Exploiting HTTP/3 0-RTT for IP Spoofing: Implementation, Analysis, and Mitigation Strategies

Yann Chicheportiche
Ariel University
Netanya, Israel
yanoukaxbox@gmail.com

Tal Malka
Ariel University
Ramat Gan, Israel
tal.coder@gmail.com

3rd Amit Dvir
Dept. of Computer & Software Engineering
Ariel Cyber Innovation Centre,
Ariel University, Israel
amitdv@ariel.ac.il

***Abstract*—HTTP/3, the latest evolution of the Hypertext Transfer Protocol, builds upon the QUIC transport protocol to enhance web performance through reduced latency, multiplexed streams, and integrated security features. However, this shift from TCP to QUIC introduces new security challenges that remain underexplored. In this paper, we analyze the security implications of HTTP/3's early data transmission feature, known as 0-RTT, and demonstrate how it can be abused to bypass IP-based access controls. We present a practical implementation of a 0-RTT spoofing attack in which an attacker reuses a legitimate session ticket to forge HTTP/3 requests with a spoofed source IP address. Our proof-of-concept shows that, even with server-side filtering and security logic, unauthorized access to protected resources becomes possible during the early stages of the QUIC handshake. We provide a detailed attack workflow, analyze server behavior, and discuss the limitations of current mitigation techniques. Finally, we propose concrete recommendations and future research directions to strengthen the resilience of HTTP/3 deployments against this class of attacks.**

I. Introduction

HTTP, or Hypertext Transfer Protocol, is the main communication protocol used on the Internet, allowing information to be exchanged between web browsers and servers. First created in the early 1990s, HTTP has improved over time to handle the growing needs of the Web. It started with the basic HTTP/1.0, advanced to the faster and more efficient HTTP/2, and now has reached HTTP/3, which brings even more improvements. Each version has aimed to make the web faster, more secure, and better able to support modern websites and apps. As a key part of how the Internet works, HTTP continues to evolve, playing an important role in improving online experiences. [1] [2]

HTTP/1, introduced in the early 1990s, laid the foundation for how web browsers and servers communicate. It was simple and effective for its time, using a request-response model to transfer data. However, it faced limitations as the web grew, such as inefficiency in handling multiple requests. Each file on a webpage—images, style-sheets, or scripts—required a separate connection, leading to delays and increased load times. Although HTTP/1.1, an improved version, introduced features such as persistent connections to address these issues, the protocol still struggled with modern demands for speed and scalability.

HTTP was originally designed without focusing on security and reliability; this is one of the main motivations behind the development of HTTP/2. HTTP/2, introduced in 2015, was a major step forward in addressing the limitations of HTTP/1. Built on the same basic request-response model, it introduced features such as multiplexing, which allowed multiple requests to be handled simultaneously over a single connection. This significantly reduced latency and improved website loading times. HTTP/2 also introduced header compression (a technique used to reduce the size of HTTP headers transmitted in requests and responses). The motivation is based on HTTP headers that carry metadata such as content type, language preferences, and cookies, which can be repetitive and bulky, especially in frequent communications. HTTP/2 uses an algorithm called HPACK, which compresses headers by identifying repeated values and replacing them with references to previously sent data.

However, despite its improvements, HTTP/2 still relied on TCP (Transmission Control Protocol), which brought challenges like head-of-line blocking (Head-of-Line Blocking: Head-of-line blocking occurs when the processing of a request or response is delayed because another item at the front of the queue must be handled first. In HTTP/2, even though multiple requests can be multiplexed over a single TCP connection, a problem (suck as packet loss) affects all streams on that connection, slowing down the delivery of other requests.

QUIC (Quick UDP Internet Connections) is a modern transport protocol developed by Google to overcome the limitations of traditional protocols like TCP [3]. Based on UDP, QUIC combines reliable data transfer and encryption into its core, optimizing communication while enhancing security. Key features include faster connection establishment with 0-RTT (The 0-RTT feature in QUIC allows a client to send application data before the handshake is complete. This is made possible by reusing negotiated parameters from a previous connection. To enable this, 0-RTT depends on the client remembering critical parameters and providing the server with a TLS session ticket that allows the server to recover the same information.) and 1-RTT handshakes, multiplexing without head-of-line blocking, and seamless connection migration between network interfaces, such as switching from WiFi to mobile data. By reducing latency and improving resilience to packet loss, QUIC meets the demands of modern Internet applications. Adopted as the foundation for HTTP/3, QUIC is becoming a crucial component of the web, enabling faster and more reliable online experiences.

