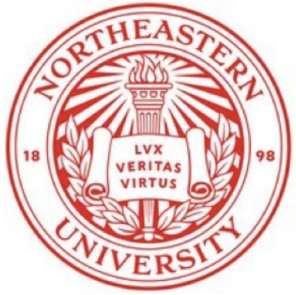


FUZZVPN: Finding Vulnerabilities in OpenVPN

Anqi Chen and Cristina Nita-Rotaru
Northeastern University

USENIX WOOT '25, Seattle

Northeastern University

VPNs

Goal: Privacy, security, anonymity; for resource access control, data protection, etc.

Usage: 95% American adults familiar with VPN; 46% use VPNs (link); 23.1% of internet users worldwide use a VPN (link).

Mechanism: Masking IP, encrypting the network traffic

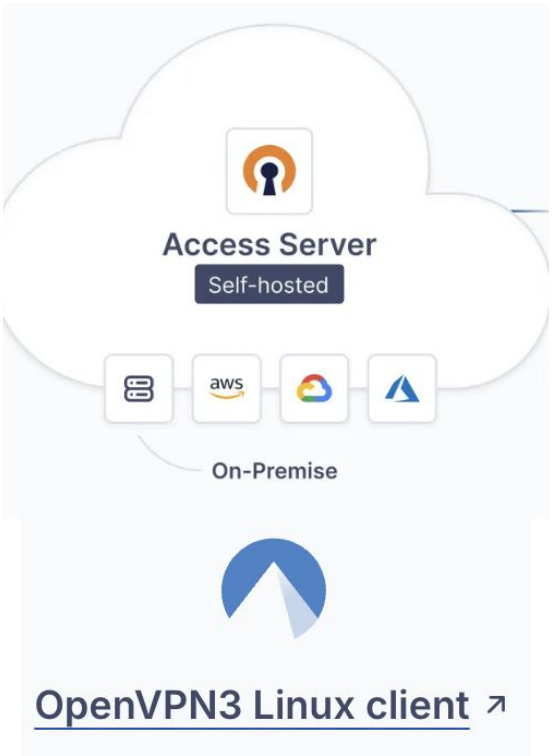
Software: ExpressVPN, NordVPN, Surfshark, AnyConnect

VPN protocols:

- **Open-source: OpenVPN, Wireguard, IKEv2, L2TP**
- **Proprietary: SSTP by Microsoft; AnyConnect by CISCO**

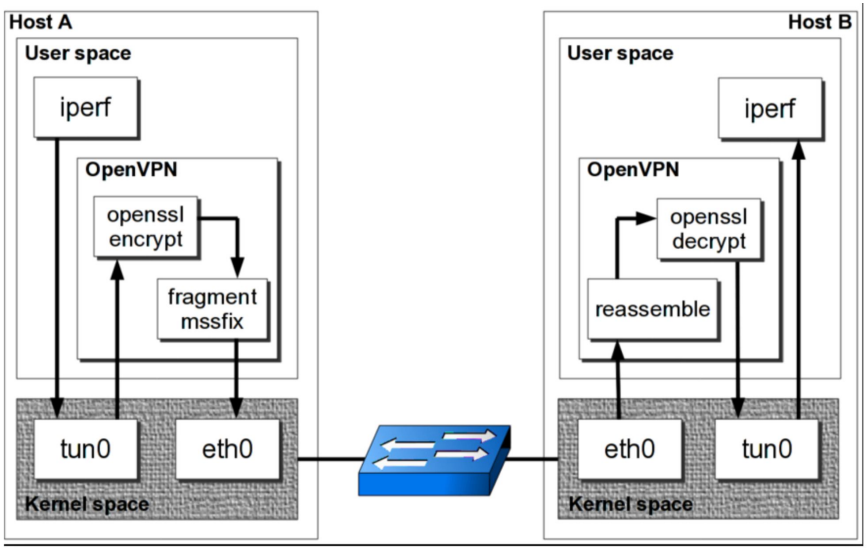
OpenVPN

- Most popular protocol of many VPN software: e.g. NordVPN and Proton VPN, ranked top2 popular in 2024
- **Open source: OpenVPN protocol (our focus), a customized security protocol based on TLS**
- Proprietary: Access Server (proprietary) and 5 OpenVPN Connect Client implementations for different OS's
- Many platforms**:** Linux, Windows, macOS, iOS, Android, and FreeBSD;
- ⚠️ **Not yet standardized by IETF (RFC draft is work-in-progress)**



