Blind In/On-Path Attacks and Applications to VPNs

William J. Tolley and Beau Kujath, *Breakpointing Bad/Arizona State University;* Mohammad Taha Khan, *Washington and Lee University;* Narseo Vallina-Rodriguez, *IMDEA Networks Institute/ICSI;* Jedidiah R. Crandall, *Breakpointing Bad/ Arizona State University*

https://www.usenix.org/conference/usenixsecurity21/presentation/tolley

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the 30th USENIX Security Symposium is sponsored by USENIX.

Blind In/On-Path Attacks and Applications to VPNs

William J. Tolley*
Breakpointing Bad
Arizona State University

Beau Kujath
Breakpointing Bad
Arizona State University

Mohammad Taha Khan
Washington & Lee University

Narseo Vallina-Rodriguez
IMDEA Networks Institute
International Computer Science Institute

Jedidiah R. Crandall
Breakpointing Bad
Arizona State University

Abstract

Protecting network protocols within an encrypted tunnel, using technologies such as Virtual Private Networks (VPNs), is increasingly important to millions of users needing solutions to evade censorship or protect their traffic against in/on-path observers/attackers. In this paper, we present a series of attacks from two threat models: an attacker that can inject spoofed packets into the network stack of a VPN client (called client-side), and an attacker that can spoof packets on the Internet and send them to a VPN server (called server-side). In both cases, we assume that the attacker is in/on-path, and can count encrypted bytes or packets over time. In both threat models, we demonstrate attacks to infer the existence of, interfere with, or inject data into TCP connections forwarded through the encrypted VPN tunnel. In the server-side threat model, we also demonstrate an attack to hijack tunneled DNS queries and completely remove the protections of the VPN tunnel. For the attacks presented in this paper, we (1) assess their feasibility in terms of packet rates and timing; (2) test their applicability against a broad range of VPN technologies, types, and vendors; and (3) consider practical issues with respect to real-world attacks. We followed an ethical disclosure process for all attacks presented in this paper. Client-side attacks were addressed with two CVEs and partially mitigated by a series of updates from some operating system and VPN client vendors. Server-side attacks have not been addressed and are still feasible with all operating systems and VPN servers that we tested.

1 Introduction

Virtual Private Networks (VPNs), and other related technologies that form an encrypted tunnel for Internet traffic, have become pervasive security and privacy tools that are relied upon by a wide variety of users. As examples: government agencies use VPNs to help protect national secrets; at-risk users such as journalists and activists use tools that include VPNs, Lantern, Orbot, Psiphon, *etc.* [23] to protect free speech and free assembly; and everyday users use similar technologies to connect to the Internet *via* untrusted networks, or simply to remain private online. VPNs were originally developed to provide point-to-point access to remote resources, and later retrofitted to forward any traffic generated at higher layers in the network stack of a device running a VPN to a remote VPN server through an encrypted tunnel. But, what security and privacy guarantees do VPNs, as they are implemented today, actually provide?

In this paper, we present attacks on connections that are tunneled inside a VPN. Irrespective of VPNs, attacks on network connections have traditionally fallen into two categories: (1) **In/on-path attacks**, in which an attacker is part of the network infrastructure and routes the packets to/from the client and server so they can easily infer connections, count packets, and interfere with data streams; and (2) **Blind off-path attacks** in which side-channel inferences are necessary to carry out that attack because the attacker cannot see packets in transit to learn about values such as sequence numbers. We refer the reader to Marczak *et al.* [22] for a formal definition (and distinction) of in- *vs.* on-path[1]. Because our attacks are easier to implement as in-path rather than on-path (though both are possible), we sometimes use simply "in-path" throughout the rest of this paper.

Network protocols such as TCP or DNS contain secret randomized values to protect them from off-path attacks, *i.e.*, attackers who do not see communications going back and forth between client and server but attempt to interfere with, or infer information about, connections *via* side-channels in protocol implementations. For example, previous works have shown that off-path attackers can infer the existence of connections [3], count packets between end-points [18], or even interfere with or inject data into the data stream [11, 15]. To mitigate such attacks, the TCP protocol randomizes the ephemeral port chosen by a client making a connection request, and the initial sequence number is randomized by both

*Corresponding author: william@breakpointingbad.com

[1] Basically, on-path attackers can delay or drop packets while in-path attackers cannot.

client and server. For DNS, a protocol that is typically UDP-based, the ephemeral port of the client is randomized, and there is a random transaction ID (TXID) to protect against spoofed responses from off-path attackers.

For in-path attacks, session-layer encryption between the client and server, such as TLS [19] or DNS over HTTPS/TLS [8, 9, 28], can mitigate some attacks but they cannot protect metadata about the connection, and can be thwarted by an attacker with a forged certificate. Thus technologies such as VPNs are often used to add another layer of security and privacy to protect against in-path attackers.

In this paper, we propose and demonstrate a third category of attack against encrypted tunnels: **blind in/on-path attacks** where the fields necessary for the attack (*e.g.*, port numbers and sequence numbers) are encrypted and not directly visible to the in-path attacker. We use attacks on VPN tunnels as example applications to demonstrate blind in/on-path attacks. Figure 1 shows the differences between the different types of attacks in a scenario with standard TCP/UDP connections as well as in a VPN scenario. A key insight of our work is that encryption can hide contents for packets (specifically headers and data), but it cannot hide properties such as the number of packets, their size, and their timing. Thus the attacker is *"blind"* in the sense that they cannot directly see/modify tunneled headers and data, but can still infer headers and modify headers or data because of the same properties that make off-path attacks possible.

We show that the randomized values used for protection against off-path attacks can easily be inferred by a blind in/on-path attacker despite being sent through an encrypted tunnel. This can lead to a complete breakdown of the security and privacy of protocols such as TCP and DNS that is supposed to be added when they are tunneled inside a VPN or other VPN-like technology. The attacks presented in this paper challenge the current understanding of real-world VPN's security by showing that even a properly configured and secured VPN is still vulnerable to connection tampering from a malicious actor with the ability to access and control the gateway (including network adjacent attackers who have altered the victim's routing through ARP cache poisoning, for example) or any router between the VPN server and client. The attacks disclosed in this paper allow a malicious actor to determine if a person using a VPN is connected to a particular application server, and to subsequently reset or hijack any identified TCP/IP connections within the encrypted tunnel that are identified; or spoof responses to UDP-based DNS queries.

Our results include that for (as an example) Linux-based systems, a network adjacent attacker utilizing a client-side attack can infer and hijack or reset HTTP(S) connections of a given website 91.6% of the time in a real-world environment. In the case of DNS queries subjected to a server-side attack, we find that for a timeout of 5 seconds (the DNS lookup timeout for most modern browsers), the attack is successful 11.6% of the time, but the attack is successful 75.3% of the time for a timeout of 15 seconds (*i.e.*, the default DNS lookup timeout for Android).

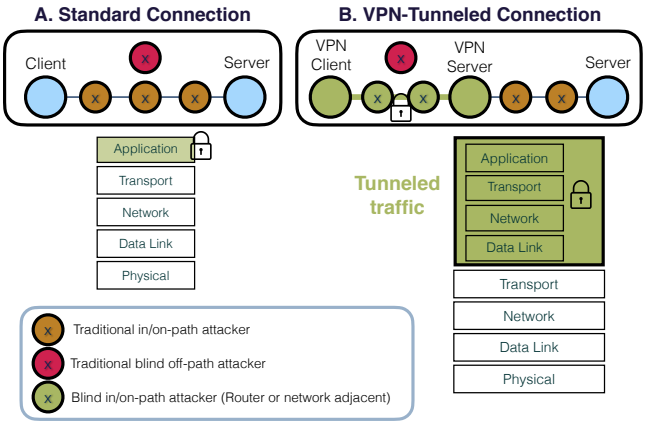

Figure 1: Outline of the different attacks and threats in a standard TCP/UDP connection and a VPN-tunneled TCP/UDP connection. In this paper we focus on *Blind in/on-path attacks* against traffic protected by encrypted VPN tunnels.

The rest of this paper is structured as follows. Section 2 introduces our attack assumptions and discusses our ethical disclosure process. Section 3 provides background information on VPN technologies, source address validation at different levels of the Internet, and the TCP flags that are used in our attacks. Section 4 provides an overview of our threat model, and the different phases of our attack. In Section 5 we enumerate the experiments we carried out for different types of attacks. Section 6 presents our findings and analysis. Sections 7, 8, and 9 provide a discussion, related work, and conclusion, respectively.

2 What Is the Vulnerability?

We developed two types of attack to demonstrate what can be achieved by a blind in/on-path attacker:

- *Client-side* attacks inject spoofed packets into the network stack of the VPN client. Because of the prevalence of Network Address Translation and bogon filtering, for all practical purposes this implies that an attacker is network adjacent to the client. Because any network-adjacent attacker can easily place themselves in-path *via* ARP poisoning on a typical network, we do no distinguish between malicious access points and malicious attackers who share the same network in layer 2. Many clients (particularly those running UNIX-like operating systems based on the weak host model) do not discriminate packets based on the interface through which they entered the system, so it is possible for an attacker to spoof packets appearing to come from a remote application server to the VPN client's IP address inside the

VPN network NAT.

- *Server-side* attacks inject packets by spoofing them to the VPN server, appearing to come from a remote application server. Such packets can be spoofed from effectively anywhere on the Internet, but since our attacks assume that the attacker can view encrypted VPN traffic (to count encrypted packets or bytes over time), server-side attacks must, for all practical purposes, be carried out by a router that is in-path between the VPN server and VPN client. Because the packet arrives at the VPN server on the same interface as legitimate packets, and is otherwise indistinguishable from a legitimate packet in terms of header information, we believe that server-side attacks will be much more challenging to mitigate.

There is no operating system implementation detail, VPN design decision, or configuration setting that we can point to as being the vulnerability that enables our server-side attacks. Rather, our server-side attacks are based on the general architecture that defines how VPNs work. Filtering packets for tunneled connections by interface (typically using a firewall rule) or technologies such as Linux's network name spaces are ways to effectively mitigate our client-side attacks in most cases, but the client-side attacks are still good demonstrations of blind in/on-path attacks.

