Invisible Adversaries: A Systematic Study of Session Manipulation Attacks on VPNs

Yuxiang Yang*, Ao Wang†, Xuewei Feng*, Qi Li‡§, and Ke Xu*§✉
*Department of Computer Science and Technology & BNRist, Tsinghua University
†School of Cyber Science and Engineering, Southeast University
‡Institute for Network Sciences and Cyberspace & BNRist, Tsinghua University, §Zhongguancun Lab
yangyx22@mails.tsinghua.edu.cn, wangao@seu.edu.cn, fengxw06@126.com, {qli01, xuke}@tsinghua.edu.cn

***Abstract*—Virtual Private Networks (VPNs) are widely used for censorship evasion and traffic protection. VPN users expect to be provided with adequate security protection, and at the same time not be affected by other users connected to the same VPN server, which can be illustrated as the non-interference property. However, in this paper, we have identified several vulnerabilities that violate this property, specifically within the connection tracking frameworks of VPN servers, stemming from shared resource misuse and insufficient validation of session state transitions. We present three session manipulation attacks targeting TCP and UDP traffic tunneled through VPNs. The attacker who only connects to the same VPN server can launch denial-of-service attacks, hijack TCP connections of other clients, or inject forged DNS responses into their queries. We evaluate these attacks against five popular connection tracking frameworks across different OSes and nine major commercial VPN providers. Experimental results reveal that all frameworks and eight providers are vulnerable to at least one of the attacks. We have responsibly disclosed our findings with countermeasures, resulting in 19 assigned CVEs/CNVDs and acknowledgments from the communities and providers.**

I. Introduction

VPNs have emerged as a cornerstone in securing digital communication across the Internet [1], [2]. By creating encrypted tunnels between devices and remote servers, VPNs shield information from prying eyes, ensuring data confidentiality, integrity, and authenticity. Nevertheless, according to the non-interference property [3], VPN servers should be properly configured to ensure that processes from different users remain isolated and do not interfere with one another.

Previous studies have highlighted critical VPN vulnerabilities, including cryptographic weaknesses [4]–[6], endpoint misconfigurations [7]–[11] and in/on-path adversaries exploiting features such as packet size patterns [12], DHCP behaviors [13], or local IP manipulation [11]. Recently, Mixon-Baca et al. [14] proposed attacks breaching non-interference to redirect traffic or de-anonymize VPN peers. Their work primarily focuses on port conflicts between VPN clients and the VPN server itself, and the feasibility of the attack on real-world VPNs remains unverified. Compared with previous works to hijack TCP [15]–[19] and UDP [20]–[25] sessions in plaintext environments, session-level vulnerabilities in VPN ecosystems remain underexplored. This leaves open questions about VPN session isolation guarantees in real-world deployments.

In this paper, we conduct an in-depth exploration of session-level vulnerabilities in VPN ecosystems. We identify critical flaws in the shared connection tracking table and the insufficient validation of session state transitions at VPN servers. Then we propose three session manipulation attacks targeting TCP and UDP traffic tunneled through VPNs, which can be launched by an off-path adversary who shares the same VPN server as the victim. First, as the VPN server's public-facing source ports to a target server are limited, the attacker can exhaust all of them to launch a port exhausting DoS attack to the victim. Second, when the connection tracking framework of the VPN server adopts *Port Preservation* when managing session entries, the attacker can deduce randomized source ports of the victim sessions by sending probe and verification packets with guessed ports and observing the destination of the verification packets. Furthermore, it can abuse the inferred source ports to launch a TCP hijacking attack or spoof responses to UDP-based DNS queries from other VPN users. Third, the connection tracking frameworks may change session states when receiving crafted TCP `RST` packets without sufficiently verifying the sequence numbers. The attacker can remove others' session entries, and replace itself as the clients, thus totally hijacking the TCP sessions.

We verify these vulnerabilities and attacks on 5 popular connection tracking frameworks (i.e., Linux `Netfilter`, FreeBSD `PF`, `IPFW`, `IPFilter`, and `natd`) and 9 popular commercial VPN providers [26] with ethical guidelines. Our case studies on session DoS, HTTP injection, and DNS hijacking show that all frameworks and 8 providers are vulnerable to at least one of the attacks. On average, an attacker can launch a DoS attack in 4 seconds with over 90% success, and manipulate HTTP traffic within 64.11 seconds at a 66.7% success rate. DNS hijacking is also feasible, with success rates ranging from 20% to 70%. These results highlight that breaching non-interference undermines VPN security guarantees.

Finally, we propose countermeasures that aim to disrupt the attack prerequisites and uphold non-interference at the core of VPN security. We disclosed the vulnerabilities and attacks to Linux, FreeBSD, and affected VPN providers. Till now, we have received acknowledgments from Linux, FreeBSD, and six VPN vendors rewarded us under bug bounty programs. The other two vendors are still investigating them. In total, our findings have led to the assignment of 19 CVEs/CNVDs.

Contributions. Our main contributions are as follows:

- We uncover new vulnerabilities at the VPN server due to the violation of the non-interference property, which can be exploited by malicious users to manipulate others' sessions.
- We verify the existence of the vulnerabilities against popular connection tracking frameworks and evaluate the session manipulation attacks against well-known VPN vendors.
- We propose practical countermeasures and have responsibly disclosed the issues to the relevant organizations. We make our code publicly available at https://github.com/yyxRoy/vpn-attacks for ease of replication.

II. Background and Threat Model

A. VPN Architecture and Connection Tracking

VPNs enable secure communication over public networks by establishing encrypted tunnels between client devices and remote servers. In practice, modern VPN implementations usually create a virtual network interface (e.g., `tun0`) on the client side, which redirects traffic through a secure tunnel to the VPN server. All outbound packets are routed via this interface, encrypted, and then forwarded to the VPN server, which decrypts and relays them to the destination. To support IP address translation and session management, most VPN servers rely on connection tracking frameworks (e.g., Linux `Netfilter` or FreeBSD `ipfw`) to maintain per-session state. These frameworks act as stateful NAT middleboxes, rewriting packet headers by replacing the client's source IP with the server's public IP, and tracking session metadata (e.g., port, protocol, state, timeout) in a shared connection tracking table.

When an internal client `C` initiates a connection to a remote server `S`, the VPN server `V` will create a session entry that maps `C`'s internal address and port to `V`'s external ones. Consider `C` sending a TCP `SYN` packet with source port `c` to `S` listening on port `s` via `V`. The entry is recorded as:

$$\{[C : c \leftrightarrow V : c] \leftrightarrow [S : s],\ tcp = SYN\_SENT,\ timeout = 120s\} \quad (1)$$

Depending on the port allocation strategy, `V` may either attempt *Port Preservation* (reusing the same source port `c`) or apply *Random Selection* (assigning a new external port). Port conflicts are resolved by selecting unused ports dynamically. As the TCP session progresses, the entry state transitions accordingly (e.g., from *SYN_SENT* to *ESTABLISHED*) based on observed packets such as `SYN/ACK` and `ACK`. These transitions enable the framework to manage flows and reject unsolicited packets [27]. Moreover, to accommodate asymmetric routing or middlebox failures, most connection tracking frameworks enable a *loose state instantiation* mechanism by default, which permits the creation of a new *ESTABLISHED* entry directly from incoming non-SYN packets (e.g., `PUSH/ACK`) without a three-way handshake, effectively restoring the session mapping with the sequence numbers provided in the packet.

