visible, and we instead use the number of user ratings to pick the most rated apps. The idea is that an app with more ratings will likely have more users.

2. **Multiple OSs and versions.** In our evaluation, we also cover multiple platforms, namely Android, iOS, macOS, Windows, and Linux. This is important because the behavior of an app may depend on the platform that it is running on. Therefore, we also explicitly picked apps that support multiple platforms. Additionally, the features and behavior of an app may depend on whether a free or paid version is used, and because of this, we will also test both paid and free VPN clients.
3. **Overlap with previous papers.** To enable an easier comparison with related work, we also evaluated VPNs that recent papers studied [10, 30, 35, 47, 49]. That is, we picked VPNs that are evaluated in all or many of these papers. These papers themselves chose VPNs based on popular review and recommendation websites, personal recommendations, widely-used free and paid ones, etc.

In total, we investigated 63 VPN applications and 4 OS built-in VPNs on multiple mobile and desktop platforms over five mainstream operating systems. All combined, we tested 195 unique VPN client and OS combinations for the LocalNet attacks, and 53 unique combinations for the ServerIP attacks.

5.2 Hardware and Software Setup

We tested our attacks against 15 devices that cover mobile and desktop platforms: three iPhones (iPhone 12 mini with iOS 14.1, iPhone 12 with iOS 16.1.1, and iPhone 13 Pro Max with iOS 15.6.1), three Androids (Android 9, 12 and 13), one Android pad (Android 8.1.0), two MacBooks (macOS 12.6 and 13.0.1), two laptops running Windows 10 and 11 Pro, and laptops running four different Ubuntu releases (16.04, 18.04, 20.04, and 22.04). To create a rogue AP we use a Panda 300Mbps Wireless 802.11n USB dongle, a laptop running Ubuntu 16.04, and version 0.4.6 of the `create_ap` script to start and configure the AP [36]. Finally, we use a web browser, e. g., Chrome or Safari, to check if we can visit the target websites outside the VPN tunnel, and we also double-check leaks outside the VPN tunnel by inspecting traffic in Wireshark.

Once the rogue AP is started using the `create_ap` script, victim devices can connect to it for access to the Internet, and all the traffic coming from the victim device can be inspected with the Wireshark packet analyzer. The `create_ap` script also allows users to configure the IP address of the AP and thereby the subnet used by the local network. In our tests, this IP address is picked such that the resulting subnet of the local network contains the target website's IP address.

To test paid VPN clients, we created an account for each client, paid for the service, and verified its correct operation. When no standalone client was available, but VPN profiles like OpenVPN configurations were offered by the VPN provider, we downloaded the config files, and then conducted the tests using the OS built-in VPN or a 3rd-party VPN client.

5.3 LocalNet attacks: Experimental Setup

The steps to perform the LocalNet attacks are shown in Figure 5 and were implemented to determine if a VPN client is vulnerable. The goal of these steps is to make the victim leak traffic toward `target.com` outside the VPN tunnel. In our tests, this website is not using HTTPS, meaning if it is accessed outside the VPN tunnel we can then redirect the request to a malicious phishing clone of the website. This malicious copy is hosted in our local network on a Windows web server running Internet Information Services (IIS). The victim will only see this (easily recognizable) malicious copy if it tries to access it outside the VPN tunnel, i.e., it will only see it when vulnerable to our attack.

In step ① of the attack, we assume that the victim has connected to the rogue AP and that it will use DHCP to request an IP address. In step ②, the rogue AP will *not* reply with a typical RFC1918 private IP address, but it will instead hand out an IP address within the same subnet as the IP address corresponding to `target.com`. In our example, the IP address of the target is 1.2.3.20, and the rogue AP uses the corresponding subnet 1.2.3.0/24 for the local network. When the VPN client then establishes a secure tunnel in step ③, it will update the client's routing table to send all traffic through this tunnel, *except* traffic towards devices within the local network so that local devices remain accessible when using the VPN. This means most internet traffic will still use the VPN tunnel, as indicated in step ④, meaning the victim is unlikely to notice the attack since the VPN keeps working as expected.

In step ⑤, the victim visits `target.com`. Since the IP address of this website now falls within the local network, the victim will directly access the server without using the secure VPN tunnel. That is, the client will use ARP to determine the MAC address of the local device that has IP address 1.2.3.20, and will then send traffic directly to the malicious clone of the website that is hosted within the same local network.

We remark that in the above example, all traffic to IP addresses in the subnet 1.2.3.0/24 will be sent outside the VPN tunnel. The attack can be made more targeted by defining the local network as a smaller subnet such as 1.2.3.16/29. This would make the victim send traffic to IP addresses within the range 1.2.3.16-23 outside the VPN tunnel. Alternatively, an adversary can assign the IP range 0.0.0.0/1 or 128.0.0.0/1 to the local network to intercept nearly all IP traffic.

5.4 LocalNet Attacks: Results

The number of VPN clients vulnerable to LocalNet attacks is summarized in Figure 6 and a detailed overview is presented

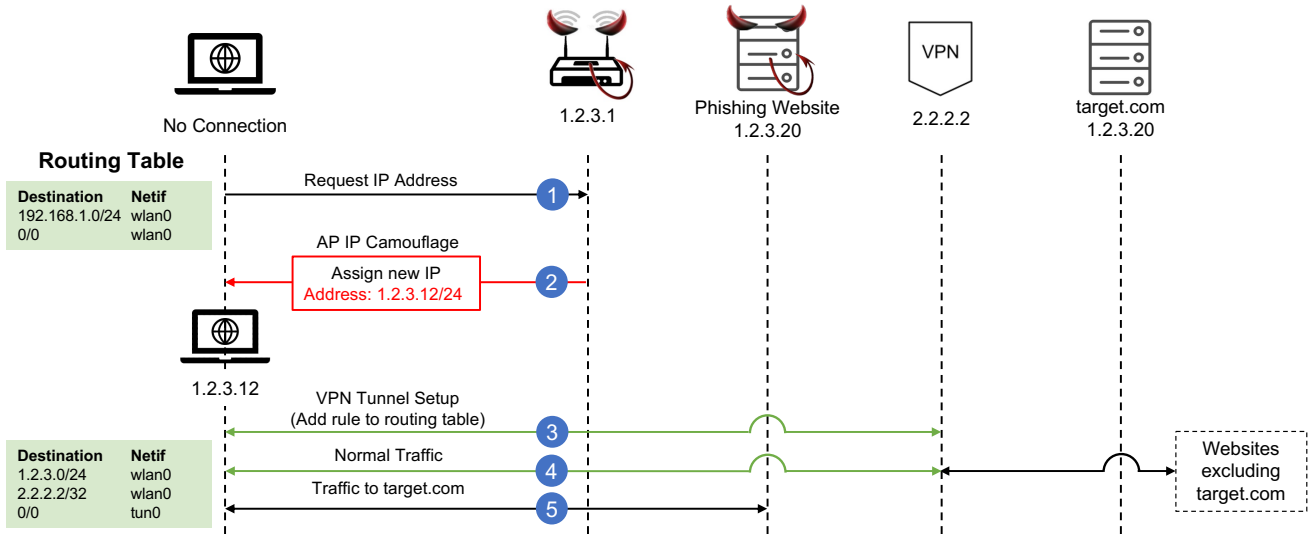

Figure 5: Illustration of how the LocalNet attacks can be used to leak traffic towards `target.com` outside the VPN tunnel. In this example, the leaked web request is then redirected to a malicious phishing website.

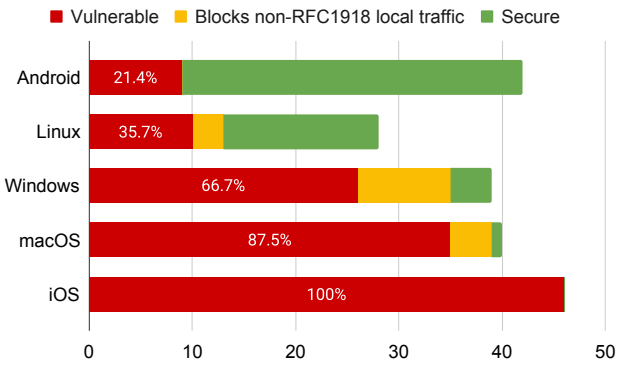

Figure 6: Number of VPN clients vulnerable to (variants of) the LocalNet attacks for each tested OS.

in Table 1 in the Appendix. In total, we tested 63 VPN applications, which includes 18 paid ones, and we also tested 4 built-in ones. Some of the apps were available for all five OSs, while others were only available for specific OSs: we tested 42 apps for Android, 46 for iOS, 39 for Windows, 40 for macOS, and 28 for Linux. Overall, we can see that the attack affects a large number of VPN apps. Following an in-depth analysis, we make the following observations:

5.4.1 The OS influences the security of VPN clients. From Figure 6 we observe a clear correlation between the OS and the vulnerability of an app. For instance, most VPN apps on Android were not vulnerable (33 out of 42), while all of those tested on iOS were vulnerable on at least one device (see also Observation 5.4.2). On Linux, 10 out of 28 VPNs were vulnerable, on Windows two-thirds were vulnerable (26 out of 39), while on macOS 35 out of 40 VPNs were vulnerable.