How it Works

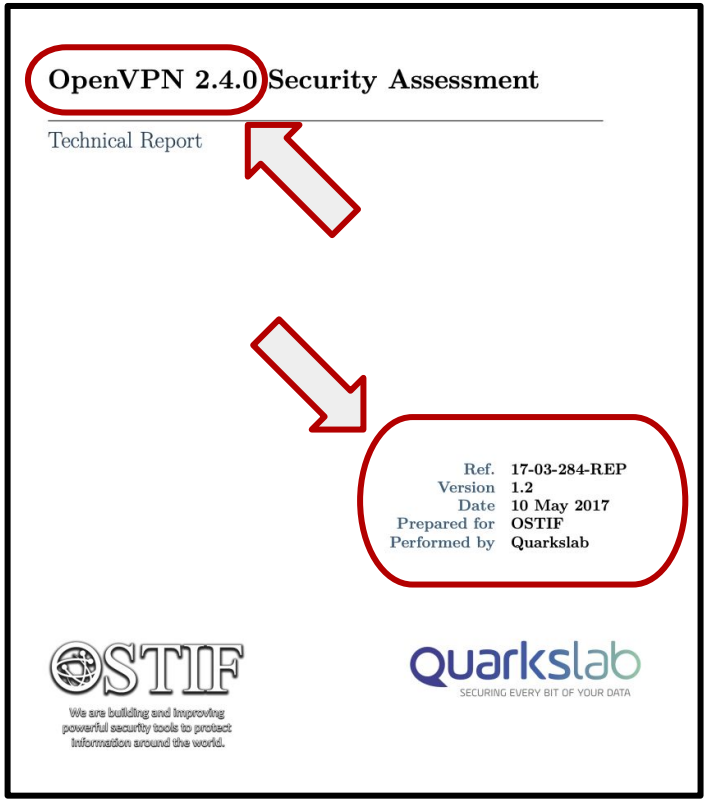
- Protects: TCP or UDP
- **Control channel**: three extra options tls-auth, tls-crypt, tls-crypt-v2
- **Data channel**: uses OpenSSL (or mbedTLS)
- Protects both link layer and network layer
  - Layer 2: a TAP device (layer 2 virtual network device) will be created;
  - Layer 3: a TUN device (layer 3 virtual network device) will be created
- Data channel offload: moving the data channel encryption/decryption to kernel space while keeping the control channel in user space (link , news in 2025 June)



picture source link

Why study OpenVPN security

- Broadly used
- Complex: many configuration options, execution modes, network layers
- Customized protocols, but also using TLS
- Not studied enough by previous work
  - Only a technical report 8 years ago studied the protocol itself on v2.4.0 while the newest is v2.6.14
  - the versions have updated a lot since then



Previous Work

- OpenVPN vulnerabilities
  - DoS attacks
  - Fingerprinting
  - Hijack TCP connections in a VPN tunnel
  - OS routing rule manipulation to bypass VPN
  - Memory bugs
- Network protocol fuzzing
  - Mutation-based (e.g. AFLNET);
  - Generation-based (e.g. PEACH);
  - Proxy-based (e.g. Bleem (not open source))
  - Limitations: OpenVPN's session-specific encryption, unclear MSC, or not open-source tool
- Formal methods
  - Wireguard formal verification work
  - Openvpn none since RFC is work-in-progress

Work in progress to create an RFC that documents the OpenVPN protocol

openvpn.github.io/openvpn-rfc/openv...