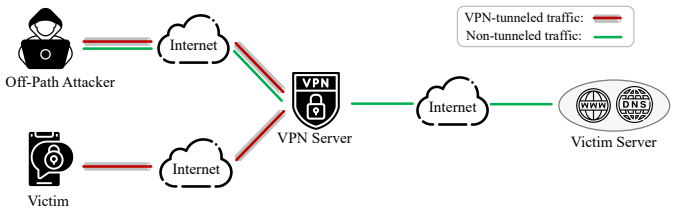

Fig. 1. Threat model

B. Threat Model

Figure 1 illustrates the threat model of our attacks. The target server provides services by accepting requests to an open port (e.g., 53 for DNS, and 80 for HTTP) and returning the responses. Depending on different scenarios, the target server can be chosen as popular servers as previous works [12], [17]–[19], [25] (e.g., search engines, famous websites) or the default DNS resolvers shared by the VPN users [12]. The VPN server works as an intermediate to protect the users who connect to it, which will transfer packets traversing through it by maintaining a stateful connection tracking table. The victim VPN user who has connected to the VPN server tries to communicate with the target server. For instance, it may send a request periodically to the target server and await the responses from it (e.g., DNS lookup, HTTP requests). The physically off-path attacker also connects to the VPN server and aims to manipulate the sessions of other users. It can send normal packets through the VPN tunnel (as the tunneled traffic in red color outlined in Figure 1) and is also capable of sending spoofed packets with the IP addresses of the target server outside of the tunnel (as those in green color). As reported in [28], [29], lots of ASes in the world still do not validate packets with spoofed source addresses. The physically off-path attacker also connects to the VPN server and aims to manipulate the sessions of other users. Since it has full control over its devices, it can selectively route normal packets through the VPN tunnel via the virtual network interface (e.g., `tun0`) (as the tunneled traffic in red color outlined in Figure 1). Simultaneously, we assume it is capable of sending spoofed packets directly to the VPN server's public IP address via the physical interface (e.g., `eth0`) (as the non-tunneled traffic in green color). Recent studies [19], [28] demonstrate that IP spoofing remains widely feasible across the Internet.

As required by the non-interference property [3], the behavior of one user or process in a system should not interfere with that of other users or processes. VPN servers typically enforce client isolation policies to block peer-to-peer communication within the tunnel, preventing the attacker from scanning the virtual subnet to discover other active clients. Consequently, the attacker initially has no knowledge of the victim's identity (e.g., the internal IP address assigned by the VPN or the client's real public IP) or the specific ephemeral source port allocated for its outgoing sessions. However, we find that the shared resources of connection tracking frameworks of the VPN server violate this property, which allows a malicious

VPN user to abuse the vulnerabilities to interfere with other users' sessions. Based on this threat model, we propose three practical session manipulation attacks, i.e., port exhausting DoS attack, TCP hijacking attack, and DNS hijacking attack.

III. Session Manipulation Attacks

A. Port Exhausting DoS Attack

Once the attacker has connected to the VPN server, it shares the network resources of the VPN server with the other VPN users that connect to it. However, a notable drawback of the shared environment lies in the source port limitation when considering upper-level transport protocols. Given that the source port field in TCP and UDP protocols is limited to 16 bits, a VPN server can use only 65,535 distinct source ports when initiating sessions to a target server. Consequently, once all these source ports are occupied by other sessions, users are rendered unable to initiate any outbound session to the target server, thus facilitating a port exhausting DoS attack.

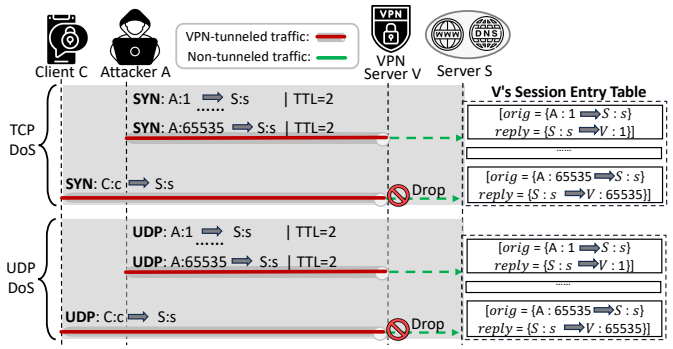

Fig. 2. Denial of service by exhausting usable ports

Figure 2 depicts the steps of the attack. For a target server S, the attacker can launch the attack by sending outgoing packets to consume all of V's available ephemeral ports for the connection 4-tuple. Take TCP protocol as an example, the attacker first sends 65,535 SYN packets with different source ports to S through the VPN tunnel. To prevent a large number of packets from arriving at S, the attacker can set the TTL of the SYN packet to a small number (e.g., 2), resulting in being dropped at the intermediate routers. V will create 65,535 session entries in the state of *SYN_SENT*, which will last for a certain period (e.g., 120 seconds by default in Netfilter). Moreover, the attacker can continue flushing the entries' *timeout* by sending outgoing packets again and again when they are closed to be cleaned.

After that, other VPN users cannot visit the target server anymore, as there is no ephemeral port left for a new session. Furthermore, since all VPN clients share the same DNS resolvers in a well-configured VPN setting, the attacker can amplify the attack by occupying all source ports to the DNS resolvers in advance, making others unable to resolve domain names and resulting in a more serious DoS attack. The attack clearly violates the non-interference property, where the attacker's actions disrupt other users' normal VPN usage.

The attack is more stealthy than traditional NAT DoS techniques that exhaust the entire table or system resources [30]–[32], which often generate explicit logs (e.g., "*nf_conntrack: table full, dropping packet*" in Linux), making them easier to detect by network administrators. In contrast, our approach targets specific servers and selectively consumes ephemeral ports, producing no system logs and using fewer resources. Setting low TTL values ensures packets are dropped before reaching the target, further reducing the detectability.

Enhancing Attack Persistence. Instead of leaving entries in the *SYN_SENT* state, the attacker may spoof SYN/ACK and ACK packets to complete the TCP handshake, transitioning entries to the *ESTABLISHED* state, which typically has a much longer timeout (e.g., 5 days in Netfilter). The same methods can be applied to the UDP protocol as well.

B. TCP Hijacking Attack

The procedures to perform this attack are as follows:

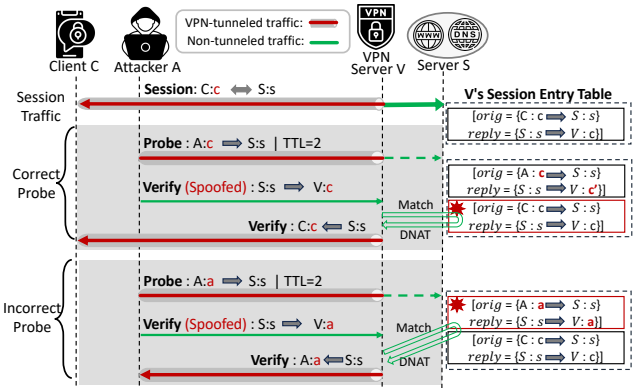

Fig. 3. Inferring the source port of the victim session

1) Making Inferences about Active TCP Sessions: The attacker A aims to determine whether a client C is communicating with a server S on TCP port s, and if so, identify C's source port c. If the VPN server adopts *Port Preservation*, it maintains a session entry as:

$$\{[C : c \leftrightarrow V : c] \leftrightarrow [S : s]\} \tag{2}$$

The attacker A has to probe the ephemeral port space to determine the port used by C with the steps in Figure 3. There will be two cases:

a). Hitting the port c: First, A sends a Probe packet (i.e., SYN) packet from source port c to S:s via the VPN tunnel, i.e., {A:c→S:s, SYN}. Due to port c already being used by C, the VPN server V maps the packet to a different external port c′, resulting in the entry:

$$\{[A : c \leftrightarrow V : c'] \leftrightarrow [S : s]\} \tag{3}$$

The attacker then send a Verify packet (i.e., spoofed SYN/ACK) impersonating S to V with port c outside the VPN tunnel, i.e., {S:s→V:c, SYN/ACK}. When the SYN/ACK arrives, it will match entry (2) and will be forwarded to C.