One reason why Android is more secure is because, since version 4.4, it uses policy-based routing where routing decisions are also made based on the app generating the traffic instead of purely on the destination address [13, 47, 48]. This is implemented using per-UID routing [41].

We also noticed that although VPN clients may be developed by the same company, their behavior and default settings on different platforms vary. For instance, depending on the platform, access to the local network is allowed or not, their firewall is on or off, and the same VPN provider may be vulnerable in one platform but not the other (see Table 1). In particular, we observe many providers whose app is secure on Android but vulnerable on iOS and/or macOS. We can conclude that the OS has a major influence on a client's security and that this is not solely a problem of the app developer.

5.4.2 All VPN apps on iOS are vulnerable and only one VPN app on macOS is secure. All tested VPNs on iOS leak traffic (see Figure 6 and Table 1). On macOS, only Cisco AnyConnect was secure. All other VPN clients on macOS leaked traffic, with the exception of ExpressVPN and Mullvad who blocked access to IP addresses within the local network, which we also consider a security risk (recall Section 4.1.2).

Surprisingly, Psiphon was secure on our iPhone 12 mini running iOS 14.1 but was vulnerable on all other tested iOS devices (iPhone 13 Pro Max, iPad 8th gen, iPhone 12 Pro, and iPad mini 2). Inspection of the Psiphon source code revealed that when using iOS 14.0 or 14.1, it sets the API parameter `includeAllNetworks`, but not `excludeLocalNetworks`, resulting in secure behavior (see Listing 1 in the Appendix). Older iOS versions do not support these parameters, and on iOS 14.2 and above Psiphon always sets both, making it vul-

nerable. Few vendors set `excludeLocalNetworks` because it may cause Airdrop to no longer work [7] and because it may reduce overall network reliability [28]. This causes iOS to be less secure. Overall, having nearly all VPN apps vulnerable on macOS and iOS shows that its OS is partly to blame.

To prevent the attack on iOS, the above two parameters can be used to disallow local network access when the local network uses non-RFC1918 IP addresses. For instance, Windscribe updated its client with this approach and was the first to prevent the attack on iOS.

5.4.3 8.2% of clients block traffic towards non-RFC1918 IP addresses in the local network. With 16 out of 195 tested VPN/OS combinations, the attack did not cause traffic leakage, but instead made the victim unable to visit the target website, with the browser usually showing an error that the request was blocked by a firewall. These cases are represented by the "Blocks" category in Figure 6 and by the △ symbol in Table 1. We investigated the root cause of this issue for a random subset of affected VPNs. This revealed that these VPNs configure the routing table to send data to the local network outside the VPN tunnel. However, they also employ a firewall that blocks all outgoing traffic except (encrypted) packets to the VPN server or packets to private RFC1918 IP addresses. This means that traffic to IP addresses in the local network gets blocked instead of being sent through the VPN tunnel. An adversary can abuse this to block access to certain IP addresses, which is also a security risk (see Section 4.1.2).

On Windows, X-VPN blocked local traffic but broadcasted ARP requests when a process tries to access a local IP address. We conjecture that this is because the Windows firewall was configured to block *packets*, but not the initiation of *connections*, to local IP addresses. An adversary can abuse this to determine whether the victim is visiting a sensitive website. In particular, when the adversary sees an ARP request for the IP address corresponding to `target.com`, then they know the victim is trying to load this website.

5.4.4 The behavior of three VPN clients depended on the version being used. Most standalone clients remain vulnerable even with a version update if their earlier versions are vulnerable. However, among these clients we investigated, only three VPN clients (Network Manager OpenVPN Gnome, OpenConnect CLI, and OpenVPN CLI) on four different Ubuntu releases (16.04, 18.04, 20.04, and 22.04) behave differently. We find that the earlier two versions on Ubuntu 16.04 and 18.04 are vulnerable, whereas the later two versions are secure. In addition, by comparing both free and paid versions on several clients (e. g., Hide.me VPN and VPN Proxy Master), we observe that the paid services are no more secure than the free ones on the same clients in our experiments.

5.4.5 Three VPN apps used DNS servers that always returned special use IP addresses. We encountered three cases where the VPN client was using a DNS server that returns IP addresses in the range 198.18.0.0/15 for *all* domains (see the symbol † in Table 1). These IP addresses are normally used in benchmark tests of network interconnect devices [1, 14]. Packets with a source address from this range are not meant to be forwarded over the Internet, and in this sense, they can be treated as private IP addresses. Interestingly, the VPN server redirected traffic to these special IP addresses to a real IP address of the domain, allowing the user to still access all websites. It is unclear why these VPN providers do this, but we conjecture that one advantage is that if a client accidentally leaks traffic outside the VPN tunnel, then due to the usage of these special IP addresses the leaked packets are less likely to be forwarded over the Internet.

Our default attack fails against these clients because they use an IP address from this special range to visit `target.com`. However, by using a subnet within 198.18.0.0/15 for the rogue AP's local network, our attack remains possible.

5.4.6 Several VPNs have an option to (dis)allow local network access. Some VPNs have an option to (dis)allow access to the local network while using the VPN. Interestingly, we found that the default setting of this option can vary depending on the OS. Some of these apps such as Surfshark, ExpressVPN, and Windscribe (iOS) had their local access setting turned on by default. So users who are unaware of this setting may be in a vulnerable position by default.

For ExpressVPN and Windscribe, the local network access option had no effect on iOS and Android. That is, when using these clients on Android, access to the local network was always disallowed independent of this setting, and on iOS access to the local network was always allowed. This implies that on iOS, configuring these clients to disallow access to the local network will *not* prevent our attack.

5.4.7 Windscribe and ExpressVPN only allow local network access when using RFC1918 private IP addresses. The open-source Windscribe client on Windows and macOS only allows enabling local network access when the local network is using private IP addresses as defined in RFC1918 [42]. In particular, local network access is only allowed when using 192.168.0.0/16, 172.16.0.0/12, or 10.0.0.0/8. Similar behavior was observed with the proprietary ExpressVPN client on Windows and macOS, but there was no warning in the user interface. Although this behavior prevents traffic leaks, it has the side effect that the victim cannot visit the target website anymore as described in Observation 5.4.3.

With Windscribe on Android, local network access is disabled by default but can be enabled independent of the IP addresses used by the local network, and doing so makes the client vulnerable. On iOS, access to the local network is enabled by default, and can be turned on or off independent of the IP addresses used for the local network, meaning Windscribe on iOS is by default vulnerable to our attack.

5.4.8 Four tested VPNs provide only a browser whose traffic is secured. Four tested VPNs do not tunnel all traffic generated by a system, but instead, provide a browser where

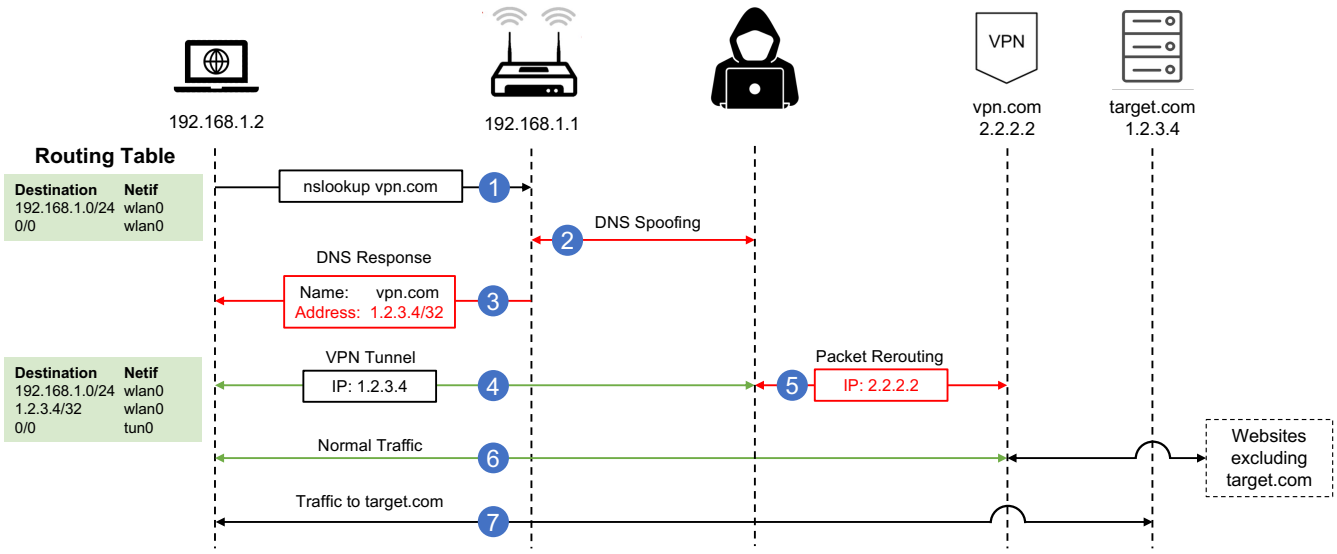

Figure 7: Illustration of how the ServerIP attacks can be used to leak traffic towards `target.com` outside the VPN tunnel.

only traffic generated within this browser is tunneled over the VPN connection. These were VPN Proxy Browser, Smart VPN: Proxy, Best VPN Unlimited proxy, and VPN Proxy: Fast & Unlimited. Traffic outside these VPN-browsers will be sent in plaintext outside the tunnel, which can be considered a security risk [49, §VI.H]. However, our attack cannot be used to leak browser traffic since that traffic is always tunneled. Since these VPNs do not tunnel all traffic of a device, they are excluded from Figure 6 and Table 1.