We demonstrate this new category of vulnerability by performing the following series of attacks against connections protected by an encrypted VPN tunnel:

1. A client-side attack to infer and hijack TCP connections from the perspective of a malicious network adjacent attacker (*e.g.*, a WiFi Access Point).
2. A server-side attack to infer and hijack TCP connections from the perspective of a middle router in-path between the VPN server and the VPN client.
3. A server-side attack to hijack DNS queries from the perspective of a middle router in-path between the VPN server and the VPN client.

For the sake of clarity, we assert the following about our attacks:

- **Both client- and server-side attacks work regardless of the strength of the VPN's encryption.** Because we inject packets into the network stack of the VPN client or VPN server at the ends of the tunnel, where either encryption has not happened yet or decryption has already happened, our attacks are independent of any cryptography implementation of the VPN tunnel.
- **While client-side attacks can be mitigated by reasoning about which interface a packet arrives on to distinguish between spoofed and legitimate packets, server-side attacks cannot be mitigated in this way.** For our server-side attacks, spoofed packets arrive on the same interface as legitimate traffic, and can be identical in every other way. No vendors have proposed any mitigation for our server-side attacks, and all VPNs and OSes that we tested are still vulnerable.

Responsible disclosure: Our work resulted in the assignment of two CVEs, CVE-2019-9461 and CVE-2019-14899. The former is because Android responds to unsolicited packets sent to an incorrect interface in plaintext[2], and the latter is because all UNIX-like operating systems that we tested (Linux, BSD, and Apple's macOS and iOS) respond to unsolicited packets sent to the incorrect interface, and although the responses are encrypted, reveal enough information to infer the existence of connections and the correct sequence and acknowledgment numbers. In both cases, we show that the kernel of these operating systems before disclosure does not correctly discriminate packets meant for the VPN interface from normal traffic, which allows us to blindly probe the client until we have the information needed to inject arbitrary data into the connection. For all route-based VPN apps/configurations that we tested before our disclosure, we found them to be vulnerable on affected OSes.

However, despite major OS and VPN vendors issuing patches in response to our disclosure (*e.g.*, Android, Apple, and WireGuard), many of our attacks presented in this paper are still possible even with the latest versions. There are two reasons our attacks stay unmitigated. First, for client-side attacks, operating systems often need the weak host model for connectivity reasons, *e.g.*, so a mobile device can switch seamlessly between different cellular interfaces. Thus the filtering of malicious packets needs to be precise and is probably best carried out by the VPN client application because it has the most information about the VPN tunnel configuration. The other reason why many of our attacks remain unmitigated is because, **despite our disclosures, no vendors have proposed or implemented any mitigation for our server-side attacks. Our server-side attacks are not associated with any vulnerability; instead, they only assume that the VPN server correctly performs network address translation.**

3 Background

Here we present prerequisite background information, some of which (VPN Basics in Section 3.1) is general and the rest of which only applies to specific attacks for specific protocols, operating systems, or implementations. For example, IP source address validation (Section 3.2) is only directly relevant as a solution for client-side attacks.

[2] Our attacks do not assume this behavior, but it is something we noticed during testing that is specific to Android.

3.1 VPN Basics

There are two commonly used methods for controlling traffic in VPN software: policy-based implementations and route-based implementations. Route-based VPNs, which are the most common tunneling methods used in typical commodity VPNs, use virtual interfaces on both the client and server to act as endpoints on a virtual network. In the typical configuration, the VPN software on the client device modifies the routing table to send all of the traffic to the tunneling interface (*e.g.*, `tun0`) by giving it a more specific route than the default gateway. All of the traffic that is received by the tunneling interface is encrypted and encapsulated by the VPN software and then routed on the public-facing interface to the VPN server, where it is NATted to the VPN server's public IP and sent on to the ultimate packet's destination over the Internet. Policy-based VPNs, however, do not use an additional interface as an endpoint for a virtual connection, but instead use firewall rules to determine which traffic belongs to the VPN and encrypts any traffic matching the policy.

The purpose of VPNs is to prevent anyone in-path between the VPN client and the VPN server from seeing the contents of the user's traffic, and it is generally assumed that this portion of the tunnel is protected. Even if an attacker can see the packets sent between the VPN server and the final server (*e.g.*, a web server), they would not be able to determine the VPN client on the other side of the VPN server through analysis of the packets alone. An attacker can still perform traditional in-path attacks between the VPN server and web server, particularly when the application-generated traffic is not additionally encrypted by SSL/TLS standards. However, the encrypted tunnel between the VPN client and VPN server is meant to prevent these attacks from happening between the VPN client and VPN server.

3.2 IP Validation in Modern Protocol Stacks

On modern Linux, and other UNIX-like systems, source address validation for IPv4 is disabled by default, meaning that any packet received on any interface will be processed by the kernel, and if that IP address is a known local address, forwarded to the application or service associated with it. This is known as the *weak host model*.[3] In modern operating systems, this allows a user to have multiple interfaces receiving packets from the same source (*e.g.*, multi-homing), thus providing redundancy and more reliable network connectivity as users roam across network access technologies.

In an attempt to address the lack of source address validation, the concept of reverse path was developed in RFC 2827 and RFC 3704, which added filtering to check that incoming packets are routable *via* the interface on which they are received [7]. That is, if the packet is not routable through the incoming interface, the packet should be dropped, and only if the packet is routable through the incoming interface, will it be routed to its destination. This is implemented in most Linux-based systems through the `rp_filter` kernel variable, which offers three options defined in RFC 3704:

1. **Strict Mode:** In this mode, the source address from an incoming packet is compared to the Forwarding Information Base (FIB) and the packet is dropped if the incoming interface is not the best outgoing interface for responding to the packet.
2. **Feasible Mode:** In this mode, the source address from incoming packets is compared against the FIB, but maintains alternative routes and only drops packets which are not routable at all *via* the incoming interface.
3. **Loose Mode:** This mode compares the source address for incoming packets against the FIB, but will only drop the packet if it is not routable *via* any local interface.

RFC 3704 recommends using strict mode unless there is a specific reason for using feasible or loose mode, *e.g.*, in multihomed networks. A mobile phone offers an example of a device that relies on asymmetric routing, since it will likely have a WiFi interface and multiple interfaces for receiving packets from cell towers. The mobile phone needs to maintain persistence as the user switches networks as they travel beyond the range of their current cellular tower, change their WiFi network, re-connect to the network after losing coverage, get IP addresses re-assigned due to Carrier-Grade NATs or DHCP [24, 26], or switch from WiFi to mobile and *vice versa*. The reasons cited in a git commit from November 2018 to the `systemd` project [27] for setting the default for reverse path filtering to loose mode included default route changes (*e.g.*, plugging in an Ethernet cable while connected to WiFi) and connectivity checks. As a result, most Linux distributions using `systemd`, such as Arch, Debian, Fedora, and Ubuntu, will no longer drop packets with source addresses matching a connection inside the tunnel (using the `tun0` interface), and will accept them on any interface.

For our client-side attacks, this lack of source address validation gives an in-path attacker the ability to spoof packets to potential virtual IPs on the client machine and learn the virtual IP used by the `tun0` interface for the VPN connection. Additionally, the attacker can spoof packets with the source address of a given end-host to the virtual address and determine if an active connection exists by the timing and size of the client's responses, as we will describe in the next section. This is the root cause of our client-side attacks.

In an effort to prevent DDoS attacks, RFC 2827 establishes methods for limiting spoofed attacks by performing ingress filtering on the provider's routers between the client and the network edge. These recommendations are defined in BCP 38 and BCP 84 and require that the router that provides connectivity to downstream users drop packets that contain source

[3] In the strong host model, a packet received on an interface is only routed if the destination IP address is associated with the interface.

addresses not included in the prefixes they provide connectivity for [7]. These rules mirror the `strict`, `feasible`, and `loose` modes listed above for reverse path filtering on client machines. Previous work has shown that BCP 38 and BCP 84 are not universally implemented [37] across the Internet, but even if all the machines on the network edge implemented the filtering described in BCP 38 and BCP 84, this does nothing to prevent routers in the core of the Internet from spoofing source addresses. Additionally, these recommendations do not consider an attack from a malicious provider at the network edge, such as a state-level ISP.

3.3 Challenge ACKs and PSH/ACKs

The original specification of TCP in RFC 793 considered a connection to be reset if a `RST` packet was received anywhere in the receive window [1]. This made it relatively easy for an off-path attacker to reset connections compared to requiring the exact sequence number blindly, so RFC 5961 introduced the concept of a *challenge ACK* [30]. When a TCP host receives a `RST` in the receive window but where the sequence number is not an exact match, it sends a challenge `ACK` that should cause a `RST` with an exact sequence number as a response from the remote host only if that remote host truly has no record of the connection. Thus in-window `RST`s succeed only when the off-path attacker guesses the exact sequence number or the remote host that is the other party to the TCP connection effectively agrees that there is no connection. For our purposes in this paper, the important aspect of challenge `ACK`s is that they are part of an actual connection and therefore get TCP timestamps added to them.

The `PSH` flag in TCP informs a receiver that data should be pushed up to the application layer immediately. Combining `PSH` with `ACK` is a way to ensure that both a sequence and acknowledgment number for a connection are committed into the state of the connection and related data is sent to the application, even if overlapping sequence and acknowledgment numbers are received with different data later.

4 Vulnerability Set Overview

As discussed in Section 2, there is no operating system implementation detail, VPN design decision, or configuration setting that we can point to as being the vulnerability that enables our server-side attacks. Because blind in/on-path attacks are a general class of attacks, which we demonstrate in this paper by focusing on VPNs and attacking two specific protocols (TCP and DNS) as examples, in this section we review the general set of vulnerabilities that in/on-path attackers pose to a user's connections. We then consider how each of these vulnerabilities can be extended into the threat model of a *blind* in/on-path attacker attacking connections inside a VPN tunnel.