b). Hitting an unused port a: The attacker sends the {A:a→S:s, SYN} packet through the VPN tunnel first. V

will create an entry (4) that keeps the source port, as there is no collision. Then, when the spoofed {S:s→V:a, SYN/ACK} arrives at the VPN server, it will be forwarded back to the attacker according to entry (4).

$$\{[A:a \leftrightarrow V:a] \leftrightarrow [S:s]\} \quad (4)$$

To reduce detectability, the attacker can set a low TTL for Probe packets, ensuring they do not reach the server. By scanning the full ephemeral port range, the attacker can infer all active sessions, violating the non-interference property.

2) Obtaining SEQ and ACK Numbers: After determining the source port c, the attacker attempts to obtain the sequence and acknowledgment numbers of the victim connection recorded in entry (5). Earlier approaches primarily deduced these values by exhaustively exploring the entire 4G spectrum via side channels [12], [17]–[19], which is time-consuming and unreliable. In this work, we find that vulnerabilities exist in connection tracking frameworks that change the state of the session entries when receiving TCP RST packets without sufficiently verifying the sequence numbers. Here we use Netfilter to illustrate our attack. Other frameworks (e.g., PF, IPFilter) exhibit slight variations in state management and attack details (see Section IV-A).

$$\{[C:c \leftrightarrow V:c] \leftrightarrow [S:s],\ tcp = ESTABLISHED,\ timeout = 432000s\} \quad (5)$$

Before the patch [33], upon receiving an in-window (a value close to 2^{16} in our test) TCP RST, Netfilter will directly transfer the session state from *ESTABLISHED* (432,000s timeout) to *CLOSE* (10s timeout), which is very vulnerable to blind TCP reset attacks. After the patch, Netfilter will keep the state of the session as *ESTABLISHED*, decrease the timeout to 10 seconds, and wait for the challenge ACK from the endpoint (according to RFC 5961 [34]) to restore the timeout to 300 seconds, which is much harder for attackers to abuse.

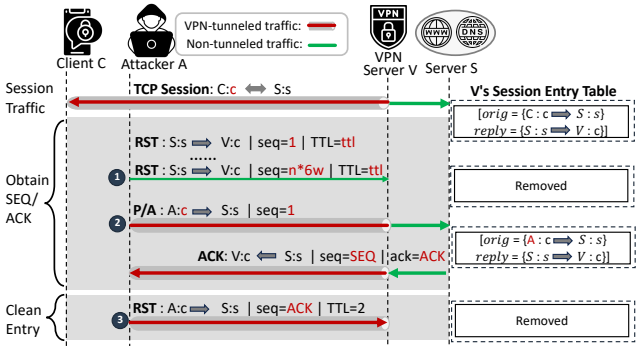

Fig. 4. Obtaining SEQ and ACK numbers

We propose a new method to bypass the verification as shown in Figure 4. ❶As Netfilter will only check whether the RST sequence number is within the challenge ACK window, the attacker can remove the victim session entry by sending spoofed TCP RST packets with varying sequence numbers as the target server, i.e., {S:s→V:a, RST, seq=i, TTL= ttl}, to the VPN server outside the VPN tunnel. We set the sequence number interval to 60000 to ensure that one of them will be located within the acceptable range, and the attacker only needs to send near 71k RST packets, which takes only seconds for modern machines.

To bypass the patch [33], the attacker can prevent triggering challenge ACK via controlling the TTL of the RST packets. It can probe the hop distance to the VPN server with tools (e.g., Traceroute), and then specify it in the RST packets. Upon arrival, the in-window RST packet will first incur the state transition of the victim session entry and then be dropped as the TTL value has decreased to 0, thus no challenge ACK from the victim client will be triggered anymore. The entry (5) will remain in *ESTABLISHED* while its expiration *timeout* will decrease to 10 seconds, as shown in entry (6). After its timeout, the entry will be completely removed.

$$\{[C:c \leftrightarrow V:c] \leftrightarrow [S:s],\ tcp = ESTABLISHED,\ timeout = 10s\} \quad (6)$$

❷Then the attacker replaces the victim client by constructing a new session entry (7) via sending a data packet with its own private IP address to the remote server with arbitrary sequence and acknowledgment numbers, e.g., {A:c→S:s, PUSH/ACK, seq=1, ack=1}.

$$\{[A:c \leftrightarrow V:c] \leftrightarrow [S:s],\ tcp = ESTABLISHED,\ timeout = 300s\} \quad (7)$$

After translation by the VPN server, the packet is routed to the target server. From the perspective of the remote server, this packet matches the victim session and will incur an ACK packet, i.e., {S:s→V:c, ACK, seq=*SEQ*, ack=*ACK*} back with the server's exact *SEQ* and *ACK* [35]. When the ACK packet arrives at the VPN server, it will be translated and routed to the attacker (entry (7)), allowing it to obtain the *SEQ* and *ACK* directly.

3) Hijacking Active TCP Sessions: With these values, the attacker can directly launch a TCP DoS attack by sending a TCP RST packet to the remote server, or it can imitate the victim client to send requests to the remote server and hijack the responses. ❸In addition, to inject forged responses into the victim client, the attacker needs to first remove its session entry (7) by sending a TCP RST (Step 3 in Figure 4). It then continuously sends forged responses to the VPN server impersonating the target server outside the tunnel, i.e., {S:s→V:c, PUSH/ACK, seq=*SEQ*, ack=*ACK*}. These packets are initially dropped silently due to the removed session entry. Once the victim client launches the next request and restores the entry, the forged responses will be forwarded to the victim client, thus achieving TCP injection.

C. DNS Hijacking Attack

According to [12] and our empirical studies, when a client is visiting a specific website on mainstream browsers, once the domain entry expires in the browser's DNS cache, ordinary user actions (e.g., loading in-page subresources or refreshing pages) can periodically trigger DNS resolution on the target domain name. For instance, after the DNS cache entry for a domain name (a.com) expires, accessing a resource within the web page (a.com/resources) dispatches a DNS request. We summarize the DNS cache behavior of four widely used browsers across different desktop OSes in Table I. Overall, most browsers either follow the DNS response TTL

or apply a short fixed caching window (e.g., 60 s), which opens a recurring window for an attacker to race and inject spoofed DNS responses. In contrast to previous DNS cache poisoning attacks targeting public DNS resolvers [23], [24], as the VPN server does not provide a DNS cache, our DNS hijacking attack does not aim to contaminate caches, but to inject forged responses into victims' DNS queries as Tolley's work [12]. Although the impact may be relatively less severe compared to cache poisoning, considering the capabilities and threat model of an off-path attacker, this attack's effectiveness already demonstrates its harmful potential.

TABLE I
DNS CACHE BEHAVIOR OF DIFFERENT BROWSERS.