5.5 ServerIP Attacks: Experimental Setup

The steps to perform the ServerIP attacks, with the main goal to make the victim load `target.com` outside the VPN tunnel, are shown in Figure 7. For convenience, we host this website on a DigitalOcean server. The website displays the IP address and port of the client, making it easy to detect if the attack succeeds: if the client loads the page and it shows the VPN server's IP address, then the attack failed. Conversely, if it shows the victim's real IP address, then the attack succeeded.

In step ① of the attack, we assume the victim has connected to our rogue AP, and that it uses the DNS server that was advertised in the DHCP responses of the rouge AP. We assume the victim uses this DNS server to look up the IP address of the VPN server, i. e., the IP address of `vpn.com`. Since the rouge AP can advertise a DNS server under the control of the adversary, we can trivially make the server return the wrong IP. In our attack, in step ②, the DNS server returns the IP address of the target website instead of the IP address of the VPN server. We achieve this in practice using the `create_ap` script with the parameters to set up a custom dnsmasq server that uses the Linux hosts file to return the spoofed IP address.

After the client receives the DNS response in step ③, it will try to connect to the VPN server at the incorrect IP address in step ④, which in Figure 7 is 1.2.3.4. To ensure the victim can successfully establish the VPN tunnel, the adversary will reroute all the VPN packets sent to 1.2.3.4 to the real VPN server at 2.2.2.2 (step ⑤). We use iptable rules to redirect VPN traffic to the real IP address of the VPN server and we detect VPN traffic based on the destination IP address and the port. This iptable rule also ensures that responses from the real VPN server are forwarded to the client. All combined, the adversary does not break the cryptographic VPN tunnel, but simply forwards VPN packets to the real VPN server (and back) by simply rewriting the destination IP addresses.

After establishing the VPN tunnel in steps ④ and ⑤, the client updates its routing table to send all traffic over the VPN tunnel except traffic to the VPN server itself. In other words, all traffic to 1.2.3.4 will be sent directly to its destination outside the tunnel. The victim is unlikely to notice this attack since all other traffic is still sent normally over the VPN tunnel (step ⑥). Finally, in step ⑦ all traffic to `target.com` is now sent directly to 2.2.2.2 outside the VPN tunnel.

In case the VPN server uses the same port as the target traffic, deep packet inspection must be used to differentiate VPN tunnel traffic from the target traffic. For instance, OpenVPN or DTLS packets can be differentiated from leaked HTTPS traffic due to the plaintext header used in OpenVPN and DTLS.

Although we act as a rogue AP in our experiments, a real adversary can also be an on-path attacker (recall Section 3).

5.6 ServerIP Attacks: Results

We noticed that the custom clients of most VPN providers are not vulnerable to the ServerIP attacks. This is because they do not use DNS to find the VPN server's IP address, which will

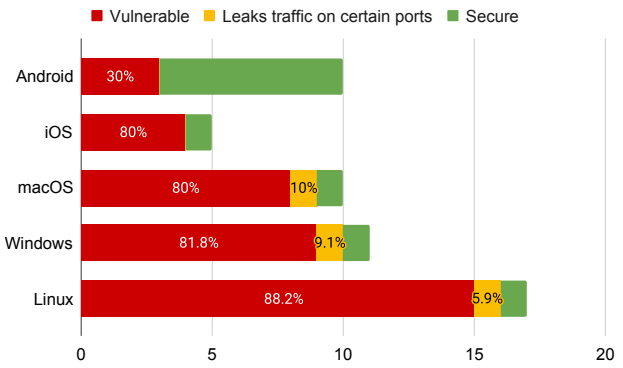

Figure 8: Number of 3rd-party and built-in VPN clients affected by (variants of) the ServerIP attacks for each tested OS.

be further described in Observation 5.6.1 below. Motivated by this, we shifted our focus to testing 3rd-party clients and OS built-in VPN clients. Since fewer such clients exist compared to custom ones, it becomes important to test various external VPN servers and profiles.

We collected usable VPN profiles that allowed us to establish a VPN tunnel and we searched for profile templates that we analyzed without connecting to the VPN server. Usable templates were gathered from websites that provide free profiles (e. g., VPNBook and VPN Gate) and from paid providers (e. g., StrongVPN and Surfshark). Profile templates, which do not contain credentials to connect to the server, were gathered by looking through public GitHub repositories that contain mirrors of VPN profiles (e. g., M-VPN and NordVPN), and by searching for VPN setup guides on university and company websites. We also found VPN setup instruction pages of universities where screenshots contained enough information for our analysis, i. e., whether hostnames were used in the profile (Observation 5.6.3). In total, this resulted in 21 profiles and profile templates that were analyzed, covering WireGuard configs, OpenVPN profiles, L2TP/IPsec configurations for built-in VPNs, and so on (see Table 2).

We evaluated a combination of 3rd-party and built-in VPN clients using the collected usable profiles, with the results summarized in Figure 8 (see Table 2 in the Appendix for details). Since our attack is only possible when using a hostname to identify the server, these results exclude configurations with hardcoded IP addresses, which we separately discuss in Observations 5.6.3 and 5.6.7. In total, we tested 17 VPN apps, some of which are only available on certain OSs: we tested 10 Android apps, 5 iOS apps, 11 Windows apps, 10 macOS apps, and 17 Linux apps, resulting in 53 experiments in total.

Against 73.6% of tested VPN clients when using a profile with hostnames, ServerIP attacks can be abused to leak arbitrary traffic. Further notable observations are:

5.6.1 Two VPN providers' custom clients are vulnerable. Windscribe on iOS when using IKEv2, and Avira Phantom VPN on macOS and iOS, use plaintext DNS to find the VPN server's IP address and as a result can be attacked to leak arbitrary traffic. All others do not use plaintext DNS and are therefore secure. We conjecture that most clients either use a proprietary protocol to get the server's IP address or that they use a hardcoded list of VPN servers, where the custom client must be updated if this list changes.

Although we did not further test these custom VPN clients and excluded them from Figure 8 and Table 2, they nevertheless constitute a significant portion of all VPN apps on the market. Additionally, even if an adversary is unable to spoof the VPN server's IP address against these clients, they may steal leak traffic that is sent to the real VPN server's IP address on certain OSs (see Observation 5.6.7).

5.6.2 The built-in IKEv2 API on iOS is vulnerable. During the disclosure, we found that Windscribe was vulnerable to the ServerIP attacks on iOS because it by default used the built-in `NEVPNProtocolIKEv2` of iOS. This means iOS itself causes both the plaintext DNS requests and the traffic leaks to the VPN server's IP address. To avoid plaintext DNS requests, the VPN server's IP address can be given to `NEVPNProtocolIKEv2` while attributes such as `serverCertificateCommonName` can be used to still securely verify the VPN server's identity.

5.6.3 76% of tested VPN profiles use hostname(s) instead of static IP address. When using a static (hardcoded) IP address for the VPN server, an adversary cannot perform ServerIP attacks to leak arbitrary traffic. Unfortunately, using static IPs is uncommon in practice: out of the 21 profiles we collected, only 5 use static IP addresses. This matches the official setup guide for OpenVPN which recommends using hostnames [45], since it has the advantage that the server's IP address can easily be changed. This is useful when updating servers or when an IP address gets blocked by governments. This means that most profiles, when used with an insecure VPN client, can be attacked to leak arbitrary traffic.

5.6.4 The OS influences the security of VPN clients. The observation as made in 5.4.1 also applies to the ServerIP attacks: there is a correlation between the OS and the vulnerability of a 3rd-party client. Most noticeable is that on Android only built-in VPNs were vulnerable. The situation is more serious on other platforms: on Windows, Linux, macOS, and Android, only WireGuard was secure.

Similar to Observation 5.4.1, we found that the same client may behave differently depending on the OS. For instance, Cisco AnyConnect has four different behaviors over five platforms (see Table 2) and was only fully secure on Android.