Table 1 discusses and compares the feasibility of five different in-path attacks for three different scenarios:

- "No VPN", where users are not protecting their traffic with VPN tunnels so all attacks are "trivial" because the attacker can see and spoof or modify any byte, header, or data.
- "Ideal VPN", where users benefit from a hypothetical VPN where the existence of packets and their size and timing are completely hidden from the attacker. The dominant paradigm for reasoning about what kinds of attacks are possible against connections that are tunneled through an encrypted VPN tunnel is based on "Ideal VPN" implementations, but modern VPN technologies are far from this model.
- "Real-world VPNs" subject to blind in-path attacks. All the VPN technologies that we tested fall under this category and the five attacks are practical in this scenario. This calls into question the current paradigm for reasoning about what security properties VPN tunnels provide.

The objective of this paper is to demonstrate the feasibility of the attacks against "Real-world VPN" implementations.

Due to the fact that the TCP connections inside the VPN are tunneled, the headers of the tunneled connection are not visible to the attacker as shown in Figure 1, but it is possible to infer the information in the headers by analyzing the responses from the client and server to spoofed packets. Using the methods described below, we can determine if a user has an active connection to a given IP address and find the `SEQ` and `ACK` numbers required to reset or hijack the TCP connection from either the perspective of a network adjacent user or an in-path router between the victim and the VPN server. Similarly, for DNS, we can infer when a DNS query is likely to have been made for a given domain by the victim machine and spoof acceptable responses back to the client *via* the server's NAT.

Our client-side attacks are network adjacent attacks where the client does not have reverse path filtering or any other kind of source address validation enabled (Section 4.2), and we take the role of the attacker (*e.g.*, a WiFi Access Point) and spoof packets of the tunneled connection to the wireless or Ethernet interface where they are processed by the kernel on the victim's machine. If we can correctly guess the four-tuple associated with an active connection, the kernel will respond to these packets and we can determine from examining their timing and size that there is an active connection. Once we determine that there is an active connection, we can continue to spoof packets and use the client's responses to narrow down the sequence and acknowledgment windows, giving us everything we need to inject data into the connection.

Our server-side attacks are from the perspective of the ISP, or any in-path router en-route to the VPN server[4] as shown

[4] For asymmetric routes, it is actually the route from VPN server to VPN client that matters. We assume that the attacker is positioned in the network so as not to be affected by asymmetric routing.

Attack	No VPN	Ideal VPN	Real-world VPN
Infer the existence of a TCP connection	Trivial, look at port and IP address fields in the TCP and IP headers	TCP and IP headers are protected by encryption	Ports and IP addresses can be inferred *via* packet timings and sizes (see Section 4.2)
Reset a TCP connection	Trivial, spoof a `RST` based on ports and IP addresses	`RST` cannot be injected because of the encrypted VPN tunnel, and port and IP address information is hidden by the encrypted VPN tunnel	`RST` can be injected at the client end of the VPN tunnel depending on client OS and configuration, `RST` can be injected at the server end of the VPN tunnel regardless of OSes or configurations (See Section 4.2.3)
Hijack a TCP connection to inject arbitrary data	Trivial if there is no application-layer encryption/authentication (such as TLS), simply read the sequence and acknowledgement numbers from the TCP header	Regardless of application-layer encryption/authentication, data cannot be injected because of the encrypted VPN tunnel	Sequence and acknowledgment numbers can be inferred *via* packet timings/sizes, data packets can be injected just like above (see Section 4.2.3)
Hijack a DNS query	Trivial, intercept it and reply with the fake one	DNS query and response are protected by the VPN tunnel encryption	Ports can be inferred, transaction IDs brute forced, DNS responses injected at the server end of the VPN tunnel regardless of OSes or configurations (see Section 4.3)
Perform a man-in-the-middle attack	Easy, if the attacker is in-path and has a valid SSL/TLS certificate	VPN tunnel would protect the traffic even if the attacker has a valid SSL/TLS certificate for a tunneled connection	Easy, if a server-side attacker is in-path and has a valid SSL/TLS certificate (see Section 6.3)

Table 1: In/on-Path attacks and how they change the way we should think about VPNs and other technologies based on encrypted tunnels. In the *No-VPN* case, in/on-path attacks are trivial. In the *Ideal VPN* case, we consider a hypothetical VPN in which packets and their size and timing are completely hidden from the attacker. The *Real-world VPN* scenario considers real-world VPN implementations in which blind in/on-path attacks are feasible.

in Figure 1 (See Section 4.3). The process is essentially the same, except that the packets are not being sent to the incorrect interface; instead, they are instead sent to the VPN server (which should be reachable from anywhere on the Internet) with the same properties as legitimate traffic. Attacking connections at the server-side end of the tunnel has two major advantages. The first advantage is that there is no way for the VPN server to distinguish between attacker probes and legitimate packets from the actual connection because they will be identical and come in from the same interface. The other advantage is that any router along the path between the VPN client and the VPN server can now carry out the attack; they only need the (very common) ability to spoof packets on the Internet with arbitrary return IP addresses. A major challenge for attacking TCP at the other end of the tunnel is that packet loss, packet reordering, and packet delay can play a significant factor. Conceptually, the prospect of attacking at the other end of the tunnel renders all the types of mitigation offered for our client-side attacks moot because they are all based on reasoning about interfaces and IPs.

4.1 Attack Considerations and Scope

Our attacks have many aspects to them that are dependent on the attacker's position in the network, the protocol being attacked, and whatever types of Network Address Translation (NAT) or filtering may be being applied. It is important to note that any tunneled protocol can be attacked from either side of the tunnel (spoofing to the VPN client or VPN server), and our attacks on TCP and DNS/UDP are simply based on our choice to demonstrate simple attacks for illustration of the underlying concepts. While we chose to distinguish between client-side and server-side attacks for the presentation in this paper, leading to network adjacent and in-path attacks, respectively, it is important to note that injecting packets either way combined with the powerful primitive that a blind in/on-path attacker can count encrypted bytes or packets over time can lead to many different attacks. For example, an in-path attacker could carry out some of our attacks that are labeled as network adjacent if they had the ability to spoof packets to the client from arbitrary return IP addresses despite not being network adjacent (*e.g.*, in the absence of NAT and bogon filtering). We only mention the possibility here and it is not part of our main presentation. In fact, two major advantages, from the perspective of the attacker, of spoofing packets to the VPN server rather than the VPN client are:

- It is safe for the attacker to assume that the VPN server has an Internet-routable IP address. Thus any type of bogon filtering applied by routers between the attacker and VPN server is moot. It also means that the attacker can reach the VPN server without having to go through any NAT.
- The VPN server has a well-defined behavior that the attacker can use to inject traffic into the tunnel, which is that NAT is specified in RFCs (particularly RFC 2663 [29]) to work based on the five-tuple of protocol, source and destination IP address, and source and destination port. So an attacker can easily infer the ephemeral port[5] and then inject data into the VPN tunnel at will. This is compared to spoofing packets to the client, which requires that the client

[5] This ephemeral port is chosen by the VPN server, but typically is chosen to match the ephemeral port chosen by the client when possible. Our server-side attacks only care what the ephemeral port of the VPN server is, it does not matter if they match.

respond with some type of error that enters the tunnel and carries information that is useful to the attacker.

We also want to stress that the specifics of any attack we present do not represent vulnerabilities in themselves. For example, for inferring the sequence number to reset or hijack a TCP connection for client-side attacks we take advantage of the fact that TCP challenge ACKs are larger than RSTs because they contain an optional timestamp that RSTs do not. This is only the simplest one of a plethora of ways we could have implemented this part of that specific attack, and changing that behavior of challenge ACKs will not prevent the attack. The underlying vulnerability is a more general one: secret randomized values are used to protect protocols from blind off-path attackers but those values currently have no protection against being inferred by a blind in-path attacker.

4.2 Client-side Attacks

In the case of the client-side attacks, which we assume are network adjacent for this paper, we consider a person using a VPN because they are concerned about their security and privacy on a public WiFi access point. When connected to the VPN, all of their packets are routed through the local gateway on to the VPN server, and the gateway will only see encrypted packets traveling between the local IP of the client and the public IP of the VPN server. Since the gateway does not know the virtual IP address assigned to the tun0 interface, the public IP address of the web server that is communicating with on the other end of the tunnel, or the the ports associated with either end of the connection, they cannot perform traditional in-path hijacking attacks.

A client-side attacker can, however, infer the existence of connections to a given website, determine the sequence and acknowledgment numbers of an existing TCP connection, and reset that connection with a TCP RST. In the case that there is no additional encryption at the application layer, *via* SSL/TLS or otherwise, they can also inject arbitrary data into the connection. To perform this attack, the attacker needs to perform the following steps, further outlined in Figure 2:

1. Determine the VPN client's virtual IP address;
2. Use the virtual IP address to make inferences about active connections; and
3. Use the replies to unsolicited packets to determine the sequence and acknowledgment numbers of the active connection to hijack the TCP session.

4.2.1 Phase 1: Finding the Client's Virtual IP

In the first part of client-side attacks, after the client has connected to the malicious access point and then to the VPN server, we probe the connected user with SYN packets across the virtual IP space, which for most VPNs is a subset of the 10.0.0.0/8 block, to solicit a response from the victim machine that leaks information about the state of the active connection inside the encrypted VPN tunnel, allowing us to infer both the existence of a VPN connection and the victim's private IP address on the VPN server's subnet. For example, if the attacker spoofs a SYN packet to the device's WiFi interface with the source address of the local network gateway (and this works the same for any other interface, such as a cellular network), Linux will always respond with a RST with the source address of the virtual IP address in plaintext.

Furthermore, when sending a SYN packet to the incorrect virtual IP address, the packet is dropped and there is no response from the victim machine. A SYN packet sent from the access point gateway to the correct private VPN IP address, however, will send a RST packet on the wireless interface notifying the gateway that the address is receiving packets which were not intended for it. Conversely, probing with SYN/ACK packets will generate the exact opposite behavior, responding with RST packets for each SYN/ACK packets with the incorrect private IP, and not responding at all when sending SYN/ACK packets to the correct private IP.