OS	Browser	Version	DNS Cache Timeout
Windows 11	Chrome	143.0.7499.170	60s
	Edge	143.0.3650.96	60s
	Firefox	146.0.1	$\geq$ 660s
macOS 26.1	Chrome	143.0.7499.170	$\max(60s, \text{Response's TTL})$
	Edge	143.0.3650.80	Response's TTL
	Safari	21622.2.11.11.9	$\approx$ Response's TTL $\times$ 1.5
	Firefox	146.0.1	660s
Ubuntu 22.04	Chrome	143.0.7499.192	Response's TTL
	Edge	143.0.3650.96	Response's TTL
	Firefox	146.0.1	660s

Besides identifying the correct TxID, the attacker must also determine whether the victim is visiting the target domain. We follow the similar steps as [12] that the attacker first performs TCP inference (Section III-B1) to detect long-lived TCP sessions to IPs associated with the target domain. If TCP sessions exist, the attacker can then ascertain that the victim client is likely accessing a website associated with the target domain. As it is quite possible for the browser to launch DNS requests of the domain periodically, the attacker can start to scan the source port range to launch a DNS hijacking attack on the target domain name by following the two steps:

1) Making Inferences about Active DNS Sessions: Similar to Section III-B1, the attacker needs to determine the source port of the DNS request from the user to the target DNS resolver. The VPN server will maintain a session entry (e.g., entry (2)) for the user's DNS request. The attacker can scan the entire port number space following the same procedure in Figure 3. The difference is that the `Probe` and `Verify` packets are UDP packets with the guessed source ports. After the probing and verifying process, the attacker can determine the source port `c` used by the victim's DNS request.

2) Injecting DNS Responses via Brute Forcing TxIDs: DNS packets contain a 16-bit transaction ID (`TxID`), requiring the attacker to brute-force up to 65k possible values by sending spoofed responses to the inferred UDP session. This is feasible with modern machines with sufficient bandwidth and computing power. In our case study (Section IV-D), the attacker can scan the full `TxID` space in 4.27 seconds on average when the DNS request timeout is 10 seconds.

Practical Considerations. The feasibility of the attack hinges on three challenges: (1) The spoofed DNS response must precede the legitimate response from the DNS resolver. To enlarge the time window, the attacker can launch a DoS attack against the target DNS resolver by flooding it with a large number of spoofed DNS requests or rerouting its traffic to a black hole by leveraging the ICMP redirect mechanism [12], [24], [36]. (2) Users often initiate multiple DNS requests simultaneously, creating multiple session entries in the VPN server. The attacker can infer all active sessions in use and inject spoofed DNS responses into each session with fake answers to the target domain name in parallel by manipulating multiple machines (discussed in Section V-A). (3) The injection must be completed before the expiration of the DNS request timeout. After our empirical case studies in Section IV-D, we confirm that the attacker is frequently able to finish the injection before the timeout.

IV. REAL-WORLD EMPIRICAL STUDIES

In this section, we first explore the vulnerabilities in different connection tracking frameworks from Linux and FreeBSD. Then we take empirical case studies to evaluate the effectiveness of the attacks in real-world VPN networks.

Ethical Considerations. Our experiments are conducted with careful consideration of ethical problems. The analyses of different connection tracking frameworks are performed in a fully controlled local environment. For real-world VPN testing, we subscribe to services using two accounts per provider to carry out our non-malicious security research under its safe harbor provisions. The attacker and victim machines are both under our control. All DNS queries are directed to our own controlled resolvers, and target servers are self-hosted, except in the HTTP injection test, where a public web server is used. However, we only inject fake HTTP responses to our victim client device. To minimize any impact on VPN server availability, all traffic during DoS, port inference, and sequence inference phases is rate-limited to no more than 10 MB/s for a duration of 10 seconds. No disruption to VPN services is observed or reported. We also follow responsible disclosure procedures to notify affected vendors.

A. Connection Tracking Framework Analysis

We perform tests on open-source connection tracking frameworks from different OSes, i.e., Linux `Netfilter`, FreeBSD `PF`, `IPFW`, `IPFilter`, and `natd`, as they may be chosen by VPN providers. For each framework, we test if it violates the non-interference property and may be vulnerable to the attacks under different configurations. The results are shown in Table II.

Port Exhausting DoS Attack. We find that attackers can exhaust all usable external ports in `Netfilter` (with *Port Preservation*), `PF`, and `natd` under both port allocation strategies, while the attack is ineffective against `IPFilter` and `IPFW` due to inherent session entry limits. For `Netfilter` with *Port Preservation*, an attacker can occupy all external ports to the target server, causing victim packets to be dropped. However, under *Random Selection*, it becomes increasingly hard for the port selection algorithm to find unused source ports to create new session entries, making it difficult for an attacker to exhaust all ports. In `PF`, both allocation strategies are vulnerable. Under *Port Preservation*, an attacker can

TABLE II
TESTED RESULTS OF DIFFERENT CONNECTION TRACKING FRAMEWORKS.

System Setup			Port Exhausting DoS Attack		TCP Hijacking Attack			DNS Hijacking Attack
OS	**Connection Tracking Framework**	**Source Port Allocation**	**External Ports Fully Occupiable**	**Vulnerable**	**RST Check**	**Expiration Timeout**	**Vulnerable**	**Vulnerable**
Linux	Netfilter	preservation	yes	✔	in-window check	10s	✔	✔
		random	no	✘	in-window check	10s	✘	✘
FreeBSD	PF	preservation	yes	✔	no check	90s	✔	✔
		random	yes	✔	no check	90s	✘	✘
	IPFilter	preservation	no	✘	no check	60s	✔	✔
		random	no	✘	no check	60s	✘	✘
	IPFW	preservation	no	✘	strict check	-	✘	✔
		random	no	✘	strict check	-	✘	✘
	natd	preservation	yes	✔	strict check	-	✘	✔
		random	yes	✔	strict check	-	✘	✘

exhaust all 65,535 external ports. With *Random Selection*, `PF` selects ports from 50001 to 65535. When these ports become fully occupied, a bug in the implementation causes subsequent packets to be sent with the internal IP address without NAT. This issue has been acknowledged and fixed by the maintainers. In `natd`, both strategies are also vulnerable. Under *Random Selection*, the full range from 32768 to 65535 can be consumed. A similar bug to that in `PF` is triggered afterward, causing packets to be sent with private IPs instead of being NATed. In contrast, `IPFilter` limits session entries to 30,000 under *Port Preservation* and 256 under *Random Selection*, while `IPFW` enforces a strict limit of 16,384 entries under both modes. These constraints effectively prevent port exhaustion attacks in both frameworks.

TCP Hijacking Attack. The attack can succeed if the framework adopts *Port Preservation* and lacks sufficient verification in state transition when receiving `RST` packets. We find `Netfilter`, `PF`, and `IPFilter` with *Port Preservation* are vulnerable. For `Netfilter`, if a TCP `RST` packet with a controlled `TTL` value has a sequence number within the challenge ACK window (around 2^{16}), the session entry remains in the *ESTABLISHED* state, but the timeout drops to 10 seconds. `PF` and `IPFilter` are even more vulnerable. They accept `RST` packets with arbitrary sequence numbers and transition the session to *CLOSE*, lasting 90 seconds in `PF` and 60 seconds in `IPFilter`, both shorter than the original *ESTABLISHED* timeout. FreeBSD has assigned a high-severity CVE (CVE-2023-6534) for the problem. In contrast, `IPFW` and `natd` strictly validate `RST` packet sequence numbers and are not vulnerable to this attack.