Our Contributions

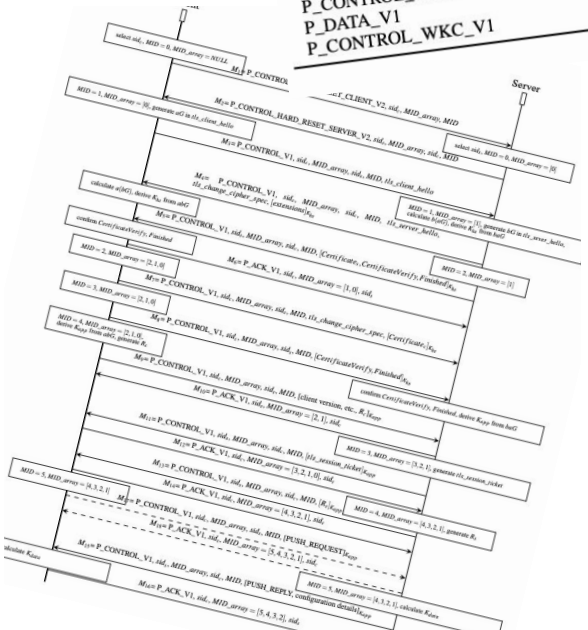
- **Created a detailed Message Sequence Chart (MSC)** of the OpenVPN handshake protocol in UDP and TCP mode respectively
- **Created FUZZVPN**, a tool for systematic testing of the protocol, including protocol-level attacks and malformed configuration testing
- **Found several new attacks**:
  - 2 are denial of service,
  - 17 are improper handling of configuration options,
  - 1 is server sending data prematurely,
  - and 1 performance degradation attack,
  - several ACK packet attacks, …

MSC Learning Methodology

We focus on the control channel handshake phase

Opcode Name	Value	Description
P_CONTROL_HARD_RESET_CLIENT_V2	7	Client hard reset
P_CONTROL_HARD_RESET_SERVER_V2	8	Server response
P_CONTROL_V1	4	Control packet
P_ACK_V1	5	Acknowledgement packet
P_DATA_V2	9	Data packet
P_CONTROL_SOFT_RESET_V1	3	Soft reset, a graceful transition from the old to new key
P_CONTROL_HARD_RESET_CLIENT_V3	10	Hard reset with client-specific *tls-crypt* key
P_CONTROL_HARD_RESET_CLIENT_V1	1	Deprecated
P_CONTROL_HARD_RESET_SERVER_V1	2	Deprecated
P_DATA_V1	6	Deprecated
P_CONTROL_WKC_V1	11	Deprecated

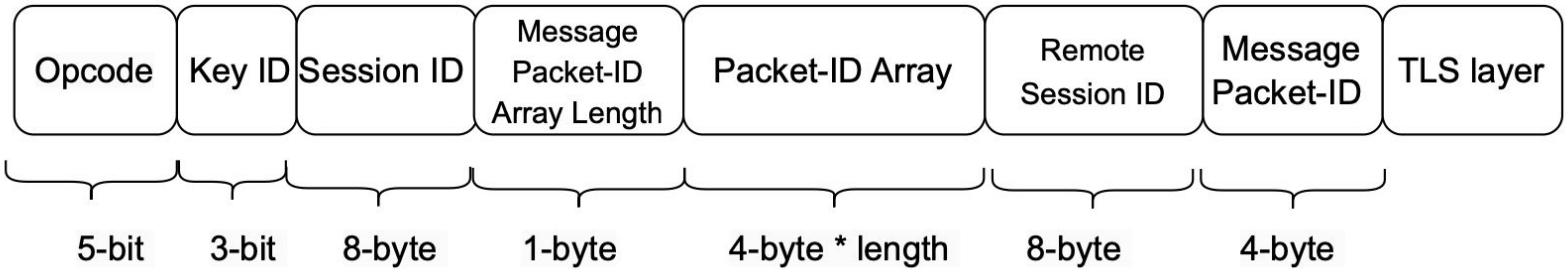
- **Passive learning**: learn the packet structure
- **Active learning**: learn the packet exchange sequence and the semantics



Passive Learning

- Capture control channel packets and see the structure
- Research available online information to understand field semantics
  - Opcode: the openvpn packet type
  - Key ID: data channel key's identifier
  - Packet-ID Array: for acknowledgement record