Note that for server-side attacks this phase can be skipped because the server will NAT the spoofed packet to the client for us based on port information.

4.2.2 Phase 2: Making Inferences About Active Connections

Similarly, for client-side attacks, if we want to determine if a VPN user is connected to any particular application server address over the VPN tunnel, we can send SYN or SYN-ACKs from that address to the victim's private VPN IP across the entire ephemeral port space. The observed behavior for both SYN and SYN/ACK packets is similar to that of the SYN probe used above to determine the private VPN IP address. That is, when sending a SYN packet to the correct four-tuple, a RST packet will be sent on the wireless interface, but when sending to the incorrect four-tuple, nothing is sent back to the gateway.

After we have determined that there is an active VPN connection on a connected device, we will test for an active connection by spoofing SYN packets from a given server IP to the VPN user. We can assume that the website will be running on either port 80 or 443, and since we learned the victim's virtual interface IP from the previous step, we now only need to scan the entire ephemeral port space[6], looking for a RST to indicate that there is an active connection.

4.2.3 Phase 3: Hijacking Active Connections

Finally, once we have determined that the user has an active TCP connection to an external server on a given port, we

[6] 32768 to 60999 on most Linux machines, for example.

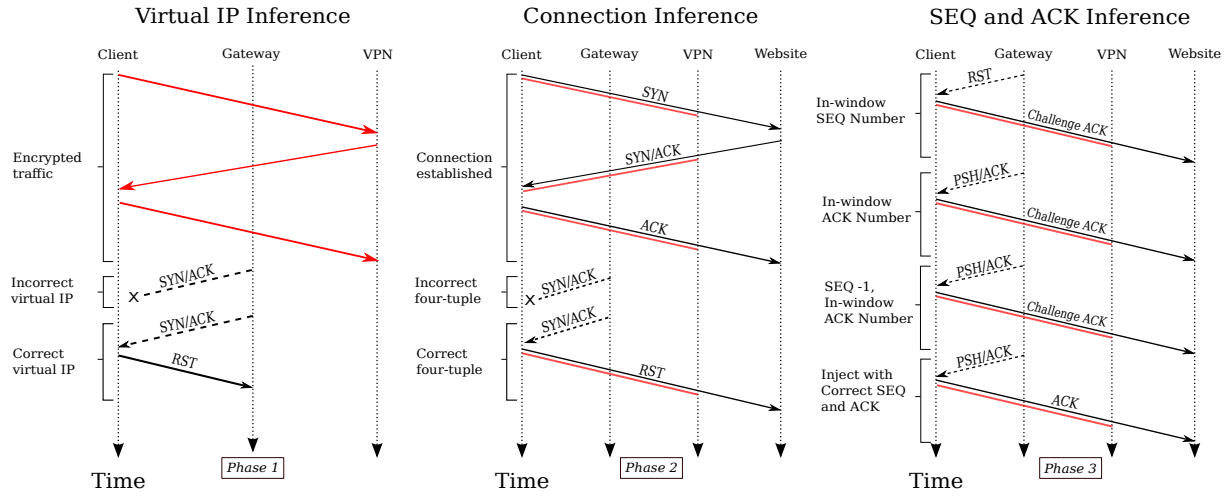

Figure 2: Outline of the three phases of a client-side attack on TCP.

will attempt to infer the exact next sequence number and in-window acknowledgement number needed to inject spoofed packets into the connection. For client-side attacks, to find the appropriate sequence and ACK numbers, we will trigger responses from the client in the encrypted connection found in Phase 2. The attacker will continually spoof RST packets into the inferred connection until it observes challenge ACKs. The attacker can reliably determine if the packets flowing from the client to the VPN server are challenge ACKs by looking at the size and timing of the encrypted responses in relation to the attacker's spoofed packets. For example, an Android device will trigger a TCP challenge ACK for each reset it receives with an in-window sequence number for an existing connection. If the client uses OpenVPN to exchange encrypted packets with the VPN server, the client will always respond with an SSL packet of length 79 when a challenge ACK is triggered.

We spoof RST packets to different blocks across the entire sequence number space until one triggers an encrypted challenge ACK. The spoof block's size plays a significant role in how long the sequence inference takes but should be conservative as to not skip over the client's receive window. In practice, when the script observes an encrypted challenge ACK, it can verify this is true by spoofing additional packets with the same sequence number. If there were the same number of encrypted responses with size 79 triggered, then we know it is triggering challenge ACKs.

After we have inferred the in-window sequence number for the client's connection, we can quickly determine the exact sequence number and in-window ACK needed to inject. First, we spoof empty push-ACKs with the in-window sequence while guessing in-window ACK numbers. Once the spoofed packets trigger another challenge ACK, an in-window ACK number is found. Finally, the attacker continually spoofs empty TCP data packets with the in-window ACK and sequence numbers as it decrements the sequence number after each send. The victim will respond with another challenge ACK once the attacker spoofs the exact sequence number minus one. The attacker can now inject arbitrary payloads into the ongoing encrypted connection using the inferred ACK and next sequence number.

4.3 Server-side Attacks

The server-side attacks, which are assumed to be in-path for this paper, follow a similar procedure to the one described above for client-side attacks, but do not need to know the client's virtual IP address, since the VPN server will NAT packets to it.

To determine if the victim is communicating with a given online application, the attacker will spoof packets with the destination IP of the public VPN server and the source IP address of the application server. The source port of the probes will typically be 80, 443, or 53 depending on if the attacker is trying to infer an existing web connection or a DNS query. The attacker only needs to determine the destination port of the VPN server to complete the four-tuple that will solicit a response that will be forwarded to the victim from the VPN server.

In order to find the port being used on the VPN server to communicate with our target web address, the attacker can probe each ephemeral port the server could have chosen for that connection. The attacker can send empty UDP or TCP packets depending on the type of connection they are trying to infer to each port on the VPN server. If the spoofed packet matches an existing conntrack entry on the VPN server, it will be NATed and forwarded back *via* the encrypted tunnel to the victim, thus the attacker will be able to observe it. Otherwise, the packet does not match an existing connection and will be dropped by the server.

Instead of probing each individual port and waiting to see if it responds, the attacker can dramatically increase the speed

and accuracy of the connection inference by sending different sized UDP or TCP probes throughout the probe so that when a match is found, the attacker can narrow down the exact port based on the size of the response. Typically, Ethernet limits the maximum frame size to ~1500 bytes, but for simplicity we chose to split the entire ephemeral port range into blocks of 1000 (*e.g.* 32k-33k, 33k-34k, ..). The attacker scans through each 1k-block by sending a random payload with a size that increases by 1 after each send and resets back to 0 at the start of the next 1k-block. We inject pseudorandom data to avoid the effects of compression on the size of the ciphertext.

The attacker can use the size of the now encrypted probes they observe to narrow down the range where the exact port in use is located. For example, if the attacker observes an encrypted packet of size 500, then they know the last three digits of the exact port in use is around 500, but could have been triggered by a few different 1k-blocks of ports. After the initial rapid scan with 1k-block buckets, the attacker does one final scan to determine which 1k-block triggered that size of packet. Our script jumps back five 1k-blocks to account for any forwarding delay, then begins probing while increasing the packet size until another one of the probe packets is observed. To further increase accuracy, the attacker can send multiple copies of the probes to ensure the amount of identical sized packets that are sniffed match the amount of probes sent to each port. In this final scan there are no packets spoofed with identical sizes so the attacker can know exactly which port was matched based on the size of the packet. At this point with the port inferred on the VPN server, the attacker has all four values of the 4-tuple needed to inject packets into the TCP connection or UDP flow.

UDP is most naturally attacked by spoofing packets to the VPN server rather than the VPN client. The VPN server's NAT leads to a behavior that is easy to see: incorrect ports lead to no encrypted/tunneled packets, and the correct port leads to encrypted/tunneled packets. Spoofing UDP packets to the VPN client (for example, in a network adjacent attack) is possible, but UDP's behavior of sending ICMP errors for all the incorrect ports and (typically) nothing for the one correct port is harder to detect with traffic analysis. In general, spoofing to the VPN client causes packet responses to be tunneled through the VPN, whereas spoofing to the VPN server causes the spoofed packets to be NATed only when the 4-tuple is correct. This an important qualitative difference, and is why we hijack DNS on the VPN server side of the encrypted tunnel.

4.3.1 Server-side DNS Hijacking

Once the attacker has inferred a UDP connection as shown in Figure 3, the flow on the VPN server that corresponds to a current domain query from the victim machine, they can start trying to inject an acceptable DNS response. In order for the victim to accept the attacker's spoofed response as valid

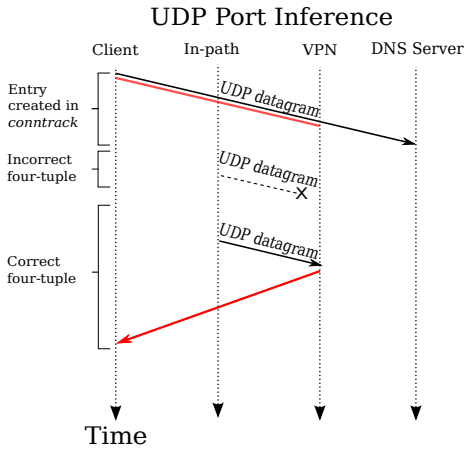

Figure 3: Outline of a server-side attack on DNS.

it must include the correct transaction ID (TXID), domain name, and reach the client before the lookup times out. The attacker can quickly cycle through each of the ~65k possible `TXIDs` since they are not concerned with sniffing any more packets from the victim machine at this point.

4.3.2 Spoofing Transaction IDs and the Hostname

DNS packets contain a 16 bit transaction ID field to help prevent DNS injection attacks. However, this means the attacker only needs to send ~65k packets to the inferred UDP flow to try each possible `TXID` it could have chosen. Therefore, DNS hijacking involves significantly less work to inject packets into compared to TCP which has two different 32 bit identifiers in the `SEQ` and `ACK` number. The attacker can determine which target hostname to attempt in the injection by using the TCP version of this attack to infer long-lasting connections on the client machine. Preferably, web connections that will repeatedly need to query for the same hostname once the domain entry's TTL expires in the browser's DNS cache.