HTTP/3, as defined in RFC 9114 [4], represents the latest evolution of the HTTP protocol, addressing many of the limitations found in HTTP/2. Built on the QUIC transport protocol, HTTP/3 eliminates head-of-line blocking by allowing each stream to operate independently, so packet loss in one stream does not delay the others. In addition, HTTP/3 introduces QPACK, an upgraded header compression scheme specifically designed to handle QUIC's characteristics, such as out-of-order delivery and packet loss, without sacrificing efficiency. These features, along with prioritized requests, connection migration, and integrated security, significantly reduce latency, improve bandwidth usage, and make HTTP/3 especially well suited for modern web applications, particularly in high-latency or loss networks. As adoption increases across browsers and servers, HTTP/3 is poised to become the new standard, providing faster, more reliable, and more efficient online experiences.

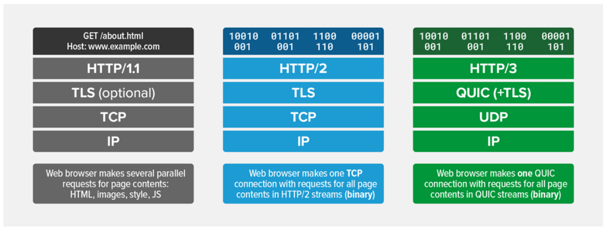

GET /about.html
Host: www.example.com
HTTP/1.1
TLS (optional)
TCP
IP
Web browser makes several parallel requests for page contents: HTML, images, style, JS
10010 001 01101 001 1100 110 00001 101
HTTP/2
TLS
TCP
IP
Web browser makes one TCP connection with requests for all page contents in HTTP/2 streams (binary)
10010 001 01101 001 1100 110 00001 101
HTTP/3
QUIC (+TLS)
UDP
IP
Web browser makes one QUIC connection with requests for all page contents in QUIC streams (binary)

Fig. 1: High-level comparison of HTTP/1.1, HTTP/2, and HTTP/3. The evolution shows the shift from TCP-based communication to QUIC over UDP, with improvements in multiplexing, latency, and connection handling. https://www.f5.com/fr_fr/glossary/quic-http3

As entirely new protocols for the Web, both QUIC and HTTP/3 have demonstrated unique performance advantages compared to previous generations, especially in terms of reduced latency, improved connection migration, and better resilience to packet loss [5]. Studies have shown that these protocols are increasingly valued in modern deployments for their efficiency and adaptability [6]. However, because HTTP/3 relies heavily on the relatively new QUIC transport protocol, security research in this area is still emerging and many aspects of their threat landscape remain unexplored [7] [6] [8] [9]. So, in this project, we'd like to develop a new attack and a corresponding mitigation defense. Our work thus bridges the gap between formal protocol definitions and their practical security implications, with a focus on the unique challenges posed by HTTP/3 as a modern web transport protocol.

II. Related Work

Security research on HTTP/3 is still emerging, especially considering its recent adoption and reliance on the QUIC transport protocol. Several studies have explored attack vectors, mitigation strategies, and specification details, yet many aspects of HTTP/3's threat surface remain insufficiently covered. [8] [9] [6] [7] [5] [1] [12] [3] [2] [4] [10] (see Table I for an overview of known attacks).

Chatzoglou et al. [6] conducted one of the first comprehensive security analyses of HTTP/3, using the H23Q dataset to compare HTTP/3 with HTTP/2. Their work revealed that several vulnerabilities were inherited from HTTP/2, such as resource exhaustion and stream management flaws. While valuable, their research largely focuses on known threats and does not introduce new vulnerabilities unique to QUIC or HTTP/3's design, such as header compression risks (QPACK) or connection migration abuse.

Balaji et al. [8] proposed QUICLORIS, a novel slow-rate Denial-of-Service (DoS) attack specifically designed for QUIC. Their findings demonstrate how attackers can exploit the protocol's stream multiplexing by holding server resources open with incomplete requests. While QUICLORIS sheds light on a critical DoS vulnerability, the paper does not address broader issues such as QUIC's interaction with TLS, replay attacks, or state exhaustion from connection migration.