5.6.5 Cisco AnyConnect only leaks traffic on selected ports. For Cisco AnyConnect on Windows, macOS, and Linux, the attack was partially successful: only packets on certain ports were leaked (symbol ◐ in Table 2 and category "Leaks" in Figure 8). In all cases, TCP or UDP packets with the same destination port as the VPN tunnel were sent outside

the tunnel. For instance, when the VPN tunnel was using the DTLS protocol on port 443, all TCP and UDP traffic to port 443 is leaked outside the tunnel. Depending on the VPN server being used, traffic to ports 80 and 8000 was also leaked outside the tunnel. Traffic to other ports was being blocked, which we also consider a security issue (recall Section 4.2.3).

5.6.6 The behavior of two clients depended on the chosen VPN server. For Cisco AnyConnect on macOS and Windows, the ports on which traffic is leaked depend on the chosen server. For instance, when connected to a server in the UAE, only traffic on port 443 was leaked, while when connected to a server in the USA, traffic on ports 80, 443, and 8000 was leaked. Additionally, we observed that Windscribe always uses plaintext DNS to look up the VPN server's IP address when using a 3rd-party OpenVPN profile.

5.6.7 Static server IP addresses are still a privacy risk. Using static IPs for the VPN server in OpenVPN profiles prevents leakage of arbitrary traffic. However, all tested OpenVPN clients, on all OSs except Android, leak traffic to the VPN server's IP address and are therefore still affected by the privacy risks discussed in Section 4.2.1.

5.6.8 Some apps use proprietary protocols or software updates to get the server's IP address. We found that X-VPN, Best VPN Proxy Master, and Cisco AnyConnect use a proprietary protocol to obtain the VPN server's IP address. These apps first communicate with a master server and then switch to another server for the remainder of the tunnel's lifetime. The other apps appear to maintain a list of VPN servers and can immediately connect to a desired VPN server. This list is likely kept fresh through regular software updates.

5.6.9 Android's built-in VPN is more vulnerable than its VPN apps. From Table 2 we see that Android is only vulnerable to ServerIP attacks when using its built-in VPN. Namely, Android 8.1.0 is vulnerable when using IPsec Xauth but not when using L2TP/IPsec. We also confirmed this on a Galaxy S3 running Android 7. By inspecting the Android Open Source Project code, we believe this is the case for all Android versions below Android 12 (see Listing 2 in the Appendix). Since Android 12, these legacy VPN protocols are no longer supported by the official Android code base, meaning Android 12 and above are not vulnerable.

Our Samsung Galaxy Note 10+ 5G still supports legacy VPN protocols and was vulnerable when using L2TP/IPsec. By reverse engineering its firmware, we found that they modified Android's VPN code, causing it to be vulnerable for both LT2TP/IPsec and IPsec Xauth even when using Android 12.

6 Countermeasures

In this section, we give an overview of countermeasures, discuss their effectiveness, and test selected defenses in practice.

6.1 Countermeasures to LocalNet Attacks

6.1.1 Disable local traffic. The most straightforward and complete defense against the LocalNet attacks is disabling local traffic by default. However, not all VPN clients offer such functionality. For the clients that did support disabling local traffic, and that implemented this correctly, we confirmed that this prevented the attack. The downside of this defense is that legitimate use of the local network, such as accessing printers or streaming videos to a TV, no longer works while using the VPN. This downside can be lessened by still allowing local traffic when connected to a trusted network. For instance, when connected to a trusted protected Wi-Fi network, such as one's company or home network, local traffic can still be allowed. Another option is implementing a variant of policy-based routing where only specific applications are allowed to access the local network (see also Section 6.2.1).

6.1.2 Filtering excluded local IPs. An alternative defense, in case access to the local network while using the VPN is essential by default, is to only allow direct access to non-routable IP addresses. In particular, RFC 1918 defines valid IP ranges for private (local) networks [42], and only these ranges should ever be excluded from the VPN tunnel. These non-routable IP addresses are never assigned to public servers. As a result, if only RFC 1918 IPv4 addresses are ever excluded from the VPN tunnel, an adversary cannot trick the victim into excluding the IP address of any public server. Unfortunately, this defense is not complete. In particular, if the VPN is used in a split-tunnel setting (Section 2.1) to gain remote access to the private (local) network of an organization, an adversary can still perform the attack to leak traffic toward company services hosted in the private network.

6.2 Countermeasures to ServerIP Attacks

6.2.1 Policy-based routing. A defense against ServerIP attacks is policy-based routing, where routing is not purely based on the destination IP address, but also on other factors such as the application generating the traffic. In particular, a complete defense is to use policy-based routing to send the traffic of all applications, except the VPN client, through the VPN tunnel. On Linux, this can be achieved by defining multiple routing tables, where the routing table being used depends on which user generated the traffic or on the `fwmark` assigned to the packet. The VPN client can then be started under a unique user ID or it can assign a unique `fwmark` to VPN packets. If the traffic is generated by the VPN app, then a routing table is used that allows direct access to the Internet. However, all other apps use a routing table that redirects all traffic, independent of the destination IP address, to the VPN tunnel. This would prevent the ServerIP attacks, since the traffic of all other apps is always sent through the VPN tunnel. Policy-based routing was introduced in Android 4.4 and user-based routing is available on Linux since kernel 4.4 [41]. We

confirmed that Android 13 properly implements this approach and thus prevents the attack.

We remark that Tor uses a method resembling policy-based routing. To tunnel client traffic, Tor sets up a local proxy that client applications must be explicitly configured to use, instead of using a virtual network interface and modified routing tables. Application-to-proxy communications happen through the local loopback interface, whereas connections between the local proxy and the Tor network happen through the Tor protocol. DNS queries are performed directly through Tor, bypassing any locally configured DNS server. This configuration prevents all the attacks presented in this paper.

6.2.2 Verifying the server IP address. Another mitigation is to verify the VPN server's IP address once the client has connected to the server. For instance, after the client established a secure connection with the VPN server, the server can send its public IP over this secure connection. If the provided IP address does not match the IP address that the client is using for the server, the VPN connection is terminated, and the client can then try to reconnect to the real IP address. This prevents an adversary from making the client exclude an arbitrary IP address from the VPN tunnel, preventing unintended leakage. However, this does not prevent deanonymization attacks where the victim is tricked into initiating connections in plaintext with the VPN server itself (recall Section 4.2.1).

6.2.3 Authenticated DNS. To leak arbitrary traffic in the ServerIP attacks, the adversary needs to spoof DNS responses. The impact of the attack can therefore be lowered by using DNS over TLS or DNS over HTTPS [27,29]. These protocols encrypt and authenticate DNS requests by encapsulating them inside the TLS or HTTPS protocol. Alternatively, DNSSEC could be used, which authenticates all DNS replies, meaning an adversary cannot modify or spoof them [52]. When using such a protected variant of DNS, it is essential that the client does not fall back to plaintext unauthenticated DNS when protected DNS requests are failing. Otherwise, an adversary could attempt to block protected DNS requests, and then spoof DNS responses when the client falls back to using plaintext unauthenticated DNS. Unfortunately, using protected DNS does not prevent deanonymization attacks where the victim is tricked into initiating connections in plaintext with the VPN server itself (recall Section 4.2.1).

7 Discussion

In this section, we discuss the impact and limitations of the presented attacks and cover ethical considerations.

7.1 Impact and Practical Consequences

Both our attacks can leak traffic outside the VPN tunnel, which can be abused to intercept traffic, deanonymize users, modify webpages to serve malware, redirect users to phishing websites, and so on. Even if the victim is using another layer of encryption such as HTTPS, attacks may remain possible. For instance, a leaked server IP address is often enough to identify the website being visited [46]. More advanced HTTPS fingerprinting techniques can also identify the webpage being visited [40]. More worrisome, attacks like SSLstrip can remove HTTPS protection altogether [38], at least if HTTP Strict Transport Security (HSTS) is not being used, which currently only 25% of websites employ [59].

We confirmed the feasibility of monitoring HTTPS traffic for the LocalNet attacks. To redirect the leaked HTTPS traffic to the real server, we enabled proxy ARP on the AP, and added a route on the AP to forward traffic to the server through the default gateway of the network. Using this configuration, vulnerable victims leaked traffic to the server outside the VPN tunnel while still being able to load the website. An adversary can then attempt the previously-mentioned HTTPS attacks.

To intercept most IP-based traffic in the ServerIP attacks, an adversary can try to intercept traffic to the DNS server set by the VPN client, so that DNS replies can then be spoofed. We confirmed the feasibility of this against a macOS client running OpenVPN, where the VPN server was configured to advertise a single OpenDNS server.

7.2 Applicability against IPv6

We only targeted IPv4 traffic because it is still more commonly used than IPv6, e. g., at the time of writing roughly 60% of traffic to Google is using IPv4 [25]. Additionally, Ramesh et al. showed in 2022 that only 11 out of 80 tested VPN providers support IPv6 [49, §6.A]. We also experienced that even if a VPN supports IPv6, it may not be reliable. For instance, hide.me can connect over IPv6, but not all its servers support this, and connectivity was unreliable when using IPv6.