4.3.3 Timing Considerations

One of the main obstacles for the attacker trying to inject the malicious response will be the victim's DNS query timeout period. The attacker must infer the victim source port chosen and try every possible transaction ID before the client system closes the port. For most desktop browsers, including Chrome and Firefox, the default DNS timeout is 5 seconds for each lookup. However, mobile browsers including Firefox and Chrome on Android use 10 second DNS timeouts. Many applications control the timeout of each DNS query, but if not they will fallback to the system's default settings. On modern Linux systems (*i.e.*, Ubuntu 20.04) the default system setting is 5 second timeouts with 2 retransmissions meaning the port will be open for 10 seconds. The attacker's accuracy in terms of successful injections depends on the length of this

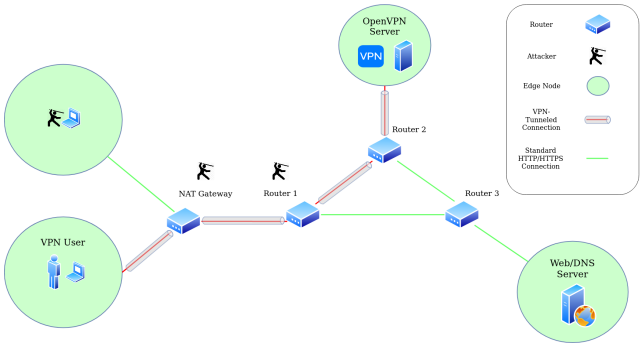

Figure 4: Description of the testing environment for Experiment IV.

DNS timeout setting. In order to not race with the legitimate DNS response, we carry out a per-IP-address denial-of-service attack on the DNS server so that it stops replying to the VPN server's IP address.

5 Methodology

Below, we describe the instrumentation required to showcase the client- and server-side attacks described in the previous section and our methodology for testing both attacks. We used Ubuntu 18.04 on all of the machines in our experiments running scripts which utilized libtins[7], a C++ packet crafting and sniffing library, to craft and spoof the packets and analyze the returning traffic[8]. We used NordVPN as our test VPN for the client-side attack and for testing the prevalence of the vulnerability on multiple operating systems as it is one of the most popular consumer VPNs and provides servers running both OpenVPN and Wireguard[9], as well as support for a variety of operating systems.

Our experiments include:

- I Do the attacks work at all?
- II Timing of each phase of the client-side attack on TCP
- III Success rate of the client-side attack on TCP
- IV Success rate of the server-side attack on DNS
- V An end-to-end "real-world" attack to put the server-side attack on DNS into context
- VI Analysis of vulnerability on a variety of OSes
- VII Prevalence of vulnerability on Android Apps

Experiment I is an integral part of all of the other experiments, so we did not perform a separate experiment. That our attacks work can be seen in each of the following experiments.

[7] http://libtins.github.io/

[8] The source code for our attacks can be found here:https://git.breakpointingbad.com/Breakpointing-Bad-Public/vpn-attacks

[9] At the time of testing, this also included L2TP/IPSec which NordVPN has since discontinued.

Client-side Attack: For Experiments II and III, our testing environment consisted of two machines, one acting as a gateway, broadcasting a wireless access point, and a victim connected to this gateway. The victim then connects to a single NordVPN server located in the United States. The victim then connects to NeverSSL, a website which only utilizes HTTP. We performed this test 1,000 times and note the success of this attack and the amount of time the attack takes. For the spoof block size, *i.e.*, how much we increment our guessed SEQ number by in each probe in Phase 3, we used 50,000. This is reasonably conservative in modern network stacks because of TCP receive window scaling.

Server-side Attack: For Experiment IV, we created a virtual lab consisting of seven virtual machines configured to emulate the routes of a connection to a VPN server and website or DNS server as depicted in Figure 4. We have an "Internet" consisting of three routers, with the VPN server, a DNS server, and a gateway, each connected to one of the routers. The VPN client connects to the gateway, where their connection is NATted.

In order to test the ability of a server-side attacker to inject a malicious DNS response to the VPN client *via* the VPN server's NAT, we tested the attack script against client queries with different timeouts between 5 and 15 seconds. We capped the victim DNS timeouts at 15 seconds, which is the maximum DNS query timeout we ran into in practice on Android 11.0.X. Most modern desktop applications including up to date web browsers (*i.e.*, Firefox 80.0.1) uses a 5 second query timeouts so we chose this as the minimum bound for the tests.

We ran 1,000 tests against each of the three standard DNS timeout configurations to test the ability and accuracy of the server-side attack on DNS. In each test, the victim VPN client issues a single DNS lookup using `nslookup` and the specified query timeout for that experiment. The attacker node starts the injection script a half second later in each test and attempts to infer the UDP flow and inject a malicious response in time.

For Experiment V, to illustrate what a real-world attack might look like, and the ways in which VPN security can be fundamentally undermined by blind in-path attacks, we developed a server-side attack that effectively removes all security and privacy properties of a VPN tunnel. In 2009, attackers in Iran with access to the national backbone obtained a valid TLS/SSL certificate for `facebook.com` and used man-in-the-middle attacks to steal Facebook passwords and view the Facebook activities of Iranian users. In this subsection we explore how that attack might have been carried out if Iranian users had had access to the latest version of WireGuard as of our experiment (version 1.0.20200827), which contains all patches WireGuard has released or intends to release based on our ethical disclosures. By tunneling all Internet traffic, including DNS requests and web traffic to/from Facebook, to a secure WireGuard VPN tunnel outside the country users should, in theory, have been protected from man-in-the-middle attacks in the backbone of the Iranian Internet. In this experiment we

seek to demonstrate that a blind in-path attacker can remove the VPN encryption layer and perform the man-in-the-middle attack to strip off TLS encryption of the HTTPS traffic.

For Experiments VI we tested the client-side attack against different operating systems to determine if they are vulnerable, and tested a variety of OS combinations for VPN client and VPN server for the server-side attacks to confirm that it is independent of OS. Additionally, since Android was a particular focus of our study in the early stages of our research effort, we also tested the client-side attacks against 35 VPN apps and services (a complete list is in the artifact associated with this paper) with Android as a client. This was for Experiment VII . Our selection procedure for deciding which VPN services and apps to examine was based on their popularity and market presence according to data gathered from the Google Play Store, Apple App store, and App Annie. We also included apps such as Wang VPN, Lantern, Psiphon, and Orbot which are commonly recommended within the security community, or actively used in nations with pervasive information controls. These tests were performed on a number of flagship mobile devices running the most recent operating system version and security updates, which at the time of writing was a Google Pixel 3 XL running Android 10 (with November 2019 security updates). We also tested older Android devices (all belonging to the research team) that are no longer officially supported but still in widespread use. For the server-side attacks on DNS, we tested against, or derived DNS timeouts from, a variety of DNS clients, as detailed in Section 6. This included a variety of browsers and operating systems.

Our experimental methodology differs between client- and server-side attacks for two main reasons: server-side attacks are independent of operating system or VPN version or configuration, and server-side attacks would most likely be carried out by large national-level ISPs—an environment that is currently beyond our scope to be able to test. Thus for client-side attacks we focus on testing a wide variety of OSes and VPNs and producing realistic performance numbers, while for server-side attacks we focus on demonstrating feasibility.

6 Results

In this section, we measure the success rate and time required for performing the attacks listed in Table 1 using the methods described in Sections 4.2 and 4.3. The exception is the TLS interception attack, demonstrated in Section 6.3, for which we do not report performance metrics given its particular nature. We outline all the information that can be monitored for each protocol and the consequences of this information being inferred.

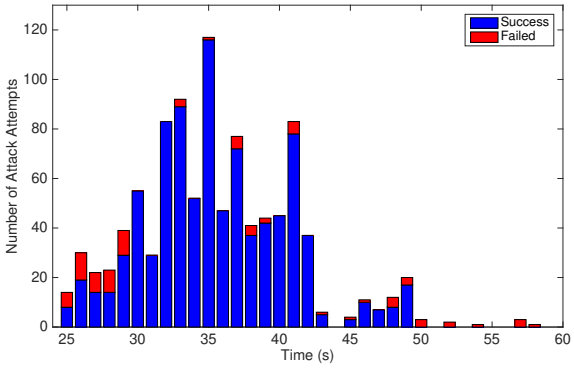

Figure 5: Results of client-side attack on a WiFi network.

6.1 Client-side TCP Attack (Experiments II and III: Timing and Success Rate)

In this case, we break-down the results for each phase of the attack as described in Section 4.2. The success rate of the entire attack is illustrated in Figure 5.

6.1.1 Phase 1

In the first phase of the client-side attack, where we determine the virtual IP address assigned to the client, we need to scan the /24 of the assigned virtual IP space. Although VPN providers can theoretically use any IP address in the IANA-reserved blocks for private networks (*e.g.*, 10.0.0.0/8, 100.64.0.0/10, 172.16.0.0/12 or 192.168.0.0/16), we found that most of the VPN servers we tested only used a subset of the 10.0.0.0/8 block, with only WireGuard servers on Mullvad using addresses in the 172.16.0.0/12 block. Additionally, we identify that the way that these addresses are used and assigned to clients is predictable in many providers. For example, NordVPN assigns users an address based on the operating system they are using. Android devices are assigned addresses in the 10.7.0.0/16 block, with Linux and macOS/iOS assigned addresses in 10.6.0.0/16 and 10.8.0.0/16, respectively.

For this phase of the attack, we assume that the attacker is familiar with the VPN server the client is connected to, so they will know the scheme for assigning virtual IPs. However, even if the attacker does not know anything about the server the victim is connected to, scanning an entire /16 only takes around 8 seconds using our attack script, so it does not significantly increase the time the attack takes to complete.