Cao et al. [9] focused on the security risks posed by 0-RTT in TLS 1.3, especially in the context of QUIC. Their work highlights the vulnerability of early data to replay attacks and explores mitigation strategies such as anti-replay mechanisms and ticket reuse policies. However, their analysis remains mostly theoretical and lacks a detailed assessment of how 0-RTT risks manifest in real-world HTTP/3 implementations under high traffic or distributed architectures.

TABLE I: Overview of QUIC and HTTP/3 Attacks

Attack Name	Short Description	Attack Type	Reference Article
HTTP/3 Flood	A volumetric DoS attack exploiting HTTP/3 by sending excessive requests or data to exhaust resources.	Denial of Service (DoS)	A hands-on gaze on HTTP/3... [6]
HTTP/3 Loris	A slow-rate attack using partially formed HTTP/3 requests that consume server resources over time.	Denial of Service (DoS)	A hands-on gaze on HTTP/3... [6]
HTTP/3 Stream	Repeatedly sends partial HTTP requests/responses to keep connections open and drain server capacity.	Denial of Service (DoS)	A hands-on gaze on HTTP/3... [6]
QUIC Loris	A QUIC-based version of Slowloris that opens multiple incomplete QUIC connections.	Denial of Service (DoS)	QUICLORIS : A slow DOS ... [8]
0-RTT Replay	Abuses the 0-RTT feature of TLS 1.3 to replay early data across connections.	Replay Attack	0-RTT Attack and Defense... [9]
Downgrade Attack	Forcing a communication channel to revert to a weaker protocol or cipher suite.	Man-in-the-middle (MITM)	Exploring QUIC Security... [7] (Downgrade is no longer an attack that works on the version QUIC-2 [10])
TLS 1.3 RTT replay	The TLS 1.3 0-RTT replay attack exploits the lack of forward secrecy in early data, allowing attackers to resend requests.	Replay attack	Github [11] [12] (The attack works on TLS 1.3 but we couldn't get it to work on QUIC)
TLS Handshake Flood	Push a server into performing numerous full TLS handshakes by sending invalid tickets.	Denial of Service (DOS)	[10] [10] : This is an idea that was brought up during a meeting (Need to explore)

Standard Specifications

A number of foundational RFCs provide detailed definitions and motivations for the design of HTTP/3 and its underlying protocols. RFC 9114 [4] formalizes the HTTP/3 protocol, emphasizing improvements in multiplexing and stream independence compared to HTTP/2. RFC 9369 [10] introduces QUIC Version 2, aimed at preventing ossification and offering enhanced support for version negotiation.

To better understand the internal mechanisms of HTTP/3, we also studied RFC 9000 [3], which defines the core QUIC transport protocol, including its flow control, stream life-cycle, and network path migration features. Moreover, to assess the security guarantees at the encryption and handshake level, we referred to RFC 8446 [12], which specifies TLS 1.3. This RFC is essential for understanding how HTTP/3 leverages TLS over QUIC and how session tickets, key exchange, and early data are managed. These standards collectively helped us identify theoretical weaknesses and predict how certain attack vectors—like spoofed migration, session resumption misuse, and handshake manipulation—may occur.

Security and Privacy Survey on QUIC

A recent survey by Joarder and Fung [7] presents an extensive taxonomy of QUIC-related vulnerabilities, ranging from encryption layer attacks to privacy concerns like user fingerprinting. Their review is comprehensive in its scope, addressing both current threats and future research directions. However, the survey does not go into detail about HTTP/3-specific mechanisms, such as how QPACK compression or the removal of TCP semantics (e.g., head-of-line blocking resolution) might introduce new, unique risks.

Moreover, while the survey categorizes a broad array of attack vectors, it highlights that many of these threats currently lack robust, standardized mitigation techniques. For several classes of attacks—such as replay attacks, resource exhaustion, or side-channel leakage—effective countermeasures are still an open research problem, and deployment best practices are not yet fully established. Additionally, there is limited empirical evaluation of these vulnerabilities in real-world HTTP/3 deployments, and the interaction between new HTTP/3 features and legacy security controls remains underexplored.