Focusing only on IPv4 does not limit an attacker. A local attacker can block all IPv6 packets, and thereby force the victim to use IPv4 to subsequently perform the attacks. Similarly, the on-path attacker can modify DNS responses to only return IPv4 addresses, meaning our attacks also remain possible even if the victim supports or uses IPv6.

To test whether our attacks also work against IPv6, we configured an OpenVPN server that is reachable over IPv6 and that offers IPv6 connectivity once the VPN tunnel is established. By using v0.6.7 of `linux-router` [61] we confirmed that both LocalNet attacks and ServerIP attacks remain possible when using IPv6. Once IPv6 support is more widely adopted by VPN apps, it will be interesting to evaluate our attacks against them when the VPN is using IPv6.

7.3 Ethical Considerations

To avoid ethical issues, we collected data through a passive approach and carried out experiments in a closed environment. We neither attack any real VPN servers nor websites

on the public network. We registered users and subscribed to VPN services from corresponding official websites. That means we were the legitimate users and intended receivers of the VPN packets. We did not attempt to send unrelated or malicious packets outside our test environment. To the best of our knowledge, the data collection process did not violate any networks' Terms and Conditions that were presented to the users. None of the mentioned traffic or network monitoring is illegal, and we did not impact the public network.

We performed the experiments in a way that does not affect other legitimate VPN users. Specifically, we perform the two attacks in an environment with simulated victim users. For the two attacks, the phishing website is situated within our local network and cannot be reached by users outside of our test network. The target website for ServerIP attacks was running on DigitalOcean with a secret IP. Our experiments did not cause any levels of denial-of-service to a VPN server, nor did they attempt to modify or target real websites.

In our experiments, we created a new Wi-Fi AP and did not imitate an existing network. This AP had a name indicating it was not a usable network, such as "testnetwork", and was set up in our lab where we carried out these experiments.

8 Related Work

Recent research efforts paid considerable attention to VPN services, products, and ecosystems. In 2018, Zhang et al. conducted a thorough security analysis of 84 well-known OpenVPN-based Android applications on the Google Play market [65]. Meanwhile, an empirical analysis was conducted on the VPN ecosystem, including the economical and technical influences of the VPN services [35]. In 2019, Bui et al. figured out that VPN clients had various configuration flaws in the cryptographic protocol of VPNs [10]. While in 2020, Wilson et al. investigated VPN applications for iOS devices, concerned with the applications for security and privacy issues [62]. A recent study published in 2022 developed a tool called VPNalyzer that could be used to systematically assess desktop VPN services and investigate the VPN ecosystem [49]. This tool can detect leakage of certain plaintext packets outside the VPN tunnel, but does not investigate IP routing manipulations, meaning it is unable to discover our novel attacks. Additionally, a quantitative survey of 1,252 VPN users and qualitative interviews of nine providers was conducted, aiming to improve the VPN ecosystem [50] while Streum et al. focus on evaluating DoS resilience for three major open-source VPN solutions [55]. Researchers also looked for implementation flaws in VPNs. For instance, Daniel et al. [15] searched for vulnerabilities in the OpenVPN state machine. Collectively, these studies outline and emphasize critical security issues for VPN applications within the VPN ecosystem.

Several works have studied the accidental leakage of DNS or IPv6 packets. Khan et al. [35] analyzed a broad set of VPN providers to study various aspects, including DNS and IPv6 leakage and the leakage of traffic when the VPN tunnel is temporarily disconnected. Wangchuk et al. [60] analyze DNS leaks, App permissions, and encryption usage of Android VPNs. Ikram et al. [30] analyze Android apps for DNS and IPv6 leakage and other things such as JavaScript injection. Fazal et al. [22] describe a risk when using hosts with both a Wi-Fi and Ethernet connection. There is also an RFC that describes how dual-stack IPv4 and IPv6 hosts might leak traffic [24]. Researchers also studied leaks before the VPN tunnel was established and how to deal with captive portals in Wi-Fi networks [11]. Other work abuses flaws in NAT functionality and the weak host model of Linux that manipulate TCP connections inside the VPN tunnel [56], or abuse ICMP messages to perform denial-of-service attacks [32, 51].

Many recent studies focus on specific platforms. For instance, VPNalyzer [49] is designed for investigating desktop VPNs, and the authors in [10] only analyze the vulnerabilities of desktop VPNs. As for mobile platforms, some research merely deals with iOS [62], while others only concentrate on Android [30, 65]. In contrast, our research covers both mobile and desktop platforms, as well as mainstream OSs, including Windows, Linux, macOS, iOS, and Android.

Ramesh et al. pointed out that some VPNs enabled access to the default gateway, which was considered a deanonymization risk [49], and may imply allowing access to the whole local network. Closer to our work, the authors in [47] manipulated a victim's routing table to leak DNS requests outside the VPN tunnel. This was done by using the gateway option of DHCP responses and by assigning a small bogus subnet to the victim that included the DNS server's IP address. They studied OpenVPN and PPTP/L2TP, while we cover all existing VPN protocols. Additionally, our LocalNet attacks can leak traffic towards *any* IP address, even if the victim does not use DNS, and our attack works even when the victim is using an authenticated DNS protocol such as DNSSec or DoH.

9 Conclusion

We discovered two novel attacks against VPN clients that enable an attacker to manipulate a victim's routing table in order to leak packets outside the VPN tunnel. We conjecture one reason why these routing manipulation vulnerabilities were only discovered more than two decades after the introduction of VPNs, is that most previous works focused on the VPN protocols themselves, but only few studied their integration into real-world clients and platforms.

We found that the security of VPN clients often depends on the OS, meaning one must always take into account the OS when discussing the security of a VPN client. We also believe that the OS should provide a straightforward and secure API to configure and start VPN clients. This avoids each VPN having to implement its own firewall or routing table code on each supported OS, which should increase overall security.

Acknowledgments

This research is partially supported by the Center for Cyber Security at NYU Abu Dhabi; the Research Fund KU Leuven, and by the Flemish Research Programme Cybersecurity.

References

[1] Benchmarking Methodology for Network Interconnect Devices. RFC 2544, March 1999.

[2] https://github.com/vanhoefm/vpnleaks, 2023.

[3] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. Imperfect forward secrecy: How diffie-hellman fails in practice. In *ACM CCS*, 2015.

[4] Omer Akgul, Richard Roberts, Moses Namara, Dave Levin, and Michelle L Mazurek. Investigating influencer VPN ads on YouTube. In *IEEE S&P*. IEEE, 2022.

[5] Suzan Ali, Mounir Elgharabawy, Quentin Duchaussoy, Mohammad Mannan, and Amr Youssef. Betrayed by the guardian: Security and privacy risks of parental control solutions. In *ACSAC*, 2020.

[6] Jacob Appelbaum, Marsh Ray, Karl Koscher, and Ian Finder. vpwns: Virtual pwned networks. In *USENIX FOCI*, 2012.

[7] Apple. macOS catalina 10.15 release notes. developer.apple.com/documentation/macos-release-notes/macos-catalina-10_15-release-notes. Accessed May 29, 2023.

[8] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-) security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In *ACM CCS*, 2016.

[9] Karthikeyan Bhargavan and Gaëtan Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. In *NDSS*, 2016.

[10] Thanh Bui, Siddharth Rao, Markku Antikainen, and Tuomas Aura. Client-side vulnerabilities in commercial VPNs. In *NordSec*, 2019.

[11] Christian Burkert, Johanna Ansohn McDougall, Hannes Federrath, and Mathias Fischer. Analysing leakage during VPN establishment in public Wi-Fi networks. In *IEEE ICC*, 2021.

[12] Cisco. Cisco anyconnect. https://www.cisco.com/c/en/us/support/security/secure-client-5/model.html/. Accessed September 16, 2022.

[13] Lorenzo Colitti. How the linux networking stack is made to work on android devices. In *Netdev Conf*, 2016.

[14] Michelle Cotton and Leo Vegoda. Special Use IPv4 Addresses. RFC 5735, January 2010.

[15] Lesly-Ann Daniel, Erik Poll, and Joeri de Ruiter. Inferring OpenVPN state machines using protocol state fuzzing. In *EuroS&P Workshops*, 2018.

[16] Jean Paul Degabriele and Kenneth G Paterson. Attacking the IPsec standards in encryption-only configurations. In *IEEE S&P*. IEEE, 2007.

[17] Jean Paul Degabriele and Kenneth G Paterson. On the (in) security of IPsec in MAC-then-encrypt configurations. In *ACM CCS*, 2010.

[18] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Trans. on information theory*, 1983.

[19] Jason A Donenfeld. WireGuard: next generation kernel network tunnel. In *NDSS*, 2017.