6.1.2 Phase 2

To make inferences about active connections, our attacker is not concerned with scanning every possible IP the victim could potentially be connected to, but is performing a targeted attack from a list of given websites or IP addresses and online services, such as a nation-state's list of banned websites or

specific non-web applications. For our evaluation, we only tested against NeverSSL to illustrate the efficiency of the attack against a single website. We also assume that the server the user is connecting to is using port 80 or 443 but the attack can be performed for any TCP port. We can determine within 6 seconds if the victim is connected to a given website during this phase by completing the four-tuple for this connection through scanning the ephemeral port space of the client.

6.1.3 Phase 3

The primary difference in the time it takes for phase 3 to complete is the attacker sniffing false acknowledgements as it probes a significantly large range of sequence or acknowledgment numbers. For example, the attacker may sniff a false acknowledgement during the spread of the entire sequence number range, it will continue to probe the small range around that false sequence number until it finds no responses are being triggered from the victim. At that point it will retry scanning the entire range again until it finds an in-window sequence that will repeatedly trigger responses. The tests that took longer in time had to retry more of the scans and send more packets to the victim, or the sequence number was simply later in the search space by chance.

During our attempts, the 8.4% of the failures were all during this phase. Based on the specifics of our attack script, there will always be a ~5% chance the attacker resets the connection as it probes. During the last scan for an in-window sequence number, the attacker sends TCP `RST` packets in blocks of 20 to the victim. A `RST` packet sent with an in-window sequence number will trigger a response, but the connection will be completely reset if the exact sequence in use is hit. During our tests, 6.1% of the failures were due to the connection being completely reset by the attacker. The other 2.3% of the failures were mostly due to our script failing to ensure that an inferred value is indeed triggering responses because of traffic analysis challenges. Many of the failures found the exact sequence within 100 bytes of the one in use, but sniffed false challenge `ACK`s and did not resend enough empty `PSH-ACK`s to ensure it was indeed the exact sequence in use.

6.2 Server-side DNS Attack (Experiment IV: Success Rate)

Against each different DNS timeout tested in the experiments, the attacker was able to infer the ephemeral port in use quick enough on average to start attempting to brute force responses with the correct `TXID` back to the victim. The attack script was able to infer the port in use by the client in 3.96 seconds on average. Additionally, our script took an average of 6.89 seconds to scan through the entire 65k transaction ID range. The attacker could potentially scan at a faster rate, but risks overloading the client socket's receive buffer and as a result decreasing the accuracy of the injection attack.

As expected, the success rate of the attacker increased as the DNS query timeout on the client also increased. This allowed the attacker to try every possible TXID the client could have chosen before the UDP socket was closed. The main results of the three experiments are shown below:

- 15 second DNS timeout (*e.g.*, Android 11) - 75.3% successful injects
- 10 second DNS timeout (*e.g.*, Ubuntu 20.04) - 48.1% successful injects
- 5 second DNS timeout (*e.g.*, Firefox 80.0.1) - 11.6% successful injects

6.3 Real-World Example (Experiment V)

We successfully demonstrated that a blind in-path attacker can remove the VPN encryption layer and perform the man-in-the-middle attack to strip off TLS encryption of the HTTPS traffic, in a setup meant to emulate the Iranian attacks on Facebook from 2009 referenced in Section 5.

We assume that the user is actively using Facebook during the attack. This is an underlying assumption to any man-in-the-middle attack, namely that the user is using the service while the attack is happening. DNS time-to-live (TTL) values depend on many factors such as website, web browser policies, recursive DNS resolver, and location, but values on the order of minutes are common. For both Firefox and Chrome we observed that, in our environment, while a user is using Facebook they will make a DNS request for `facebook.com` about every two minutes. We also assume that we know the IP address of the DNS server the victim client is using for domain name requests. Since many VPN providers default to specific DNS servers this is a likely case. We know the destination IP address and port of the DNS server, and we know that post-NATing the source IP address will be the VPN server's IP address. For our example attack we could continually scan for open NAT table entries by repeatedly carrying out the process of guessing the ephemeral port (*i.e.*, the source port chosen by the VPN server), but to avoid constantly sending the VPN server traffic we wait until we have fingerprinted a likely DNS request for `facebook.com` based on the size of an encrypted packet from VPN client to VPN server. Then, we carry out our VPN injection attack as per Section 4. Recall that this involves DoSing the DNS server so that we do not need to race with any valid response, inferring the ephemeral port number using a blind in-path attack, and then brute forcing the transaction ID (TXID) to inject a spoofed DNS response.

For route-based VPNs, such as WireGuard, there is a simple way to use DNS spoofing to cause a subsequent connection to be made outside the VPN tunnel. By returning the IP address of the VPN server itself as the response to the DNS query we can cause the new connection to leave the VPN client machine

unencrypted and outside the VPN tunnel because a specific routing rule exists for that IP address to make sure packets sent to the VPN server for the tunnel are not themselves coerced into the tunnel. The new connection will be a separate TCP connection, so can be easily distinguished by the attacker even if the destination port on the VPN server is the same. Because the attacker is, by definition, in the path from VPN client to VPN server, the attacker is in a position to perform any kind of man-in-the-middle attack on this new connection. For our example attack we use DNAT and `mitmproxy`[10] version 5.2, along with a forged certificate for `facebook.com`, to remove all encryption so that we can see and modify traffic to/from the `facebook.com` server at will.

6.4 Different OSes and VPN Apps (Experiment VI Testing OSes and Experiment VII Testing VPN Apps)

For the client-side attack we tested it against a wide variety of operating systems and Android apps. See the artifact[11]. for a complete list of operating systems and vendors that we tested, all of which were vulnerable. Notably, we tested a variety of VPN-like technologies such as Orbot, Lantern, Psiphon, TunnelBear, and others that are not advertised as VPNs. Essentially, all Linux- and BSD-based[12] operating systems were vulnerable before our ethical disclosure process, including Android and Apple devices. For client-side attacks, we only tested route-based VPNs (the vast majority of VPNs are route-based) and they were all vulnerable. We did not test policy-based VPNs, Windows OSes, or Tor [13] as part of Experiments VI and VII because we did not believe them to be vulnerable to the client-side attack due to separate network namespaces, the strong host model, and SOCKS interfaces being in user space, respectively. We later confirmed that these three OSes and apps are not vulnerable to the client-side attack.

For server-side attacks there is not a concept of a vulnerable VPN technology or OS, because the attack takes advantage of NAT when working as specified. Nonetheless, we tested with both policy-based (IKEv2/IPSec) and route-based (OpenVPN and WireGuard) VPNs on a variety of OS combinations for client/server: Windows/Windows, Windows/Linux, Linux/Linux, and macOS/Linux. For Windows as a server we only tested OpenVPN. It is not common for Windows to serve as a VPN server, and is not supported by OpenVPN (we had to mirror best practices configuration from Linux as closely as possible), but we wanted to underline the point that the server-side attacks are independent of operating system or implementation by having a VPN setup that did not involve Linux- or BSD-based OSes in any way. We specifically tested Windows 10 v20H2 as the client and Windows Server 2019 v1809 as the server. Note that Tor does not use NAT to multiplex connections on exit nodes, so is not vulnerable to our server-side attacks.

6.5 Operating System and VPN Protocol Differences

The client-side TCP attacks are possible on each operating system that we tested, but interestingly, each operating system has some nuances in the way in which it handles different kinds of spoofed packets. For example, BSD-based operating systems, including macOS (Sierra, High Sierra, and Mojave) and iOS (through version 12.4.1) require an additional step to determine the victim's virtual IP address. Android has an additional vulnerability that allows parts of the attack to be performed in plaintext, but we were able to make inferences about the encrypted packets we received which allowed us to perform the attack with only a moderate amount of effort.

The attacker is able to use the packet size of the encrypted communication to infer whether or not they are spoofing the correct four tuple. Each time the attacker guesses the correct four tuple in the spoofed `SYN` packet to the client, it will respond with an `ACK` through the encrypted connection with the VPN server. Since every single `ACK` the client sends through the tunnel is encapsulated in the exact same size encrypted packet, they can easily infer which encrypted packets are indeed `ACK`s instead of `RST`s. The connection can be reliably tested by sending a specific count of spoofed four-tuple packets, then counting the number of matching packet lengths flowing from the victim to the VPN server.

Multiple versions of iOS and BSD (note that iOS uses the FreeBSD network stack) were also found to be vulnerable, but we focused our efforts to reverse engineer routing on Linux/Android rather than BSD. Thus we only report here the small changes we made to our attack for it to work on these other OSes.

The constant size in which VPNs send challenge-`ACK`s within varies based on the protocol. For example, on Ubuntu 18.04, the OpenVPN protocol sends encrypted TCP packets of size 79, while iOS sends encrypted IPsec UDP packets of length 108 bytes for the triggered responses. An in-path attacker can reliably infer which encrypted packets are empty `ACK`s by sniffing the traffic long enough with any VPN protocol. It is important for the attacker to ensure that the outgoing client packet sent directly following each spoofed packet matches the appropriate `ACK` size for that encrypted VPN communication. Our attacks currently assume the attacker knows what type of VPN protocol is being used. Using traffic analysis and metadata, the attacker should have a clear idea of the VPN protocol being used.

[10] https://mitmproxy.org

[11] Also available at https://git.breakpointingbad.com/Breakpointing-Bad-Public/vpn-attacks

[12] For the purposes of this paper, we consider Apple OSes to be BSD-based in the loose sense that they borrow heavily from the FreeBSD networking stack.

7 Limitations and Discussion

Here we discuss the limitations and generality of the attacks we have presented.

7.1 Client-side Attack Limitations

Since enabling reverse path filtering will negatively impact the performance and reliability of networking on a number of devices, the recommended mitigation to prevent our attack is to add a pre-routing `iptables` or `nftables` rule to drop packets destined for the client's virtual IP address.

7.2 Server-side Attack Limitations

While these types of mitigation address the client-side attack for most non-mobile devices, source address validation (rather than reasoning about interfaces) is required for mitigation as we move further down the path closer to the VPN server. Research has shown that the majority of networks on the Internet do not even perform the most basic kind of source address validation [21], namely dropping packets entering their network that claim to be from their network. Source address validation becomes impossible once traffic flows reach BGP-powered routers on the Internet where asymmetric routing is possible.