Our Contributions

While previous works have explored QUIC and HTTP/3 from both theoretical and experimental perspectives, our work provides several novel contributions:

- We present a practical and reproducible 0-RTT spoofing attack against HTTP/3 servers, showing how an attacker can bypass IP-based access control mechanisms using QUIC early data and session ticket reuse.
- We developed an enhanced and modular attack client with clear separation between the legitimate client and the attacker roles, allowing detailed logging, flexible request crafting, and automated spoofing workflows.
- We performed a cross-analysis of RFC specifications (e.g., RFC 9000, 8446, 9114) and real-world server behavior, identifying security blind spots related to early data processing and IP validation timing.
- We proposed actionable security recommendations to mitigate this class of attacks and outlined future research directions, including blind probing and subnet-based spoofing strategies.

III. HTTP/3 Architecture: QUIC and TLS 1.3

To understand the attack surfaces introduced by HTTP/3, it is essential to examine its architectural foundation—namely, the QUIC transport protocol and its integration with TLS 1.3. These protocols are tightly coupled and define the security model and performance benefits of HTTP/3.

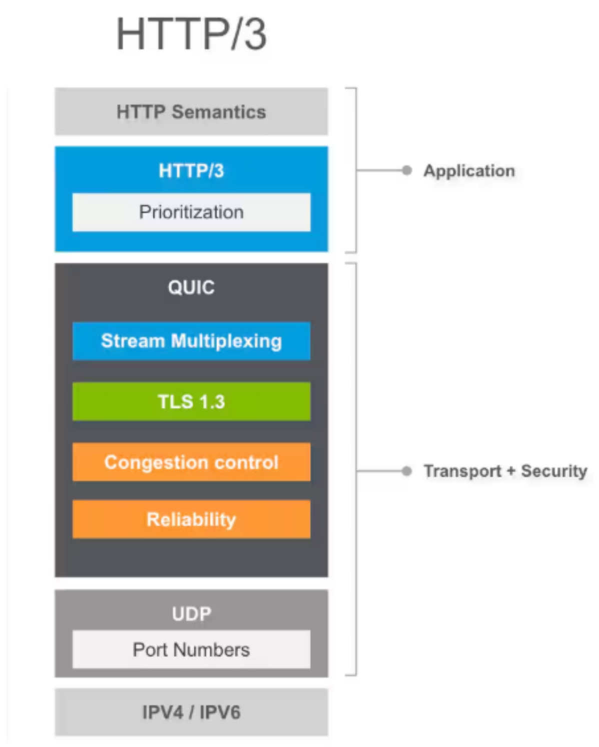


Fig. 2: Layered architecture of HTTP/3: the protocol stack illustrates the separation of concerns across application, transport, and network layers. HTTP semantics and prioritization operate over QUIC, which integrates TLS 1.3, stream multiplexing, and reliability on top of UDP and IP.
https://www.akamai.com/blog/performance/deliver-fast-reliable-secure-web-experiences-http3

Figure 2 provides a high-level overview of the HTTP/3 protocol stack. At the application layer, HTTP semantics and prioritization logic operate over QUIC, which handles transport-level responsibilities such as encryption, congestion control, and stream multiplexing. These functionalities are built atop UDP, which in turn relies on the IP layer (IPv4/IPv6) for routing and delivery. This layered design ensures modularity while enabling performance optimizations and enhanced security.

QUIC: A Modern Transport Protocol

QUIC (Quick UDP Internet Connections) is a UDP-based transport protocol designed to replace TCP for HTTP traffic, with integrated support for multiplexing and encryption. Defined in RFC 9000 [3], QUIC introduces several advanced capabilities:

- **Stream Multiplexing:** QUIC enables multiple bidirectional streams over a single connection without suffering from head-of-line blocking at the transport level.
- **Connection Migration:** Through the use of stable Connection IDs (CIDs), QUIC supports seamless client mobility, allowing sessions to persist across network path changes.
- **Built-in Encryption:** QUIC integrates TLS 1.3 within its handshake, ensuring early encryption and eliminating the need for separate layers.
- **Low-Latency Handshake:** With support for 0-RTT and 1-RTT handshakes, QUIC minimizes latency in connection establishment, a major improvement over traditional TCP + TLS.