[20] Agnieszka Dutkowska-Zuk, Austin Hounsel, Amy Morrill, Andre Xiong, Marshini Chetty, and Nick Feamster. How and why people use virtual private networks. In *USENIX Security*, 2022.

[21] Kevin R Fall and W Richard Stevens. *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.

[22] Lookman Fazal, Sachin Ganu, Martin Kappes, A. S. Krishnakumar, and Parameshwaran Krishnan. Tackling security vulnerabilities in VPN-based wireless deployments. *IEEE ICC*, 2004.

[23] Dennis Felsch, Martin Grothe, Jörg Schwenk, Adam Czubak, and Marcin Szymanek. The dangers of key reuse: Practical attacks on IPsec IKE. In *USENIX Security*, 2018.

[24] Fernando Gont. Layer 3 Virtual Private Network (VPN) Tunnel Traffic Leakages in Dual-Stack Hosts/Networks. RFC 7359, August 2014.

[25] Google. IPv6 statistics. Retrieved 6 February 2023 from google.com/intl/en/ipv6/statistics.html, 2023.

[26] Kory Hamzeh, Grueep Pall, William Verthein, Jeff Taarud, W Little, and Glen Zorn. Point-to-Point Tunneling Protocol (PPTP). RFC 2637, 1999.

[27] Paul E. Hoffman and Patrick McManus. DNS Queries over HTTPS (DoH). RFC 8484, October 2018.

[28] Michael Horowitz. VPNs on iOS are a scam. https://www.michaelhorowitz.com/VPNs.on.iOS.are.scam.php, 2022. Accessed May 23, 2023.

[29] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul E. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, May 2016.

[30] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kâafar, and Vern Paxson. An analysis of the privacy and security risks of android VPN permission-enabled apps. In *ACM IMC*, 2016.

[31] Psiphon Inc. psiphon-ios-vpn. https://github.com/Psiphon-Inc/psiphon-ios-vpn/blob/332v1.1.20/PsiApi/Sources/PsiApi/TunnelProviderManager.swift#L316, 2023. Accessed May 23, 2023.

[32] Ludovic Jacquin, Vincent Roca, and Jean-Louis Roch. Too big or too small? the PTB-PTS ICMP-based attack against IPsec gateways. In *Global Communications Conference*, pages 530–536. IEEE, 2014.

[33] Yunhan Jack Jia, Qi Alfred Chen, Yikai Lin, Chao Kong, and Z Morley Mao. Open doors for bob and mallory: Open port usage in android apps and security implications. In *EuroS&P*. IEEE, 2017.

[34] Charlie Kaufman, Paul Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. Internet key exchange protocol version 2 (IKEv2). Technical report, RFC 7296, 2014.

[35] Mohammad Taha Khan, Joe DeBlasio, Geoffrey M. Voelker, Alex C. Snoeren, Chris Kanich, and Narseo Vallina-Rodriguez. An empirical analysis of the commercial VPN ecosystem. In *ACM IMC*. ACM, 2018.

[36] Yiannis M. create_ap. https://github.com/oblique/create_ap, 2013. Accessed September 12, 2022.

[37] Aanchal Malhotra, Isaac E. Cohen, Erik Brakke, and Sharon Goldberg. Attacking the network time protocol. In *NDSS*, 2016.

[38] Moxie Marlinspike. New tricks for defeating SSL in practice. In *Black Hat Briefings*, 2009.

[39] Microsoft. Secure Socket Tunneling Protocol (SSTP). https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-sstp/. Accessed September 16, 2022.

[40] Brad Miller, Ling Huang, Anthony D Joseph, and J Doug Tygar. I know why you went to the clinic: Risks and realization of HTTPS traffic analysis. In *PETS*, 2014.

[41] David S. Miller. Merge branch 'uid-routing'. Linux commit `4fb74506838b`, retrieved 2 December 2022 from https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=4fb74506838b, 2016.

[42] Robert Moskowitz, Daniel Karrenberg, Yakov Rekhter, Eliot Lear, and Geert Jan de Groot. Address Allocation for Private Internets. RFC 1918, February 1996.

[43] Moses Namara, Daricia Wilkinson, Kelly Caine, and Bart P Knijnenburg. Emotional and practical considerations towards the adoption and abandonment of VPNs as a privacy-enhancing technology. *PETS*, 2020(1), 2020.

[44] OpenVPN. Openvpn. https://github.com/OpenVPN. Accessed September 16, 2022.

[45] OpenVPN. Setting up your OpenVPN access server hostname. Retrieved 6 February 2023 from https://openvpn.net/vpn-server-resources/setting-up-your-openvpn-access-server-hostname/, 2023.

[46] Simran Patil and Nikita Borisov. What can you learn from an IP? In *ANRW*, 2019.

[47] Vasile C Perta, Marco V Barbera, Gareth Tyson, Hamed Haddadi, and Alessandro Mei. A glance through the VPN looking glass: IPv6 leakage and DNS hijacking in commercial VPN clients. *PETS*, 2015.

[48] Amit Pundir. The state of AOSP common android-4.4 kernel. In *Linaro Connect Las Vegas*, 2016.

[49] Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi. VPNalyzer: Systematic investigation of the VPN ecosystem. In *NDSS*, 2022.

[50] Reethika Ramesh, Anjali Vyas, and Roya Ensafi. All of them claim to be the best: Multi-perspective study of VPN users and VPN providers. In *USENIX Security*, 2023.

[51] Vincent Roca, Ludovic Jacquin, Saikou Fall, and Jean-Louis Roch. New results for the PTB-PTS attack on tunneling gateways. In *GreHack*, 2015.

[52] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. DNS Security Introduction and Requirements. RFC 4033, March 2005.

[53] Bruce Schneier and Mudge. Cryptanalysis of microsoft's point-to-point tunneling protocol (PPTP). In *ACM CCS*. ACM, 1998.

[54] Charlie Scott, Paul Wolfe, and Mike Erwin. *Virtual private networks*. O'Reilly Media, Inc., 1999.

[55] Fabio Streun, Joel Wanner, and Adrian Perrig. Evaluating susceptibility of VPN implementations to DoS attacks using adversarial testing. In *NDSS*, 2022.

[56] William J. Tolley, Beau Kujath, Mohammad Taha Khan, Narseo Vallina-Rodriguez, and Jedidiah R. Crandall. Blind In/On-Path attacks and applications to VPNs. In *USENIX Security 21*. USENIX Association, 2021.

[57] Tunnelblick. Tunnelblick | Free open source OpenVPN VPN client server software for macOS. https://tunnelblick.net, 2022. Accessed September 16, 2022.

[58] Ivana Vojinovic. VPN statistics for 2022 – keeping your browsing habits private, 2022.

[59] W3Techs. Usage statistics of HTTP strict transport security for websites. Retrieved 26 January 2023 from https://w3techs.com/technologies/details/ce-hsts and https://archive.is/EilrO, 2023.

[60] Tashi Wangchuk, Digvijaysingh Rathod, et al. Forensic and behavior analysis of free android VPNs. *JAETM*, 1(1):91–101, 2021.

[61] Gary Will. create_ap. https://github.com/garywill/linux-router, 2023. Accessed May 25, 2023.

[62] Jack Wilson, David McLuskie, and Ethan Bayne. Investigation into the security and privacy of iOS VPN applications. In *ARES*, 2020.

[63] Windscribe. Windscribe: Browse the web privately as it was meant to be. https://github.com/windscribe, 2022. Accessed September 16, 2022.

[64] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J Alex Halderman, Jedidiah R Crandall, and Roya Ensafi. OpenVPN is open to VPN fingerprinting. In *USENIX Security*, 2022.

[65] Qi Zhang, Juanru Li, Yuanyuan Zhang, Hui Wang, and Dawu Gu. Oh-pwn-VPN! security analysis of OpenVPN-based android apps. In *CANS*, 2017.

Appendix

Listing 1: Psiphon code controlling local network access [31]. It is not vulnerable to LocalNet attacks on iOS 14.1 and 14.2.

```
if #available(iOS 14.0, *) {
    self.wrappedManager.protocolConfiguration!.includeAllNetworks = true
}
if #available(iOS 14.2, *) {
    // Excludes local network on both iOS and macOS running on Apple Silicon.
    self.wrappedManager.protocolConfiguration!.excludeLocalNetworks = true
}
```

Listing 2: Android's `LegacyVpn` class adds an exception for the VPN server's IP address. Variable `endpointAddress` is empty when using L2TP, causing Android's built-in VPN to only be affected by ServerIP attacks when using IPsec Xauth.

```
// Add a throw route for the VPN server endpoint
if (endpointAddress instanceof Inet4Address) {
    mConfig.routes.add(new RouteInfo(
        new IpPrefix(endpointAddress, 32), null /*gateway*/,
        null /* iface */, RTN_THROW));
}
```

Table 1: Results for LocalNet attacks on different VPN clients.