UDP mode OpenVPN control packet structure

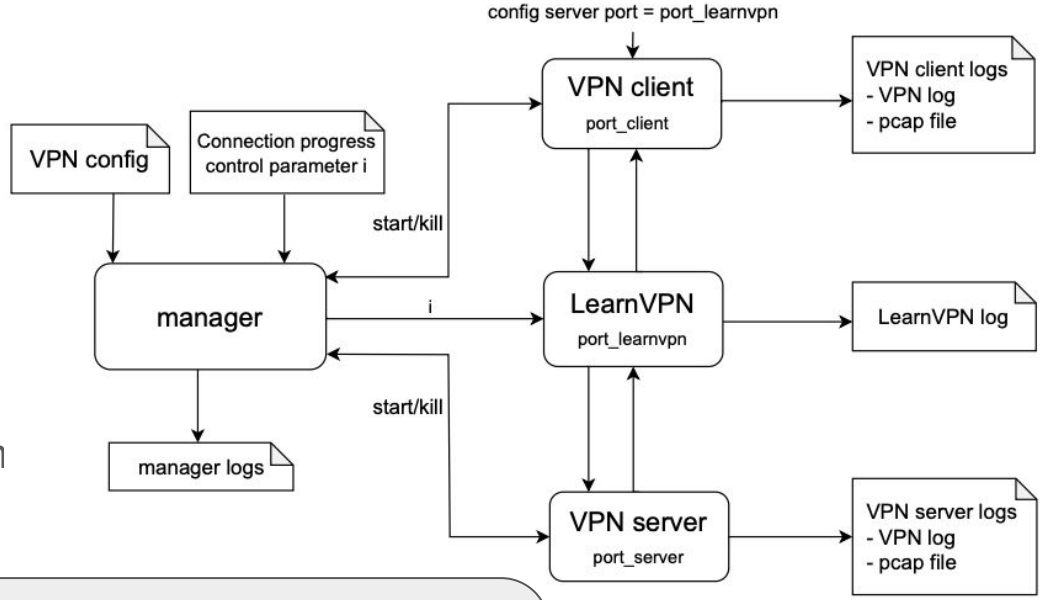


Active Learning

- MitM setup to control message sending progress and monitor the logs to understand internal behaviors

  E.g. Connection progress control parameter i = 1: only 1 control channel message is allowed to be forwarded

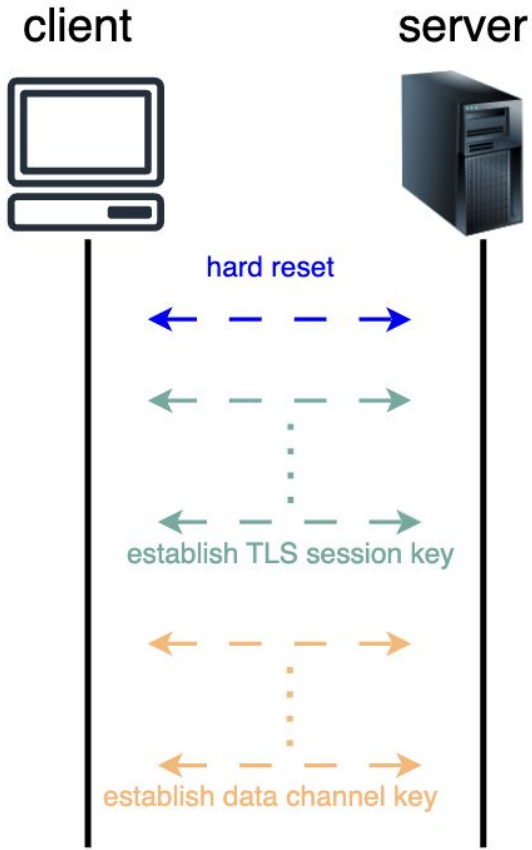
- Example of extracting information from logs to learn the internal behavior:




```
Log:
SSL state (connect): SSLv3/TLS read server certificate…
SSL state (connect): TLSv1.3 read server certificate
verify…
SSL state (connect): SSLv3/TLS read finished"
```

The OpenVPN Handshake MSC (UDP mode)