For DNS, the victim application initiating the queries will typically determine if there is a DNS cache and if so, the length of time before that DNS entry expires. Therefore, once an attacker has inferred an existing web connection they can assume another query if the user visits the website again after the DNS record TTL value (or sooner depending on caching policies) [4, 5]. Additionally, they can get an estimate on the size of the target lookup by connecting to the same VPN server and crafting the same lookup.

Another obstacle for the attacker is mistaking other victim to DNS server UDP flows that are not for the correct target domain name. For example, the attacker wants to inject the wrong IP for `facebook.com`, but the client also sends a query for `google.com` around the same time. Thus two UDP flows are being NATed and the attacker might discover the ephemeral port for the wrong flow. To address this issue, the attacker can drop packets headed from the VPN client to the VPN server for a short period of time (*i.e.*, 5 seconds) in order to ensure there are no more lookups sent from the client while we attempt to infer the port for a specific flow.

7.3 Generality of the Attack

To better understand the generality of the attacks we have presented, it is instructive to separate our attacks into the two types: client- and server-side. Our set of client-side attacks effectively require the attacker to be on the same physical network as the victim client for one of their live interfaces, *i.e.*, adjacent in the link layer. Network adjacent attacks involve spoofed packets directly from the attacker to the victim as physical frames. However, our server-side attacks require only that the attacker be a router (or network adjacent to a colluding router) along the path from VPN client to VPN server. For these attacks the spoofed packets are routed over the Internet from the attacker to the VPN server. In either case, the spoofed return IP address is typically a server (such as web or DNS) the victim is accessing *via* a tunneled connection through the VPN tunnel.

When reasoning about the generality of both types of attacks the main considerations are:

- **TLS/SSL in the application layer for the tunneled connection**: While encryption of the VPN tunnel does not prevent our attacks, application-layer encryption (*e.g.*, TLS-based protocols like HTTPS) prevents injecting data into the socket. Yet, inferring the existence of a VPN-tunneled TCP/IP connection and resetting that connection are possible despite application-layer TLS/SSL. Hijacking a TCP/IP connection to inject data is only possible in the absence of TLS/SSL, however hijacking DNS for standard DNS configurations is possible and, for route-based VPNs (which are more common than policy-based), can lead to the possibility of stripping off application-layer encryption, as detailed in Section 6.3 in our real-world example. Furthermore, application-layer encryption is less commonly used by at-risk populations globally than by typical users from more developed countries. We scraped all websites marked as potentially blocked by the Citizen Lab [20] using Selenium and found that, for example, 26% of websites in China and 51% of websites in Brazil have at least one unencrypted element.
- **Necessity of knowing the timing of a connection or DNS request, along with the IP address or domain:** For our attacks to succeed we have to predict the timing of the connection or DNS request, or at least continue carrying out the attack until the connection or request happens. We also have to know the IP address that will be connected to or the domain name that will be requested. In our real-world example in Section 6.3 we observed that a user using Facebook will make DNS requests for `facebook.com` every two minutes, for example. That is all the information we need to carry out the attack, we do not need to predict the exact timing (but can, using traffic analysis to observe encrypted VPN packet sizes likely to be the DNS requests we are looking for).
- **Some of our attacks having been mitigated by patches:** Many (but not all) operating systems or VPN client vendors have applied some kind of patch to mitigate our client-side attacks in response to our disclosure. These patches largely amount to filtering out the spoofed packets because they come in from an interface that is not the virtual interface for tunneled VPN traffic. See Section 1 for details of our responsible disclosure process. We are not aware of any patches or planned patches to mitigate our server-side at-

tacks, despite having ethically disclosed them to multiple OS and VPN vendors on August 13, 2020. It is possible that attacks could be detected based on, *e.g.*, anomalies in fields such as the TTL, or monitoring incorrect guesses of fields such as port numbers, but no vendors have put forth a proposal to do so.

- **Reverse path filtering, martian filtering, and BCP 38 and 84:** Reverse path filtering comes in two forms: on hosts and in network routers. Strict mode as per RFC 3704 effectively stops our client-side attacks, loose or feasible modes do not. Reverse path filtering on the VPN server as a host does not affect the server-side attacks, because spoofed packets enter on the same interface as real packets. All source and destination IP addresses in our server-side attacks are routable Internet IP addresses, so Martian or bogon filtering are moot. BCP 38 and BCP 84 were addressed in Section 3. We assume that a state-level attacker in collusion with an ISP could easily remove any BCP 38 and BCP 84 restrictions and carry out the attack.
- **VPN configurations, implementations, and OS diversity:** There are many different configurations of VPNs, which can affect the VPN client, VPN server, or both. For our client-side attacks, a detailed discussion of how operating system and VPN configuration can affect the attacks is in Section 6.5. For our server-side attacks, the operating system and VPN configuration of the VPN client do not matter. Most VPN servers perform Network Address Translation (NAT) on the VPN server, and all NAT implementations have the behavior that we are exploiting: packets with the correct ephemeral port are NATed into the encrypted VPN tunnel while packets with the incorrect port are not. There are alternative implementations of VPNs that do not involve NATs, such as Outline which uses a SOCKS proxy[13]. Essentially, however, the same principle applies: in general spoofed packets with correct fields get tunneled and those with incorrect fields do not, meaning that an attacker that can see the encrypted tunnel can make inferences. Policy-based routing on the VPN client does not affect the underlying vulnerability for our server-side attacks, but our current method for causing connections to be made outside the VPN tunnel *via* DNS spoofing assumes that the client is using a route-based VPN client.

At the most basic level, all of our attacks combine two key elements: the ability to spoof packets that are either directly routed into the encrypted tunnel or the response to them is, and the ability to view traffic transiting the VPN tunnel even if the attacker cannot decrypt it. The mere existence, timing, and number of bytes of ciphertext leaks a lot of security-critical protocol information (such as port numbers and sequence numbers) when an attacker is able to spoof packets. While it *may* be possible to secure TCP and DNS within encrypted tunnels by applying the appropriate filtering, these are only two protocols and there are still many critical UDP-based applications (*e.g.*, NTP).

[13] See https://getoutline.org. Our server-side attack does not work unmodified on Outline because of a fast-close behavior for DNS traffic, we have not analyzed Outline's security against blind in/on-path attacks beyond that.

7.4 Summary and Recommendations

Because of the generality of blind in-path attacks for VPN tunnels we recommend the following:

- For transport layer protocols such as TCP, and for any application-layer transport-like functionality built on top of datagrams (such as DNS built on top of UDP), the security of each protocol should be examined with respect to the threat of a blind in-path attacker on a case-by-case basis. Among protocols that we leave for future work are QUIC, NTP, SCTP, and BGP.
- For training materials for at-risk users and any communication with users about the security and privacy benefits of VPNs, it should be made clear that VPNs are not a substitute for application-layer security (such as HTTPS or DNS over HTTPS, *i.e.*, DoH).
- To the extent possible, VPN configurations should use abstractions that are at a higher level than the network routing layer. For example, SOCKS proxies provide some protection against the attacks presented in this paper when properly applied.
- VPN architectures should apply IP address and interface filtering whenever possible. In addition to filtering already discussed, such as client-side firewall rules to stop spoofed packets from reaching the virtual interface, VPN providers should also consider protecting the path from DNS servers to the VPN server.

While we have broadly studied a variety of encrypted tunnel protocols that fall under the umbrella of VPNs, including OpenVPN, Wireguard, L2TP/IPSec, IKEv2/IPSec, and PPTP, there are also many such protocols that do not fall under the VPN umbrella such as SSH tunnels and VXLANs. We leave evaluation of these for future work.

8 Related Work

The main aspect of our work that distinguishes it from any prior work is the combination of applying traffic analysis of an encrypted tunnel with spoofed packets.

8.1 Security Analysis of VPN Services

There have been various studies around investigating the potential security and privacy aspects of VPN services. Perta *et al.* [25] manually investigated the network behavior of 14 VPN services and presented a DNS hijacking attack that

allowed traffic to be captured in clear. A more comprehensive study conducted by Khan *et al.* [17] on the commercial VPN ecosystem highlighted the lack of transparency in VPN policies and claims made to consumers. The work also elaborated on instances of leakages and active traffic manipulations by VPN providers. In the mobile space, there have also been extensive studies [16, 36], which have evaluated various VPNs apps in the Android app store. Their evaluation revealed the presence of malware, traffic redirection, DNS leaks, lack of encryption, Javascript injection and TLS interception by VPN providers. As a result of the lack of trust in the VPN ecosystem, researchers have also suggested decentralized approaches for VPN services [12, 31]. While previous work on the security of VPN services has mainly focused on the trust model and the correctness of the implementation of the protocols, our work specifically looks at the VPN security from a broader network routing perspective.

Several studies have looked at VPN routing issues at a more rudimentary level than our study. Perta *et al.* [25] and Ikram *et al.* [16] reported how misconfigurations in the VPN routing tables can enable DNS traffic leakage and hijacks. In contrast, our work hijacks DNS queries that are protected by the VPN tunnel and the threat model is any router in-path between the VPN client and VPN server. Appelbaum *et al.* [6] found various security vulnerabilities in VPN routing and suggested mitigation techniques for vendors. Similarly, another work [2] investigated the leakage of a VPN user's local IP address through the WebRTC-API and detailed the privacy risks associated with this. While these works revealed issues with routing and VPNs, we are not aware of any study before our own work that illuminated how the combination of packet spoofing with traffic analysis can reveal the randomized secret numbers that protocols use to protect against attacks such as inference of connections or hijacking.