VPN Provider	Class	OS	Version Number	LAN Setting | Default LAN Access	Result
OS Built-in VPN	Free	Windows	Windows 10 Pro	No | N/A	✘
	Free	Windows	Windows 11 Pro	No | N/A	✘
	Free	macOS	Ventura 13.0.1	No | N/A	✘
	Free	iOS	iOS 16.1.1	No | N/A	✘
	Free	Android	Android 8.1.0	No | N/A	✔
	Free	Android	Android 12	No | N/A	✔
1.1.1.1	Free	Windows	2022.10.106.0	No | N/A	✔
	Free	Linux	2022.9.591	No | N/A	✔
	Free	macOS	2022.10.107.0	No | N/A	△
	Free	iOS	6.16	No | N/A	✘
	Free	Android	6.17	No | N/A	✔
Any VPN	Free	Windows	1.0.8	No | N/A	✘
Avira Phantom	Free	Windows	1.1.79.5	No | N/A	△
Best VPN	Free	macOS	1.9.1	No | N/A	✘
Betternet	Paid	Windows	7.2.1	No | N/A	✘
	Paid	macOS	2.10.0	No | N/A	✘
	Paid	iOS	5.31.0	No | N/A	✘
	Paid	Android	6.2.0	No | N/A	✔
Cisco AnyConnect	Free	Windows	4.10.04065	Yes | No	✔
	Free	Linux	4.10.05095	Yes | No	✔
	Free	macOS	4.10.04065	Yes | No	✔
	Free	iOS	5.0.00246	No | N/A	✘
	Free	Android	5.0.00247	No | N/A	✔
Clario VPN	Paid	Windows	5.9.1.1662	Yes | No	✘
	Paid	macOS	5.9.1.1662	Yes | No	✘
	Paid	iOS	1.9.28	Yes | Yes	✘
	Paid	Android	1.9.26.420979	Yes | No	✘
CyberGhost	Paid	Windows	8.3.7.9795	No | N/A	△
	Paid	Linux	1.3.4	No | N/A	✘
	Paid	macOS	8.3.9.167	No | N/A	✘
	Paid	iOS	8.4.0	No | N/A	✘
	Paid	Android	8.6.17.1164	No | N/A	✔
ExpressVPN	Paid	Windows	12.37.0	Yes | Yes	△
	Paid	Linux	3.36	No | N/A	△
	Paid	macOS	11.12.0	Yes | Yes	△
	Paid	iOS	11.70.0	Yes | Yes	✘
	Paid	Android	10.63.2	Yes | Yes	✔
Fast VPN-solo unlimited proxy	Free	iOS	2.2.4	Yes | Yes	✘
Fast VPN by Namecheap	Paid	Windows	3.4.0.0	No | N/A	✘
	Paid	macOS	3.9.0	No | N/A	✘
	Paid	iOS	3.7.0	No | N/A	✘
	Paid	Android	3.4.2	No | N/A	✔
Free VPN by Free VPN.org	Free	macOS	3.884	No | N/A	✘
	Free	iOS	3.879	No | N/A	✘
	Free	Android	3.9	No | N/A	✔
Goose VPN	Paid	Windows	5.0.0.9	No | N/A	✘
	Paid	macOS	5.1.3	No | N/A	✘
	Paid	iOS	5.1.4	No | N/A	✘
	Paid	Android	5.1.4(195)	No | N/A	✔
Hide.me VPN	Free / Paid	Windows	3.14.0	Yes |No	△
	Free / Paid	Linux	0.9.2	No | N/A	△
	Free / Paid	macOS	4.7.1	No | N/A	✘
	Free / Paid	iOS	4.11.0	No | N/A	✘
	Free / Paid	Android	3.10.0	Yes | Yes	✘
HideMyAss	Paid	Windows	5.21.6744	No | N/A	✘
	Paid	Linux	0.5	No | N/A	✔
	Paid	macOS	5.4.7	No | N/A	✘
	Paid	iOS	5.10.0	No | N/A	✘
	Paid	Android	5.68.6510	No | N/A	✘
Hotspot Shield	Free	Windows	2.10.8	No | N/A	✘
	Free	macOS	5.3.0b1028	No | N/A	✘
	Free	iOS	7.9.0	No | N/A	✘
	Free	Android	9.9.0	No | N/A	✔
IPVanish	Paid	Windows	4.1.2.122	Yes | Yes	✘
	Paid	macOS	3.3.0(67479)	No | N/A	✘
	Paid	iOS	4.4.0	No | N/A	✘
	Paid	Android	4.0.3.0	Yes | No	✔
Mullvad	Paid	Windows	2022.5.0	Yes | No	△
	Paid	macOS	2022.5	Yes | No	△
	Paid	Linux	2022.5	Yes | No	△
	Paid	iOS	2022.2	No | N/A	✘
	Paid	Android	2022.3	Yes | No	✔

Network Manager	Free	Linux	1.1.93	No | N/A	✘
OpenVPN Gnome	Free	Linux	1.8.2	No | N/A	✘
	Free	Linux	1.8.12	No | N/A	✔
	Free	Linux	1.8.18	No | N/A	✔
NordVPN	Paid	Windows	7.2.2.0	No | N/A	△
	Paid	Linux	1.0.0	No | N/A	✔
	Paid	macOS	7.12.2	No | N/A	✘
	Paid	iOS	7.21.1	No | N/A	✘
	Paid	Android	5.28.0	Yes | No	✔
OpenConnect CLI	Free	Linux	7.06	No | N/A	✘
	Free	Linux	7.08	No | N/A	✘
	Free	Linux	8.05	No | N/A	✔
	Free	Linux	8.20	No | N/A	✔
	Free	macOS	9.0.1	No | N/A	✘
OpenConnect GUI	Free	Windows	1.5.3	No | N/A	✘
	Free	macOS	1.5.3	No | N/A	✘
OpenVPN CLI	Free	Linux	2.3.10	No | N/A	✘
	Free	Linux	2.4.2	No | N/A	✘
	Free	Linux	2.4.7	No | N/A	✔
	Free	Linux	2.5.5	No | N/A	✔
OpenVPN Connect	Free	Windows	3.3.6	No | N/A	✘
	Free	macOS	3.4.0	No | N/A	✘
	Free	iOS	3.3.2	No | N/A	✘
	Free	Android	3.3.0	No | N/A	✔
openvpn3-linux	Free	Linux	openvpn3 19~beta1	No | N/A	✘
OpenVPN for Andriod	Free	Android	0.7.14	Yes | Yes	✗
OpenVPN GUI	Free	Windows	11.29.0.0	No | N/A	✘
ProtonVPN	Free	Android	4.3.80.2	Yes | No	✔
Psiphon	Free	Windows	176.20221118121227	No | N/A	✔
	Free	iOS	1.1.13	No | N/A	✘
	Free	Android	359	No | N/A	✔
PureVPN	Paid	Windows	10.0.0.2	No | N/A	✘
	Paid	Linux	2.2.2	No | N/A	✘
	Paid	macOS	8.9.1	No | N/A	✘
	Paid	iOS	9.7.2	No | N/A	✘
	Paid	Android	8.37.114	No | N/A	†
Secure VPN	Free	iOS	2.6.37	No | N/A	✘
	Free	Android	4.0.4	No | N/A	✔
ShadowX VPN	Free	iOS	2.2.0	No | N/A	✘
Shimo	Free	macOS	5.0.4	No | N/A	✘
SkyVPN	Free	Windows	0.9.27	No | N/A	✘
	Free	iOS	2.0.10	No | N/A	✘
	Free	Android	2.4.0	No | N/A	✘
SotfEther VPN	Free	Windows	4.41	No | N/A	✘
	Free	Linux	4.41	No | N/A	✘
Speedify	Free	Windows	12.6.0.10617	No | N/A	✘
	Free	macOS	12.7.0.1658	No | N/A	✘
	Free	iOS	12.6.0.11482	No | N/A	✘
	Free	Android	12.5.0.11584	No | N/A	✔
Star VPN	Free	Windows	1.5.4	No | N/A	✘
	Free	macOS	2.11.0	No | N/A	✘
	Free	iOS	3.6.0	No | N/A	✘
	Free	Android	1.8	No | N/A	✘
Strong VPN	Paid	Windows	2.6.2.0	No | N/A	✘
	Paid	macOS	2.2.2	No | N/A	✘
	Paid	iOS	2.6.0	No | N/A	✘
	Paid	Android	2.3.3.6	Yes | No	✔
Super VPN - Secure VPN Master	Free	iOS	3.2.6	No | N/A	†
	Free	Android	1.6.2	No | N/A	✔
TLS Tunnel	Free	Android	5.0.5	No | N/A	✔
TorGuard	Paid	Windows	4.8.13	No | N/A	✘
	Paid	Linux	4.8.13	No | N/A	✘
	Paid	macOS	4.8.13	No | N/A	✘
	Paid	iOS	2.2.8	No | N/A	✘
	Paid	Android	1.60.9	No | N/A	✘
TouchVPN	Free	Windows	2.0.2.274	No | N/A	✘
	Free	macOS	2.5.6.158	No | N/A	✘
	Free	iOS	4.4.1	No | N/A	✘
	Free	Android	2.0.8	No | N/A	✔
TunnelBear	Paid	Windows	4.6.1	No | N/A	△
	Paid	macOS	4.1.8	No | N/A	✘
	Paid	iOS	4.3.2	No | N/A	✘
	Paid	Android	3.6.8	No | N/A	✔
Tunnelblick	Free	macOS	3.8.7a	No | N/A	✘
Turbo VPN	Free	Windows	2.18.0.0	No | N/A	✘
	Free	macOS	1.8.1(44)	No | N/A	✘
	Free	iOS	2.9.11	No | N/A	✘
	Free	Android	3.8.7.1	No | N/A	✔
Turbo VPN Lite	Free	Android	1.1.8.1	No | N/A	✔
Urban VPN Desktop	Free	macOS	1.10.0	No | N/A	✘
VPN for iPhone·	Free	iOS	2.1.5	No | N/A	✘
VPN Go	Free	iOS	2.0.2	No | N/A	✘
	Free	Android	1.0.25	No | N/A	✘
VPN Hotspot	Free	macOS	1.4.1	No | N/A	✘
VPN PotatoVPN	Free	iOS	16.0.1	No | N/A	✘
VPN Proxy Master	Free & Paid	Windows	3.13.0.0	No | N/A	✘
	Free & Paid	macOS	2.15.0	No | N/A	✘
	Free & Paid	iOS	6.5.5	No | N/A	✘
	Free & Paid	Android	2.3.0.6	No | N/A	✔
VPN Proxy Master for iPhone	Free	iOS	2.1.5	No | N/A	†
VPN Shield	Free	macOS	1.1	No | N/A	✘
VPN Super Unlimited Proxy	Free	iOS	1.9.4	No | N/A	✘
	Free	Android	1.18.0	No | N/A	✔
VPN Surfshark	Paid	Windows	4.4.1	Yes | Yes	✗
	Paid	Linux	1.2.3	No | N/A	✔
	Paid	MacOS	4.5.0	No | N/A	✘
	Paid	iOS	3.6.3	No | N/A	✘
	Paid	Android	2.8.2.8	Yes | Yes	✗
VPN Vault	Free	iOS	3.48	No | N/A	✘
VPN··	Free	iOS	1.50	No | N/A	✘
VPN‘	Free	iOS	2.7.2	No | N/A	✘
VPN°	Free	iOS	3.15	No | N/A	✘
Viscosity	Free	Windows	1.10.4	No | N/A	✘
	Free	macOS	1.10.4	No | N/A	✘
VyprVPN	Paid	Windows	4.3.1.10763	No | N/A	✘
	Paid	macOS	4.1.0.8945	No | N/A	✘
	Paid	iOS	4.4.0.106	No | N/A	✘
	Paid	Android	4.5.2.113841	No | N/A	✔
Windscribe built-in	Free	Windows	2.5.17	Yes | No	△
	Free	Linux	2.5.17	Yes | No	✔
	Free	macOS	2.4.11	Yes | No	△
	Free	iOS	3.4.1(273)	Yes | Yes	✗
	Free	Android	3.3.1003	Yes | No	✔
Windscribe 3rd-party	Free	Linux	2.5.17	Yes | No	✔
WireGuard	Free	Windows	0.5.3	No | N/A	△
	Free	Linux	1.0.20210914	No | N/A	✔
	Free	macOS	1.0.15	No | N/A	✘
	Free	iOS	1.0.15	No | N/A	✘
	Free	Android	1.0.20220516	No | N/A	✔
XVPN	Free	Windows	73.0_2674	No | N/A	✔
	Free	macOS	73.1.0_2791	No | N/A	✘
	Free	iOS	31.3	No | N/A	✘
	Free	Android	180_2778	No | N/A	✔