In addition, RFC 9369 [10] introduces QUIC Version 2, which aims to mitigate ossification risks by modifying version negotiation and packet structure slightly, without altering the core protocol logic. While QUIC v2 does not fundamentally change the transport behavior, it demonstrates the protocol's adaptability and provides a framework for introducing secure updates in future versions.

TLS 1.3 Integration

TLS 1.3, as specified in RFC 8446 [12], is embedded within the QUIC handshake, providing confidentiality, authentication, and forward secrecy from the start of communication. Unlike earlier versions where TLS operated atop TCP, this integration enables better performance and stronger security guarantees:

- **Confidentiality and Integrity:** All QUIC payloads are encrypted and authenticated using keys derived during the TLS 1.3 handshake.
- **Forward Secrecy:** TLS 1.3 enforces perfect forward secrecy and removes insecure cryptographic features present in earlier versions.
- **0-RTT Data:** Clients can resume previous sessions and send early data immediately, reducing latency but potentially enabling replay attacks if not properly mitigated.

The tight coupling between QUIC and TLS 1.3 ensures that every QUIC connection is encrypted by default, with the cryptographic handshake driving transport-level key derivation.

HTTP/3 over QUIC

HTTP/3, defined in RFC 9114 [4], is a mapping of HTTP semantics onto QUIC. It eliminates many of

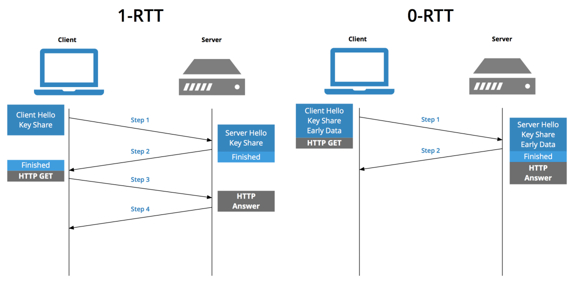


Fig. 3: Comparison of 1-RTT and 0-RTT handshake flows in QUIC. While 1-RTT requires a complete TLS exchange before data transmission, 0-RTT enables the client to send early data based on a previously obtained session ticket, reducing latency at the cost of replay attack exposure.

the inefficiencies of HTTP/2 over TCP, such as head-of-line blocking and connection reuse complexity. Key improvements include:

- **Stream Independence:** Each HTTP request/response exchange is mapped to its own QUIC stream, enabling parallelism without interference.
- **Header Compression via QPACK:** A modified version of HPACK suited for QUIC's independent stream model.
- **Fewer Round-Trips:** Combined with QUIC's handshake optimizations, HTTP/3 significantly reduces time to first byte.

The adoption of HTTP/3 across browsers and content delivery networks is growing, and its architecture is designed with modern web performance in mind. However, its reliance on complex interactions between QUIC and TLS introduces new, subtle vulnerabilities.

Security Implications

By combining transport, security, and application layers into a unified stack, HTTP/3 improves performance and simplifies connection handling. However, features like connection migration and 0-RTT—while beneficial for speed and resilience—create new opportunities for attacks such as session hijacking, address spoofing, and replay. Understanding the details of RFC 9000 [3], RFC 8446 [12], RFC 9114 [4], and RFC 9369 [10] is thus critical to assessing the security posture of HTTP/3 implementations.

IV. Attack Implementation: Client and Attacker Workflow for HTTP/3 0-RTT IP Spoofing

The QUIC protocol, underlying HTTP/3, introduces a feature known as 0-RTT (zero round-trip time) data, which allows clients to send requests during the handshake process. While this improves performance by reducing latency, it also opens new vectors for attacks if not carefully managed.

In this section, we present a practical implementation of an HTTP/3 0-RTT spoofing attack, where an attacker leverages a previously obtained session ticket to send an early request with a forged source IP address. This bypasses traditional IP-based access controls, allowing unauthorized access to protected server resources. We detail the technical components of our implementation, the attacker workflow, server-side protections, and the consequences of this vulnerability.