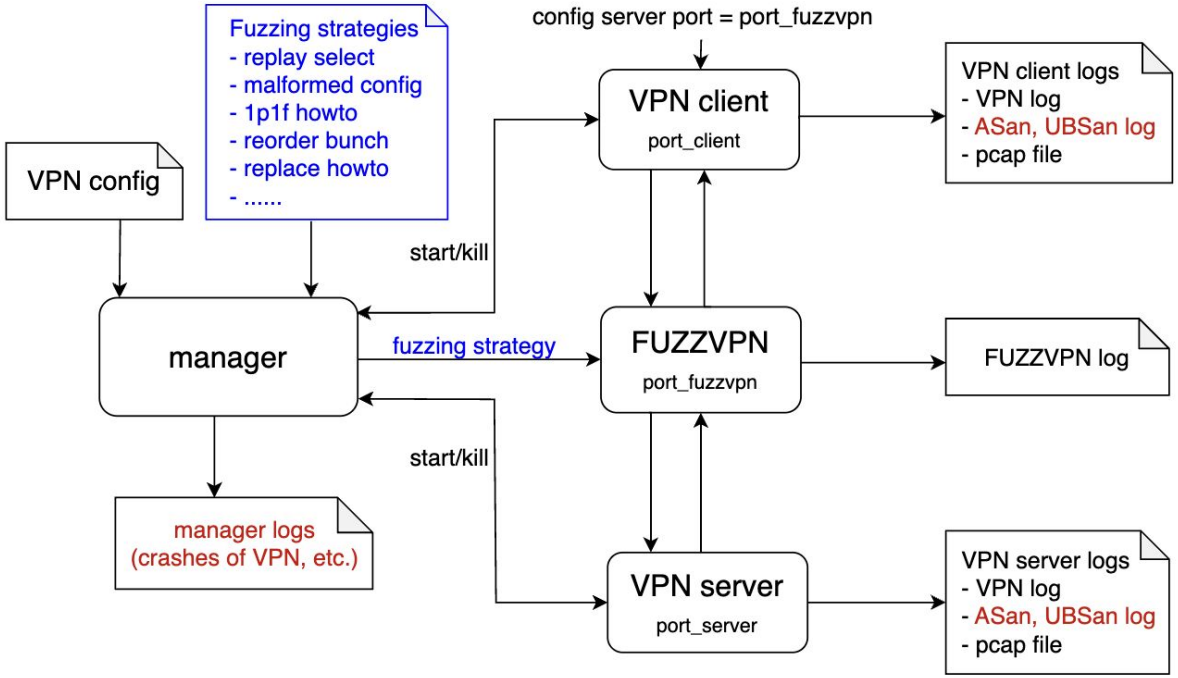
- **Control_Hard_Reset:**
  - 2 messages
  - Exchange server and client session id
- **TLS session key establishment:**
  - 6 messages
  - OpenVPN encapsulated TLS handshake using certificates
  - TLS Session key and session ticket issued
- **Data channel key establishment:**
  - 8 messages
  - Use the TLS key exporter method to generate data channel keys



Check our paper for the detailed MSC

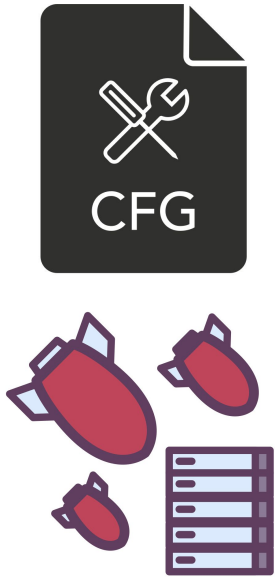
FUZZVPN

- Platform independent
- Protocol-logic aware
- Not invasive
- Test both the client and the server



Supported Testing Strategies

- Malformed configuration file testing
- Denial of service with replay packets
- Reordering of a sequence of control packets
- Field value modification
- Acknowledgement-related attacks
  - Remove one element out of the array
  - Replace one element of the array
  - Replace with an earlier **P_ACK_V1** packet:
  - Exchange *sidc* and *sids*
  - Drop **P_ACK_V1** packets



Detecting Bugs and Vulnerabilities

- Analysing the results
  - client and server execution logs (provided by OpenVPN)
  - client and server stderr (mostly ASan and UBSan output)
  - client and server-side packet traffic captured (by tcpdump
  - the FUZZVPN program output
- Look for keywords like **warnings or errors**
- Search for key sentence indicating connection success: Sc=“Initialization Sequence Completed"; and Ss=“Peer Connection Initiated with [Client Address]"
- Combine manually analysis to confirm the attacks