8.2 Off-path Attacks

The first step in hijacking a connection is detecting the presence of an active TCP/IP connection. Watson [33] demonstrates how critical this step is for performing blind spoofing, session hijacking, and packet injection attacks, as well as TCP reset attacks, by taking advantage of a TCP specification of accepting out-of-order packets that are within the range of a window size, decreasing the search space by a factor of the window size. Other works have demonstrated attacks to infer the existence of a connection completely off-path [3, 18]. Such attacks typically involve placing canaries, finding collisions, and/or making statistical inferences. For our attacks presented in this paper, an attacker simply needs to sit in-path and guess a correct four-tuple by probing the ephemeral port space, and then see the tunneled response as an encrypted VPN packet. This does require that we assume one host's IP and port (*e.g.*, a web server), but this is the same assumption made by Cao *et al.* [11] and others to perform off-path TCP/IP hijacking attacks. Additionally, the number of devices that are vulnerable to our blind in-path attacks extends beyond Linux to other UNIX-based systems and mobile devices, as well as versions which have been patched to prevent the behavior exploited by Cao *et al.*

While off-path attacks that have nothing to do with VPNs or other encrypted tunnel technologies are a serious problem, they generally can be fixed by slightly changing the behavior of an implementation or randomizing other numbers for the protocol. For example, to address the attack by Cao *et al.* the Linux kernel randomized the total number of challenge `ACK`s that are sent per second, *i.e.*, the rate limit that led to the side-channel. While this could potentially still be inferred by an off-path attacker, a blind in-path attacker such as we have presented in this paper can trivially count packets. In general, it is much more difficult to hide the existence, timing, and size of network packets from an in-path attacker than from an off-path attacker. Off-path attackers by definition cannot perform traffic analysis of encrypted packets for the tunnel.

8.3 Traffic Analysis of Encrypted Tunnels

A number of works have made inferences based on analyzing the existence, size, and timing of encrypted packets for encrypted tunnels, such as works that fingerprint websites in encrypted tunnels [10, 32] or works that focus on censorship evasion [14, 34, 35]. None of these works combine packet spoofing with traffic analysis to subvert tunneled protocols, rather they focus on other higher-layer metadata such as the structure of the HTTP being served to a web client to identify the fingerprint of known web content.

9 Conclusions

We have demonstrated a general and serious problem (using attacks on popular VPN implementations as examples): in/on-path attacks to subvert protocols that are protected inside encrypted tunnels. Our attacks were demonstrated for the TCP and DNS protocols tunneled inside VPNs, but the underlying vulnerability applies to any attempt to use an encrypted tunnel to protect any protocol that uses randomly-generated secret values to protect against off-path attacks. This challenges the current understanding of real-world VPN's security by showing that even a properly configured and secured VPN that has applied all known security patches is still vulnerable to connection tampering from a malicious actor with the ability to control the gateway or any router between the VPN client and VPN server. In summary, all route-based VPNs and all UNIX-like operating systems that we tested were vulnerable to our client-side attacks before disclosure. Client-side attacks have been partially mitigated. Sever-side attacks are independent of VPN configuration or OS, so long as the VPN uses

the OS's NAT implementation on the VPN server. Despite full disclosure, no type of mitigation has been proposed or implemented by any vendor with respect to our server-side attacks.

10 Acknowledgments

This material is based upon work supported by the U.S. National Science Foundation under Grant nos. 1518523, 1518878, 1801613, and 2007741, as well as the Open Technology Fund and the Ministry of Science and Innovation (Spain) (PID2019-111429RB-C22). We thank our shepherd, Zakir Durumeric; the anonymous reviewers of both the paper and the artifact; and Jeffrey Knockel and Philipp Winter for valuable feedback. We also thank the developers who provided important insights during the ethical disclosure process, especially Jason Donenfeld and the other members of the OSS Security mailing list. Many unnamed members of our research team lent us mobile phones for testing, and Lynn Pham donated an iPhone, for which we are grateful. Steve Gibson and Leo Laporte provided comic relief with their fascinating analysis of our initial ethical disclosure.

References

[1] Transmission Control Protocol. RFC 793, September 1981.

[2] N. M. Al-Fannah. One Leak Will Sink A Ship: WebRTC IP Address Leaks. In *2017 International Carnahan Conference on Security Technology (ICCST)*, pages 1–5, 2017.

[3] Geoffrey Alexander, Antonio M. Espinoza, and Jedidiah R. Crandall. Detecting TCP/IP Connections via IPID Hash Collisions. *Proceedings on Privacy Enhancing Technologies*, 2019(4):311–328, 2019.

[4] Mark Allman. On Eliminating Root Nameservers from the DNS. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pages 1–8, 2019.

[5] Mario Almeida, Alessandro Finamore, Diego Perino, Narseo Vallina-Rodriguez, and Matteo Varvello. Dissecting DNS Stakeholders in Mobile Networks. In *Proceedings of the 13th International Conference on emerging Networking Experiments and Technologies*, pages 28–34, 2017.

[6] Jacob Appelbaum, Marsh Ray, Karl Koscher, and Ian Finder. vpwns: Virtual Pwned Networks. In *2nd Workshop on Free and Open Communications on the Internet (FOCI)*. USENIX, 2012.

[7] Fred Baker and Pekka Savola. Ingress Filtering for Multihomed Networks. RFC 3704, March 2004.

[8] Stephane Bortzmeyer. DNS Privacy Considerations. *Work in Progress, draft-ietf-dprive-problem-statement-06*, 1, 2015.

[9] Jonas Bushart and Christian Rossow. Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS. In *10th Workshop on Free and Open Communications on the Internet (FOCI)*. USENIX, 2020.

[10] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 227–238, 11 2014.

[11] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V Krishnamurthy, and Lisa M Marvel. Off-Path TCP Exploits: Global Rate Limit Considered Dangerous. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 209–225.

[12] Amir Chaudhry, Anil Madhavapeddy, Charalampos Rotsos, Richard Mortier, Andrius Aucinas, Jon Crowcroft, Sebastian Probst Eide, Steven Hand, Andrew W Moore, and Narseo Vallina-Rodriguez. Signposts: End-to-End Networking in a World of Middleboxes. *ACM SIGCOMM Computer Communication Review*, 42(4):83–84, 2012.

[13] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 21, 2004.

[14] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, page 445–458. Association for Computing Machinery, 2015.

[15] Xuewei Feng, Chuanpu Fu, Qi Li, Kun Sun, and Ke Xu. Off-Path TCP Exploits of the Mixed IPID Assignment. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, page 1323–1335. Association for Computing Machinery, 2020.

[16] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar, and Vern Paxson. An Analysis of the Privacy and Security Risks of Android VPN Permission-enabled Apps. In *Proceedings of the 2016 Internet Measurement Conference*, pages 349–364. ACM, 2016.

[17] 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 *Proceedings of the Internet Measurement Conference 2018*, pages 443–456. ACM, 2018.

[18] Jeffrey Knockel and Jedidiah R Crandall. Counting Packets Sent Between Arbitrary Internet Hosts. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, 2014.

[19] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. Coming of Age: A Longitudinal Study of TLS Deployment. In *Proceedings of the Internet Measurement Conference 2018*, pages 415–428, 2018.

[20] Citizen Lab and Others. URL Testing Lists Intended for Discovering Website Censorship, 2014. https://github.com/citizenlab/test-lists.

[21] M. Luckie, R. Beverly, R. Koga, K. Keys, J. Kroll, and K. Claffy. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *ACM Computer and Communications Security (CCS)*, November 2019.

[22] Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, Ron Deibert, and Vern Paxson. An Analysis of China's "Great Cannon". In *5th Workshop on Free and Open Communications on the Internet (FOCI)*. USENIX, 2015.

[23] Daiyuu Nobori and Yasushi Shinjo. VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls. In *11th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 229–241. USENIX, 2014.

[24] Ramakrishna Padmanabhan, Amogh Dhamdhere, Emile Aben, KC Claffy, and Neil Spring. Reasons Dynamic Addresses Change. In *Proceedings of the 2016 Internet Measurement Conference*, pages 183–198, 2016.

[25] 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.

[26] Philipp Richter, Florian Wohlfart, Narseo Vallina-Rodriguez, Mark Allman, Randy Bush, Anja Feldmann, Christian Kreibich, Nicholas Weaver, and Vern Paxson. A Multi-perspective Analysis of Carrier-Grade NAT Deployment. In *Proceedings of the 2016 Internet Measurement Conference*, pages 215–229, 2016.

[27] sysctl.d: switch net.ipv4.conf.all.rp_filter from 1 to 2. https://github.com/systemd/systemd/commit/230450d4e4f1f5fc9fa4295ed9185eea5b6ea16e.

[28] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted DNS–> Privacy? A Traffic Analysis Perspective. *Proceedings of the NDSS Symposium*, 2020.

[29] Pyda Srisuresh and Matt Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663, August 1999.

[30] Randall R. Stewart, Mitesh Dalal, and Anantha Ramaiah. Improving TCP's Robustness to Blind In-Window Attacks. RFC 5961, August 2010.

[31] Matteo Varvello, Iñigo Querejeta Azurmendi, Antonio Nappa, Panagiotis Papadopoulos, Goncalo Pestana, and Ben Livshits. VPN0: A Privacy-Preserving Decentralized Virtual Private Network. *arXiv preprint arXiv:1910.00159*, 2019.

[32] Tao Wang and Ian Goldberg. Improved website fingerprinting on tor. In *Proceedings of the 12th annual Workshop on Privacy in the Electronic Society, (WPES)*. ACM, 2013.

[33] Paul Watson. Slipping in the Window: TCP Reset attacks. 2004.

[34] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is Blocking Tor. In *2nd Workshop on Free and Open Communications on the Internet (FOCI)*. USENIX, 2012.

[35] Philipp Winter, Tobias Pulls, and Juergen Fuss. ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship. In *Workshop on Privacy in the Electronic Society*. ACM, 2013.

[36] Qi Zhang, Juanru Li, Yuanyuan Zhang, Hui Wang, and Dawu Gu. Oh-Pwn-VPN! Security Analysis of OpenVPN-Based Android Apps. In *International Conference on Cryptology and Network Security*, pages 373–389. Springer, 2017.

[37] X. Zhang, J. Knockel, and J. R. Crandall. Original SYN: Finding Machines Hidden Behind Firewalls. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 720–728, 2015.