DNS Hijacking Attack. As stated in Section III-C1, if the attacker can infer the source port used by the victim's DNS query, it can inject malicious responses into it. Our method becomes feasible when the connection tracking framework employs *Port Preservation*. As a result, `Netfilter`, `PF`, `IPFilter`, `IPFW`, and `natd` with *Port Preservation* are all vulnerable to the attack.

B. Case Study of Port Exhausting DoS Attack

1) Experiment Setup: Our testbed includes four machines: attacker, victim client, and target server (all running Ubuntu 22.04), plus a VPN server selected from commercial providers. In this work, we investigate the feasibility of the attacks on 9 of the top 10 commercial VPN providers [26], excluding *Hide.me* due to service unavailability. For each provider, we choose a VPN server according to the application's suggestion and use OpenVPN as the default protocol. All machines are geographically separated in different regions to simulate real-world cases and are connected as depicted in Figure 1. The victim client is securely configured to route all traffic through the VPN server, following best practices.

Attack Procedure. We have two experiments in this attack. First, the target server runs an HTTPS webpage on TCP port 443 using Nginx. The attacker will then occupy all the source ports of the VPN server to the target server, following Figure 2. The victim then attempts to access the web server. Second, we configure the target server as the victim's DNS resolver and have the attacker occupy all source ports to it. If the victim cannot resolve domains anymore, the attack is deemed successful. For each VPN provider, we repeat the attacks 10 times and record the time cost and success rate.

2) Experiment Results: Table III summarizes the attack results, listing time cost (in seconds) and success rate. For instance, the column of TCP DoS attack on ExpressVPN (i.e., $4.14|\frac{10}{10}$) shows success in 4.14 seconds with a 100% rate.

For TCP DoS, we can successfully exhaust all ephemeral ports of VPN servers from ExpressVPN, PIA, PrivateVPN, IPVanish, CyberGhost, and Windscribe within nearly 4 seconds and effectively obstructed the TCP connections directed to the target server with success rates over 90%. NordVPN, Surfshark, and ProtonVPN deploy proxy-like technologies to manage outgoing traffic [37]–[40]. Specifically, NordVPN and Surfshark apply the proxy mechanism only to popular destination ports such as 443 and 80, while other ports (e.g., FTP port 21) are not covered. In contrast, ProtonVPN applies proxying to all destination ports. We find that this proxy-based design can effectively prevent the port exhaustion attack.

Similarly, the DNS DoS attack succeeded against PIA, PrivateVPN, CyberGhost, and Windscribe, disrupting resolution in 4.24 seconds with over 90% success. ExpressVPN, NordVPN, Surfshark, and IPVanish have special DNS settings that the victim's DNS queries will not be forwarded to the

TABLE III
ATTACK EVALUATION OF DIFFERENT VPN PROVIDERS.

VPN Information	Measurement Result			Port Exhausting DoS Attack		TCP Hijacking Attack		DNS Hijacking Attack		
VPN Provider	Session Limit	Source Port Assignment	RST Check	TCP DoS	DNS DoS	HTTP Injection	FTP Hijacking	5s timeout	10s timeout	15s timeout
ExpressVPN	no	preservation	in-window, 10s	$4.14\|\frac{10}{10}$	N/V	$74.62\|\frac{6}{10}$	$21.45\|\frac{8}{10}$	N/V	N/V	N/V
NordVPN	no	preservation	in-window, 10s	N/V	N/V	N/V	$86.75\|\frac{7}{10}$	N/V	N/V	N/V
	yes	random	strict-check							
PIA	no	preservation	in-window, 10s	$3.97\|\frac{10}{10}$	$4.30\|\frac{10}{10}$	$64.11\|\frac{7}{10}$	$22.77\|\frac{8}{10}$	$4.74\|\frac{2}{10}$	$7.17\|\frac{5}{10}$	$8.53\|\frac{7}{10}$
Surfshark	no	preservation	in-window, 10s	N/V	N/V	N/V	$24.70\|\frac{8}{10}$	N/V	N/V	N/V
	yes	random	strict-check							
PrivateVPN	no	preservation	in-window, 10s	$4.36\|\frac{10}{10}$	$4.05\|\frac{10}{10}$	$58.45\|\frac{7}{10}$	$21.38\|\frac{8}{10}$	$4.40\|\frac{2}{10}$	$7.16\|\frac{6}{10}$	$8.48\|\frac{7}{10}$
IPVanish	no	preservation	in-window, 10s	$3.98\|\frac{9}{10}$	N/V	$67.71\|\frac{7}{10}$	$23.44\|\frac{9}{10}$	N/V	N/V	N/V
CyberGhost	no	preservation	in-window, 10s	$3.99\|\frac{10}{10}$	$4.31\|\frac{9}{10}$	$67.48\|\frac{7}{10}$	$22.57\|\frac{9}{10}$	$4.86\|\frac{3}{10}$	$7.11\|\frac{6}{10}$	$7.72\|\frac{6}{10}$
ProtonVPN	yes	random	strict-check	N/V	N/V	N/V	N/V	N/V	N/V	N/V
Windscribe	no	preservation	in-window, 10s	$3.99\|\frac{10}{10}$	$4.28\|\frac{9}{10}$	$63.69\|\frac{6}{10}$	$26.63\|\frac{8}{10}$	$4.95\|\frac{1}{10}$	$8.45\|\frac{5}{10}$	$8.54\|\frac{8}{10}$

* N/V means that the VPN server is **Not Vulnerable** to this attack.

target server after setting the DNS resolver to it, but the client can still receive the DNS responses with a source IP address of the target server. While this setup protects against the attack, it can cause usability issues if users want to resolve customized domains with its own DNS resolver. ProtonVPN extends its proxy-like protections to UDP, fully blocking the attack.

C. Case Study of TCP Hijacking Attack

1) Experiment Setup: The experimental environment is consistent with that in Section IV-B1. We conduct two case studies: HTTP injection and FTP hijacking. For HTTP injection, we avoid scanning the entire Internet due to scale and ethical concerns. As reported by [41], approximately 13.9% of websites still do not default to HTTPS, leaving a non-trivial attack surface. Following prior works [17], [19], we select a well-known financial website, *ANONYMOUS.com*, as our target. For FTP hijacking, we set our target server to provide file-downloading services through FTP (with `vsftpd` 3.0.3).
Attack Procedure. For HTTP injection, the victim's browser maintains a long-lived HTTP session that periodically issues requests to fetch exchange rates. The attacker tries to inject forged responses with fake financial data with our method. For FTP hijacking, the victim logs into the FTP server and issues different FTP commands. The attacker will try to steal the victim's private files during the attack. For each VPN provider, we perform both attacks 10 times independently, renewing the session between the client and server after each run.

2) Experiment Results: **VPN Analysis.** 8 VPN providers are vulnerable under our threat model. ProtonVPN is unaffected due to its use of proxy technology, which employs *Random Selection* and strict TCP `RST` validation. NordVPN and Surfshark mitigate HTTP injection via proxying TCP port 80 traffic, but remain vulnerable to FTP hijacking.
Time Cost. As time costs across most VPNs follow similar patterns (except NordVPN), we report detailed results in Figure 5(a) using PIA as an example. For HTTP injection, identifying the client's source port takes 3.73 seconds (6.55 MB/s bandwidth), while obtaining sequence numbers takes 17.42 seconds on average. The most time-consuming phase is response injection, averaging 42.43 seconds due to the need to evict the attacker's own session and await the next victim request. The full attack completes in 64.11 seconds on average. For FTP hijacking, the three phases take 3.69, 17.15, and 2.11 seconds, respectively, resulting in a time cost of 22.77 seconds for the entire attack to get a private file from the server. This attack is faster as it does not require waiting for a new client request. Notably, NordVPN experiences high packet loss during the inference period. This leads to multiple retries and extended delays in determining the source port, resulting in increased time costs and a lower success rate.