Experimental Setup

- Hardware:
  - two Intel Xeon Silver 4114 CPUs (each with 10 physical cores and Hyper-Threading enabled, providing a total of 40 logical processors)
  - x86_64 architecture
  - 188 GB of physical memory
  - ubuntu 24.04 operating system
- Docker version: 27.1.1; OpenVPN version: 2.6.12
- OpenVPN client and server configured with “–verb 9"
- Each experiment 60 seconds, during which either a crash happens or the manager will kill all the running programs
- 1000 scenarios. 5.5 hours for the UDP mode. 6 hours for the TCP mode

Configuration Input Validation

Example: port validation bug

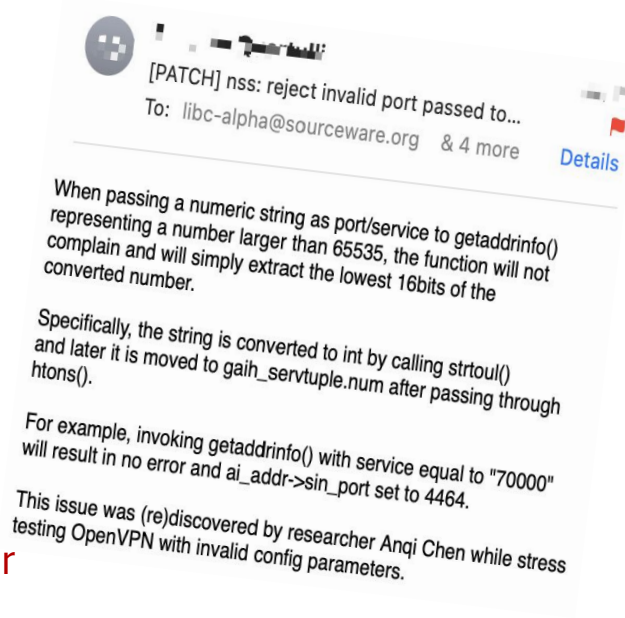
Scenario: in a configuration file input a port number (e.g. 70000) exceeding the valid port number range (0~65535 on Linux Systems)

Observation: OpenVPN implementation does not report any error, but instead just chooses another valid port 4464

(Mac version correctly reported the error, different code base)

Root cause: **a bug in the glibc/resolver** implementation of getaddrinfo, extracts the lowest 16bits of the converted number if it exceeds 65535. E.g. 70000 is 0x11170 and 4464 is 0x1170.

(Patch in progress)


[PATCH] nss: reject invalid port passed to...

To: libc-alpha@sourceware.org & 4 more Details

When passing a numeric string as port/service to getaddrinfo() representing a number larger than 65535, the function will not complain and will simply extract the lowest 16bits of the converted number.

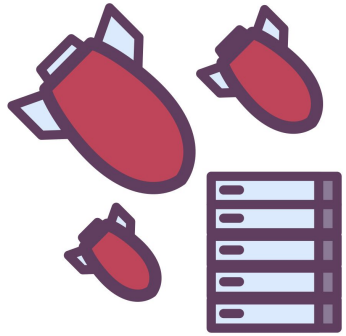
Specifically, the string is converted to int by calling strtoul() and later it is moved to gaih_servtuple.num after passing through htons().

For example, invoking getaddrinfo() with service equal to "70000" will result in no error and ai_addr->sin_port set to 4464.

This issue was (re)discovered by researcher Anqi Chen while stress testing OpenVPN with invalid config parameters.