✘ always vulnerable, ✗ vulnerable by default LAN-Access-Setting
† vulnerable by using special use IP addresses ✔ always secure,
✔secure by default LAN-Access-Setting, △ local traffic blocked

Table 2: Results of the ServerIP attacks on VPN clients that use plaintext DNS to get the VPN server's IP address.

VPN Client	VPN Service Provider	OS	Version Number	VPN Protocol	VPN Port	Result
OS Built-in VPN	VPN Gate	Windows	Windows 10 Pro	L2TP/IPsec	500 + 4500	✘
	VPN Gate	Windows	Windows 10 Pro	SSTP	443	✘
	VPN Gate	Windows	Windows 11 Pro	L2TP/IPsec	500 + 4500	✘
	VPN Gate	Windows	Windows 11 Pro	SSTP	443	✘
	VPN Gate	macOS	Ventura 13.0.1	L2TP/IPsec	500 + 4500	✘
	VPN Gate	iOS	iOS 16.1.1	L2TP/IPsec	500 + 4500	✘
	VPN Gate	Android	Android 8.1.0	L2TP/IPsec	500 + 4500	✔
	Personal Server	Android	Android 8.1.0	IPsec Xauth PSK	500 + 4500	✘
	Personal Server	Android	Android 9	IKEv2/IPsec	500 + 4500	✘
	VPN Gate	Android	Android 12	L2TP/IPsec	500 + 4500	✘
	Personal Server	Android	Android 13	IKEv2/IPsec	500 + 4500	✔
Cisco AnyConnect	University VPN	Windows	4.10.04065	DTLSv1.2	443	◐
	University VPN	Linux	4.10.05095	DTLSv1.2	443	◐
	University VPN	macOS	4.10.04065	DTLSv1.2	443	◐
	University VPN	iOS	5.0.00246	DTLSv1.2	443	✘
	University VPN	Android	5.0.00247	DTLSv1.2	443	✔
Network Manager OpenVPN Gnome	GooseVPN	Linux	1.1.93	OpenVPN	443	✘
	StrongVPN	Linux	1.8.2	OpenVPN	1194	✘
	StrongVPN	Linux	1.8.12	OpenVPN	1194	✘
	StrongVPN	Linux	1.8.18	OpenVPN	1194	✘
OpenConnect CLI	University VPN	Linux	7.06	Cisco AnyConnect	443	✘
	University VPN	Linux	7.08	Cisco AnyConnect	443	✘
	University VPN	Linux	8.05	Cisco AnyConnect	443	✘
	University VPN	Linux	8.20	Cisco AnyConnect	443	✘
	University VPN	macOS	12	Cisco AnyConnect	443	✘
OpenConnect GUI	University VPN	Windows	1.5.3	Cisco AnyConnect	443	✘
	University VPN	macOS	1.5.3	Cisco AnyConnect	443	✘
OpenVPN CLI	GooseVPN	Linux	2.3.10	OpenVPN	443	✘
	StrongVPN	Linux	2.4.4	OpenVPN	1194	✘
	StrongVPN	Linux	2.4.7	OpenVPN	1194	✘
	StrongVPN	Linux	2.5.5	OpenVPN	1194	✘
OpenVPN Connect	StrongVPN	Windows	3.3.6	OpenVPN	1194	✘
	StrongVPN	Linux	openvpn3 19∼beta1	OpenVPN	1194	✘
	StrongVPN	macOS	3.3.6	OpenVPN	1194	✘
	StrongVPN	iOS	3.3.2	OpenVPN	1194	✘
	StrongVPN	Android	3.3.0	OpenVPN	1194	✔
OpenVPN for Android	HideMyAss	Android	0.7.14	OpenVPN	553	✔
SoftEther VPN	VPN Gate	Windows	4.41	SoftEther	443	✘
	VPN Gate	Linux	4.41	SoftEther	443	✘
Shimo	University VPN	macOS	5.0.4	DTLSv1.2	443	✘
Tunnelblick	StrongVPN	macOS	3.8.7a	OpenVPN	1194	✘
Viscosity	StrongVPN	Windows	1.10.4	OpenVPN	443	✘
	StrongVPN	macOS	1.10.4	OpenVPN	443	✘
Windscribe 3rd-party	StrongVPN	Windows	2.5.17	OpenVPN	1194	✘
	StrongVPN	Linux	2.5.17	OpenVPN	1194	✘
	StrongVPN	macOS	2.4.11	OpenVPN	1194	✘
	GooseVPN	iOS	3.4.1	OpenVPN	443	✘
	GooseVPN	Android	3.4.1085	OpenVPN	443	✔
WireGuard	Surfshark	Windows	0.5.3	WireGuard	51820	✔
	Surfshark	Linux	1.0.20210914	WireGuard	51820	✔
	Surfshark	macOS	1.0.15	WireGuard	51820	✔
	Surfshark	iOS	1.0.15	WireGuard	51820	✔
	Surfshark	Android	1.0.20220516	WireGuard	51820	✔

✘ always vulnerable, ◐ partially vulnerable for some ports, ✔ always secure