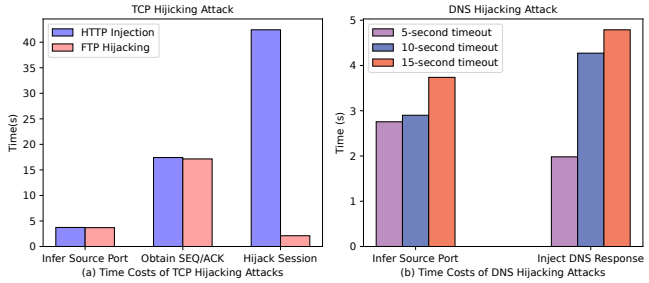

Fig. 5. Time cost of the attacks

Success Rate. For VPN providers where attack conditions are met (ExpressVPN, PIA, PrivateVPN, IPVanish, CyberGhost, Windscribe), HTTP injection succeeds with an average rate of 66.7%. Failures arise primarily from: (1) packet loss during probing; (2) the client receives valid data from the real server before injection, or (3) as the attacker will temporarily replace the original session entry, if the client sends a request during this period, the server will respond with an `RST` to terminate the connection. Figure 6 shows a snapshot of the attack result,

in which the exchange rates are manipulated to other forged values and may lead to wrong purchase or sale. FTP hijacking achieves an 81.3% success rate across 8 providers. Failures are mainly due to (1) packet loss and (2) high-frequency client communication, which refreshes session timeouts and prevents entry eviction. We further analyze this in Section V-A.

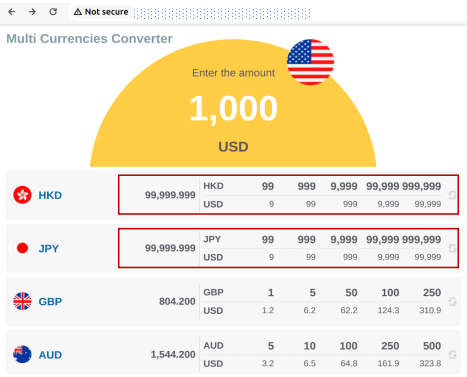

Fig. 6. The exchange rates on the web page after attack

D. Case Study of DNS Hijacking Attack

1) Experiment Setup: The setup of the experiment is also the same as that in Section IV-B1. To launch the DNS hijacking attack, the attacker needs to first mute the DNS resolvers to preclude premature DNS responses. To avoid impacting other users and raising ethical concerns, we chose to set the victim's DNS resolver to our own server.

Attack Procedure. We follow a similar methodology as Tolley's work [12]. Since forged DNS responses must arrive before query timeouts, which vary by OSes and browsers (5, 10, or 15 seconds), we ran 10 tests for each timeout to evaluate the attack. In each test, the victim VPN client issues a single DNS lookup using `nslookup` with a specified timeout. Half a second later, the attacker starts the injection script to infer the DNS source port and inject responses with varying `TxIDs`.

2) Experiment Results: **VPN Analysis.** As stated in the DNS DoS experiments, ExpressVPN, NordVPN, Surfshark, and IPVanish employ special DNS settings, which prevent the attack. ProtonVPN adopts *Random Selection*, also rendering it safe. In contrast, PIA, PrivateVPN, CyberGhost, and Windscribe are vulnerable to inference of DNS sessions.

Time Cost and Success Rate. Table III summarizes the attack performance across VPNs. We illustrate the time cost of PIA in Figure 5(b) as the others share similar trends with it. For a 5-second DNS timeout (e.g., Edge on Windows 11), it took on average 2.76 seconds to infer the source port and 1.98 seconds to inject the correct `TxID`, using 2.43 MB/s bandwidth, totaling 4.74 seconds with a 20% success rate. For the 10-second timeout (e.g., Firefox on Ubuntu 20.04), the attacker can succeed with an average time cost of 2.90 seconds and 4.27 seconds, respectively, and a success rate of 50%. For the 15-second timeout (e.g., Safari on macOS 14.1.2), the attacker can succeed with an average success rate of 70%, and the time costs of the two steps are 3.74 and 4.79 seconds. The attacker can achieve a higher success rate with a longer DNS query timeout, as it has more time to infer the correct source port and scan through the possible `TxID` space before the UDP socket is closed by the victim client.

V. Discussion and Countermeasures

A. Real-world Considerations

Impacts of Multiple Sessions. Modern browsers often establish multiple concurrent TCP sessions to speed up page loading. However, these additional sessions are usually short-lived and exert minimal impact on deducing the targeted long-lived TCP connection. It may be possible that the client maintains multiple long-lived TCP sessions with the server, and the same obstacle exists in the DNS hijacking attack. For example, the attacker attempts to inject forged DNS responses for `a.com`, but meanwhile the client also queries for `b.com`. The attacker can infer the source ports of all active sessions and inject responses into the inferred sessions. We have tested the DNS hijacking attack when the client initiates 10 different DNS queries with a 10-second timeout. The attacker controls 5 machines to perform inference and injection in parallel, and the attack succeeds in 4 of the 10 tests.

Impacts of Different Traffic Frequencies. In TCP hijacking attacks, the attacker must first remove the victim's session entry at the VPN server, which typically expires after a 10-second timeout. However, if the server receives packets matching the session, the timeout is refreshed, forcing the attacker to retry. To assess this effect, we evaluate the FTP hijacking attack under varying client-server communication intervals (e.g., 4, 8, 12 seconds). Each configuration is tested 10 times. Results show that when the interval is below 10 seconds, the attack consistently fails due to the session entry being refreshed. For intervals exceeding 10 seconds, the attack succeeds with an average success rate of 85%. As the interval increases, the time cost decreases, since the attacker requires fewer attempts and less waiting time.

B. Countermeasures

Responsible Disclosure. We have disclosed the vulnerabilities to Linux, FreeBSD, and affected VPN providers. FreeBSD has released an announcement confirming the vulnerabilities and assigned a high-severity CVE. Maintainers of Linux `Netfilter` also acknowledge the possible attack and are discussing with us about the patches. They suggest configuring proper firewall rules to mitigate the attack. Six VPN vendors have confirmed the vulnerability and rewarded us with their bug bounty policies. We are working with them to mitigate the attacks. The other 2 VPN vendors are still investigating the vulnerabilities. In total, our responsible disclosure has resulted in 19 assigned CVE/CNVD identifiers (specifically CVE-2023-6534, CVE-2024-50751 to CVE-2024-50764, and CNVD-2025-{05945, 06966, 06970, 07025}). We also recommend our countermeasures to prevent the attacks.

Limiting Concurrent Sessions. The port exhaustion DoS stems from attackers monopolizing the VPN server's shared

source ports. We recommend VPN providers limit concurrent sessions to the same target by leveraging connection tracking limits [42], setting firewall rules to curb port scanning, or deploying DDoS protections like SYN-proxy, thereby preventing exhaustion of session entries.

Randomizing Port Allocation. It is recommended for the VPN servers to adopt *Random Selection*, which will preclude the attacker from deducing used ports through the port preservation side channel employed in our attacks, thereby obviating the TCP and DNS hijacking attacks. Specifically, upon creating new entries to record sessions, the VPN server can select a random unused source port and record the port translation in the session entries.