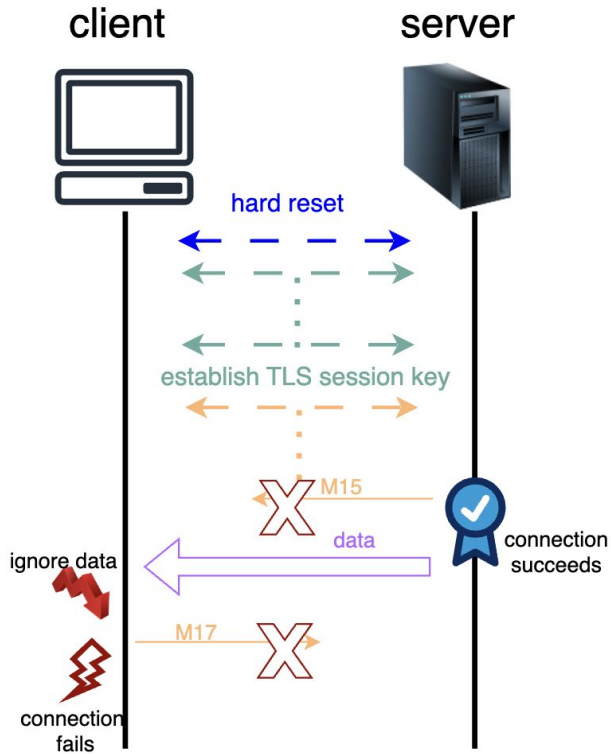
DoS Attacks

- The old flooding DoS attack by the P_CONTROL_HARD_RESET_CLIENT_V2 packets are patched in UDP mode, but not in TCP mode
- We found flooding attacks with two type of packets: P_CONTROL_V1 and P_ACK_V1
  - prevent the client's connection success;
  - also harms the server's availability and decrease other legitimate user's VPN data channel throughput.
- The tls-auth option claims to have replay protection, but it only logs warnings so cannot prevent the attack; also, it doesn't activate in TCP mode



Early Data Transmission

- Server starts sending DATA packets immediately after M15 (the last control packet from server including configuration details, etc.)
- Drop M15: the client will send several M17 [PUSH_REQUEST] packets
- Drop M15, M17 as well as the reply packets from the server: the client's connection will fail but the server can keep sending DATA packets
- **Root cause: the server already believes connection succeeds but the client has not yet connected successfully**
- (2nd scenario) Drop M15 packets to a threshold and then resumes sending it to allow the connection success: **then the client missed the data sent prematurely by the server**

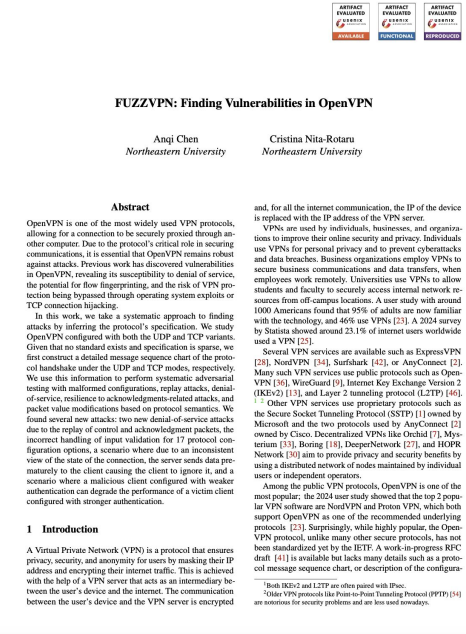


Conclusion

- We constructed a detailed MSC of the OpenVPN handshake protocol for both UDP, TCP
- We conducted a systematic adversarial testing with malformed configurations, replay denial-of-service attacks, resilience to acknowledgments-related attacks, and packet value and delivery modifications, etc.
- Found several new attacks: replay attacks, configuration option validation bugs, server prematurely sending data, ACK attacks

Code available at: https://zenodo.org/records/15644515

Check our paper to see more details :)



FUZZVPN: Finding Vulnerabilities in OpenVPN

Anqi Chen
Northeastern University

Cristina Nita-Rotaru
Northeastern University

Abstract

OpenVPN is one of the most widely used VPN protocols, allowing for a connection to be securely proxied through another computer. Due to the protocol's critical role in securing communications, it is essential that OpenVPN remains robust against attacks. Previous work has discovered vulnerabilities in OpenVPN, revealing its susceptibility to denial of service, the potential for flow fingerprinting, and the risk of VPN protection being bypassed through operating system exploits or TCP connection hijacking.