Enforcing Strict RST Checks. As the vulnerable connection tracking frameworks falsely transfer the states of session entries upon receiving illegal `RST` packets, it is also required to strictly check `RST` packets such that only those with exact sequence numbers can cause state transition of the session entries, thereby preventing attackers from removing the session entries intentionally to mitigate TCP hijacking attacks. FreeBSD has quickly released patches to fix the vulnerability with this suggestion. However, the maintainers of Linux Netfilter argue that it's hard for middleboxes to always track the states or values of endpoints. They are concerned that strict checks of `RST` packets may prevent the connection tracking table entries from being properly cleared, leading to resource exhaustion. We are still working with them to promote better patches to be applied in the kernel.

VI. Related Work

VPN Security. The security of VPN systems has been extensively investigated across platforms such as Android, iOS, and desktop systems [8], [10], [43], revealing issues such as traffic leakage and misconfiguration. Perta et al. [7] and Tolley et al. [12] demonstrated DNS and session hijacking attacks via malicious access points or in/on-path adversaries. Xue et al. [11] and Moratti et al. [13] exploited routing table manipulation to divert VPN traffic. More recently, Mixon-Baca et al. [14] introduced the *Port Shadow* attack, which exploits port collisions between a client's source port and the VPN server's listening port to infer VPN usage. In contrast, our work targets *client-to-external-server* sessions and exploits port collisions between VPN clients sharing the same connection tracking table. Furthermore, we uncover a distinct vulnerability in the handling of TCP `RST` packets that results in incorrect session state transitions, enabling effective inference and hijacking attacks. Moreover, our work introduces different attack effects (e.g., DoS). Unlike *Port Shadow*, our attack does not assume that the VPN server shares the same entry and exit IP address, making it applicable in more common VPN deployments.

NAT Security. Previous studies have investigated NAT behaviors for host enumeration [44]–[46]. Winemiller et al. [30] and Nguyen et al. [31], [32] proposed NAT DoS attacks that fill all the NAT table in hypervisors or Docker containers via a compromised machine. Feng et al. [47] demonstrated remote TCP DoS attacks against NAT networks. These attacks typically rely on predictable identifier fields or compromised internal hosts. Gilad et al. [16] showed that IP fragmentation can be used to bypass NATs for interception, while Herzberg et al. [20] inferred DNS source ports by pre-filling NAT tables using puppets. However, their method targets outdated NAT implementations. In contrast, our work targets VPN environments where connection tracking frameworks are shared among tenants. We exploit port collision behavior that arises *after* the victim session is established, without requiring puppets. Moreover, our selective session-level DoS does not rely on flooding the table, making it more stealthy.

Session Manipulation Attacks. TCP hijacking attacks have leveraged side channels such as challenge ACKs [17], IPID behaviors [19], timing patterns [15], [18], and packet sizes [48]. Yang et al. [25], [49] exploited the router vulnerability that totally disabled TCP window tracking, enabling TCP hijacking in Wi-Fi environments. Feng et al. [50] leveraged ICMP redirects to conduct Man-in-the-Middle attacks in Wi-Fi networks. Our work targets stricter VPN environments where connection tracking enforces state validation and reverse path checks. Furthermore, we uncover attacks without proximity to the victim and go beyond TCP hijacking to include port exhaustion DoS and DNS injection. For DNS manipulation, Man et al. [23], [24] inferred DNS ports via ICMP side channels, while Herzberg et al. and Zheng et al. [21], [51] used IP fragmentation for cache poisoning. In contrast, our method reveals a novel port collision channel within VPN servers, allowing a malicious VPN user to infer DNS source ports and mount hijacking attacks without network-level access or fragmentation.

VII. Conclusion

In this work, we uncover new off-path session manipulation attacks in VPN networks that malicious users can abuse the vulnerabilities that exist in the shared connection tracking table and improper state transition of session entry, which violates the non-interference property required for the VPN servers. We present our attacks on 5 mainstream connection tracking frameworks and 9 commercial VPN providers and demonstrate that off-path attackers can disrupt or hijack other client's TCP and DNS sessions. We confirm the vulnerabilities in all frameworks and 8 VPN providers. Our findings pose new challenges to the current understanding of real-world VPN's security and have led to 19 assigned CVEs/CNVDs through responsible disclosure. The authors have provided public access to their code at https://github.com/yyxRoy/vpn-attacks.

Acknowledgment

The work is in part supported by the National Science Foundation for Distinguished Young Scholars of China under Grant 62425201, the Science Fund for Creative Research Groups of the National Natural Science Foundation of China under Grant 62221003, and the National Natural Science Foundation of China under Grant 62132011. Ke Xu is the corresponding author.

References

[1] P. Matic, "Vpn: A decade's worth of growth," https://www.pcmatic.com/news/vpn_report/, Accessed July 2023.

[2] R. Ramesh, L. Evdokimov, D. Xue, and R. Ensafi, "VPNalyzer: Systematic Investigation of the VPN Ecosystem," in *Proc. NDSS*, 2022.

[3] J. A. Goguen *et al.*, "Security policies and security models," in *1982 IEEE Symposium on Security and Privacy*, 1982, pp. 11–11.

[4] B. Schneier and Mudge, "Cryptanalysis of microsoft's point-to-point tunneling protocol (pptp)," in *Proc. 5th ACM CCS*, 1998, p. 132–141.

[5] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, "Imperfect forward secrecy: How diffie-hellman fails in practice," in *Proc. 22nd ACM CCS*, 2015, p. 5–17.

[6] D. Felsch, M. Grothe, J. Schwenk, A. Czubak, and M. Szymanek, "The dangers of key reuse: Practical attacks on IPsec IKE," in *Proc. 27th USENIX Security Symp.*, 2018, pp. 567–583.

[7] V. C. Perta, M. V. Barbera, G. Tyson, H. Haddadi, and A. Mei, "A glance through the vpn looking glass: Ipv6 leakage and dns hijacking in commercial vpn clients," *Proc. on PETS*, pp. 77 – 91, 2015.

[8] M. Ikram, N. Vallina-Rodriguez, S. Seneviratne, M. A. Kaafar, and V. Paxson, "An analysis of the privacy and security risks of android vpn permission-enabled apps," in *Proc. 2016 IMC*, 2016, p. 349–364.

[9] M. T. Khan, J. DeBlasio, G. M. Voelker, A. C. Snoeren, C. Kanich, and N. Vallina-Rodriguez, "An empirical analysis of the commercial vpn ecosystem," in *Proc. IMC*, 2018, p. 443–456.

[10] R. Ramesh, L. Evdokimov, D. Xue, and R. Ensafi, "VPNalyzer: Systematic Investigation of the VPN Ecosystem," in *Proc. NDSS*, 2022.

[11] N. Xue, Y. Malla, Z. Xia, C. Pöpper, and M. Vanhoef, "Bypassing tunnels: Leaking vpn client traffic by abusing routing tables," in *Proc.USENIX Security Symp.*, 2023, pp. 5719–5736.

[12] W. J. Tolley, B. Kujath, M. T. Khan, N. Vallina-Rodriguez, and J. R. Crandall, "Blind in/on-path attacks and applications to vpns," in *Proc. 30th USENIX Security Symposium*, 2021, pp. 3129–3146.