In this work, we take a systematic approach to finding attacks by inferring the protocol's specification. We study OpenVPN configured with both the UDP and TCP variants. Given that no standard exists and specification is sparse, we first construct a detailed message sequence chart of the protocol handshake under the UDP and TCP modes, respectively. We use this information to perform systematic adversarial testing with malformed configurations, replay attacks, denial-of-service, resilience to acknowledgments-related attacks, and packet value modifications based on protocol semantics. We found several new attacks: two new denial-of-service attacks due to the replay of control and acknowledgment packets, the incorrect handling of input validation for 17 protocol configuration options, a scenario where due to an inconsistent view of the state of the connection, the server sends data prematurely to the client causing the client to ignore it, and a scenario where a malicious client configured with weaker authentication can degrade the performance of a victim client configured with stronger authentication.

1 Introduction

A Virtual Private Network (VPN) is a protocol that ensures privacy, security, and anonymity for users by masking their IP address and encrypting their internet traffic. This is achieved with the help of a VPN server that acts as an intermediary between the user's device and the internet. The communication between the user's device and the VPN server is encrypted and, for all the internet communication, the IP of the device is replaced with the IP address of the VPN server.

VPNs are used by individuals, businesses, and organizations to improve their online security and privacy. Individuals use VPNs for personal privacy and to prevent cyberattacks and data breaches. Business organizations employ VPNs to secure business communications and data transfers, when employees work remotely. Universities use VPNs to allow students and faculty to securely access internal network resources from off-campus locations. A user study with around 1000 Americans found that 95% of adults are now familiar with the technology, and 46% use VPNs [23]. A 2024 survey by Statista showed around 23.1% of internet users worldwide used a VPN [25].

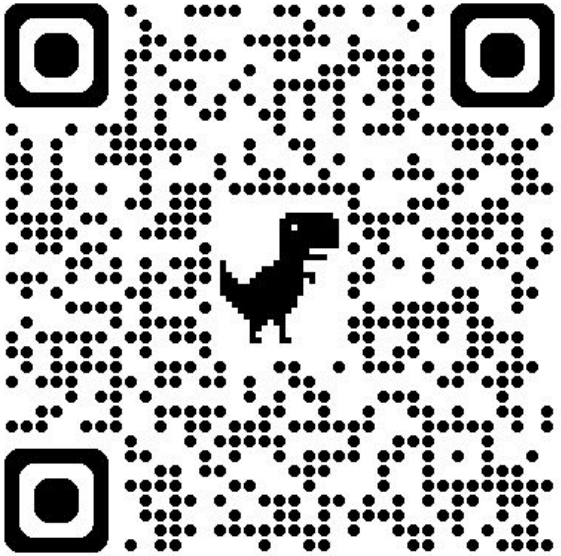
Several VPN services are available such as ExpressVPN [28], NordVPN [34], Surfshark [42], or AnyConnect [2]. Many such VPN services use public protocols such as OpenVPN [36], WireGuard [9], Internet Key Exchange Version 2 (IKEv2) [13], and Layer 2 tunneling protocol (L2TP) [46].[1,2] Other VPN services use proprietary protocols such as the Secure Socket Tunneling Protocol (SSTP) [1] owned by Microsoft and the two protocols used by AnyConnect [2] owned by Cisco. Decentralized VPNs like Orchid [7], Mysterium [33], Boring [18], DeeperNetwork [27], and HOPR Network [30] aim to provide privacy and security benefits by using a distributed network of nodes maintained by individual users or independent operators.

Among the public VPN protocols, OpenVPN is one of the most popular; the 2024 user study showed that the top 2 popular VPN software are NordVPN and Proton VPN, which both support OpenVPN as one of the recommended underlying protocols [23]. Surprisingly, while highly popular, the OpenVPN protocol, unlike many other secure protocols, has not been standardized yet by the IETF. A work-in-progress RFC draft [41] is available but lacks many details such as a protocol message sequence chart, or description of the configura-

[1] Both IKEv2 and L2TP are often paired with IPsec.
[2] Older VPN protocols like Point-to-Point Tunneling Protocol (PPTP) [54] are notorious for security problems and are less used nowadays.

Questions?

Thank you!



Scan to see our paper :)