[13] D. Cronce and L. Moratti, "Tunnelvision: A local network vpn leaking technique that affects all routing-based vpns," https://www.tunnelvisionbug.com//, Accessed August 2024.

[14] B. Mixon-Baca *et al.*, "Attacking connection tracking frameworks as used by virtual private networks," in *Proc. on PETS*, 2024.

[15] Z. Qian and Z. M. Mao, "Off-path tcp sequence number inference attack - how firewall middleboxes reduce security," in *Proc. IEEE Symp. Secur. Privacy*, 2012, pp. 347–361.

[16] Y. Gilad and A. Herzberg, "Fragmentation considered vulnerable," *ACM Trans. Inf. Syst. Secur.*, 2013.

[17] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel, "Off-path tcp exploits: Global rate limit considered dangerous," in *25th USENIX Security Symposium*, 2016, pp. 209–225.

[18] W. Chen and Z. Qian, "Off-path tcp exploit: How wireless routers can jeopardize your secrets," in *USENIX Security Symp.*, 2018, pp. 1581–1598.

[19] X. Feng, C. Fu, Q. Li, K. Sun, and K. Xu, "Off-path tcp exploits of the mixed ipid assignment," in *Proc. ACM CCS*, 2020, p. 1323–1335.

[20] A. Herzberg and H. Shulman, "Security of patched dns," in *Computer Security – ESORICS 2012*. Springer Berlin Heidelberg, 2012, pp. 271–288.

[21] ——, "Fragmentation considered poisonous, or: One-domain-to-rule-them-all. org," in *Proc. 2013 CNS*. IEEE, 2013, pp. 224–232.

[22] Y. Gilad, A. Herzberg, and H. Shulman, "Off-path hacking: The illusion of challenge-response authentication," *IEEE Security & Privacy*, pp. 68–77, 2014.

[23] K. Man, Z. Qian, Z. Wang, X. Zheng, Y. Huang, and H. Duan, "Dns cache poisoning attack reloaded: Revolutions with side channels," in *Proc. 2020 ACM CCS*, 2020, pp. 1337–1350.

[24] K. Man, X. Zhou, and Z. Qian, "Dns cache poisoning attack: Resurrections with side channels," in *Proc. 2021 ACM CCS*, 2021, pp. 3400–3414.

[25] Y. Yang, X. Feng, Q. Li, K. Sun, Z. Wang, and K. Xu, "Exploiting sequence number leakage: Tcp hijacking in nat-enabled wi-fi networks," in *Proc. NDSS*, 2024.

[26] S. Migliano, "The best vpn services of 2024," https://www.top10vpn.com/best-vpn/, Accessed March 2024.

[27] B. Ford *et al.*, "NAT Behavioral Requirements for TCP," RFC 5382, IETF, Tech. Rep. 5382, 2008.

[28] Q. Lone, A. Frik, M. Luckie, M. Korczyński, M. van Eeten, and C. Gañán, "Deployment of source address validation by network operators: A randomized control trial," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 2361–2378.

[29] CAIDA, "State of ip spoofing," https://spoofer.caida.org/summary.php, Accessed March 2024.

[30] N. Winemiller, *NAT Denial of Service: An Analysis of Translation Table Behavior on Multiple Platforms*. Rochester Institute of Technology, 2012.

[31] S. Duc Nguyen, M. Mimura, and H. Tanaka, "Slow-port-exhaustion dos attack on virtual network using port address translation," in *2018 Sixth International Symposium on Computing and Networking (CANDAR)*. IEEE, 2018, pp. 126–132.

[32] S. D. Nguyen, M. Mimura, and H. Tanaka, "Leverage slow-port-exhaustion attacks by exploiting abnormal connections from iot devices and docker containers," *Journal of Information Processing*, pp. 486–494, 2022.

[33] Netfilter, "netfilter: conntrack: tcp: only close if rst matches exact sequence," https://github.com/torvalds/linux/commit/be0502a3, Accessed March 2024.

[34] R. R. Stewart *et al.*, "Improving TCP's Robustness to Blind In-Window Attacks," RFC 5961, IETF, Tech. Rep. 5961, Aug. 2010.

[35] J. Postel, "Transmission Control Protocol," RFC 793, IETF, Tech. Rep. 793, Sep. 1981.

[36] X. Feng, Q. Li, K. Sun, Z. Qian, G. Zhao, X. Kuang, C. Fu, and K. Xu, "Off-path network traffic manipulation via revitalized icmp redirect attacks," in *Proc. 31st USENIX Security Symp.*, 2022, pp. 2619–2636.

[37] J. Althouse, "Investigating surfshark and nordvpn with ja4t," https://medium.com/foxio/investigating-surfshark-and-nordvpn-with-ja4t-7bbf5a33aad0, Accessed March 2024.

[38] E. Norbutas *et al.*, "Multi-part tcp connection over vpn," https://patents.google.com/patent/US11956099B2/en, Accessed March 2024.

[39] Remy, "Protonvpn tcp accleration syn+ack spoofing analysis," https://remyhax.xyz/posts/protonvpn-tcp-hacks/, Accessed March 2024.

[40] A. Yen, "Increase vpn speeds by up to 400% with vpn accelerator," https://protonvpn.com/blog/vpn-accelerator/, Accessed March 2024.

[41] W3Techs, "Usage statistics of default protocol https for websites," https://w3techs.com/technologies/details/ce-httpsdefault, Accessed March 2024.

[42] S. H. Majumder, "Limiting maximum connections using iptables," https://www.baeldung.com/linux/iptables-limit-connections, Accessed March 2024.

[43] J. Wilson, D. McLuskie, and E. Bayne, "Investigation into the security and privacy of ios vpn applications," in *Proc. 15th ARES*, 2020, pp. 1–9.

[44] S. M. Bellovin, "A technique for counting natted hosts," in *Proc. 2nd ACM SIGCOMM Workshop on Internet Measurment*. ACM, 2002, p. 267–272.

[45] S. Mongkolluksamee, K. Fukuda, and P. Pongpaibool, "Counting natted hosts by observing tcp/ip field behaviors," *2012 IEEE ICC Conf.*, pp. 1265–1270, 2012.

[46] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable Secur. Comput.*, p. 93–108, 2005.

[47] X. Feng, Y. Yang, Q. Li, X. Zhan, K. Sun, Z. Wang, A. Wang, G. Du, and K. Xu, "Redan: An empirical study on remote dos attacks against nat networks," in *Proc. NDSS*, 2025.

[48] Z. Wang, X. Feng, Q. Li, K. Sun, Y. Yang, M. Li, G. Du, K. Xu, and J. Wu, "Off-path tcp hijacking in wi-fi networks: A packet-size side channel attack," in *Proc. NDSS*, 2025.

[49] Y. Yang, X. Feng, Q. Li, K. Sun, Z. Wang, A. Wang, and K. Xu, "Off-path tcp hijacking attack to nat-enabled wi-fi networks," *IEEE Transactions on Networking*, vol. 33, no. 6, pp. 3270–3285, 2025.

[50] X. Feng, Q. Li, K. Sun, Y. Yang, and K. Xu, "Man-in-the-middle attacks without rogue ap: When wpas meet icmp redirects," in *2023 IEEE Symposium on Security and Privacy*. IEEE, 2023, pp. 3162–3177.

[51] X. Zheng, C. Lu, J. Peng, Q. Yang, D. Zhou, B. Liu, K. Man, S. Hao, H. Duan, and Z. Qian, "Poison over troubled forwarders: A cache poisoning attack targeting DNS forwarding devices," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 577–593.