A. Code Base, Key Improvements, and Technical Features

Our work builds on the open-source project "http3-ip-spoofing" by Michael Wedl [13]. The original client is a proof-of-concept to demonstrate how an attacker can exploit HTTP/3 0-RTT early data to perform IP spoofing. This base allows an attacker to obtain a session ticket, create a 0-RTT HTTP/3 request, and send it with a spoofed source IP address using temporary firewall rules (iptables). (Figure 4)

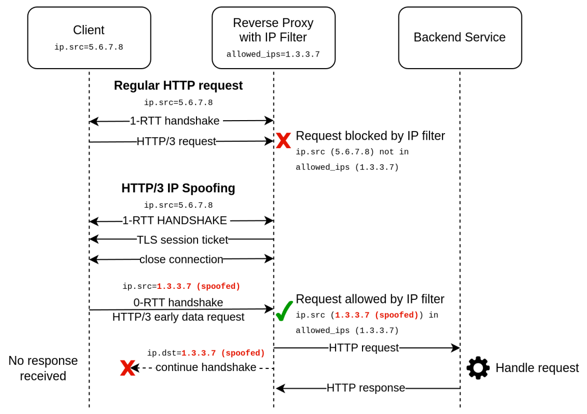


Fig. 4: HTTP/3 IP Spoofing via Early Data Requests in QUIC 0-RTT Handshake.

For our research and experiments, we made several improvements and customizations to this client: [14]

- **Separation of Roles:** We divided the workflow into two distinct modes: a "client" mode, which establishes a legitimate connection and collects session tickets, and an "attacker" mode, which reuses those tickets to send spoofed requests. This separation was introduced after encountering an initial limitation: we were able to extract the cryptographic keys only for the first (legitimate) connection, but not for the second 0-RTT spoofed connection. As a result, we could not decrypt or fully analyze the spoofed packets in Wireshark. By splitting the client into two independent components, we ensured that the keys for each connection could be captured and logged separately, enabling complete decryption and facilitating easier monitoring, reproduction, and controlled experimentation.

- **Advanced Session Ticket Handling:** Session tickets are not only collected but also serialized and stored, so the attacker can load and reuse them in a separate phase. This helps to analyze the attack step-by-step.
- **Flexible HTTP Request Crafting:** The client supports various HTTP methods (GET, POST, etc.), custom headers, and data payloads, allowing us to test attacks on multiple types of endpoints, including those requiring authentication or specific content.
- **Comprehensive Logging and Diagnostics:** Both the client and the server record all QUIC handshake secrets and protocol events in log files. This provides a complete trace for each attack attempt, useful for analysis in tools like Wireshark and for confirming whether the attack succeeded.
- **Automatic Cleanup of Spoofing Rules:** The implementation ensures that every temporary iptables rule for IP spoofing is properly removed after each attempt. This keeps the test environment reliable and prevents conflicts or side effects.
- **User-Friendly Command-Line Interface:** The command-line interface is designed to let users specify all relevant parameters (role, target URL, spoofed IP, HTTP method, headers, and request body), making the attack workflow flexible and easy to reproduce for new experiments or demonstrations.

These enhancements provide a modular, extensible, and transparent attack framework suitable for systematic testing of HTTP/3 0-RTT IP spoofing scenarios.

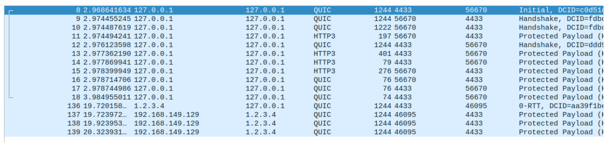
B. Workflow: From Legitimate Connection to Attack

The attack process consists of several clearly defined steps:

1. Collecting a Session Ticket with a Legitimate Client. The process begins with the client establishing a standard HTTP/3 connection to the target server using aioquic.(Figure 5) As part of the QUIC handshake, the server issues a session ticket, which is a prerequisite for 0-RTT session resumption. This ticket is captured asynchronously and stored for later use by the attacker. All relevant cryptographic secrets from this connection are also logged, allowing in-depth post-experiment analysis.

2. Crafting and Sending a Spoofed 0-RTT Request. In the next phase, the attacker role loads the previously saved session ticket and prepares a new HTTP/3 connection. The attacker crafts a full HTTP/3 request (with desired method, headers, and body) as 0-RTT early data. The aioquic client builds a raw UDP datagram containing both the QUIC handshake and the request. Before sending this datagram, a temporary iptables rule is added so that the outgoing UDP packet will appear to come from an arbitrary IP address—the attacker can spoof any address, including loopback addresses like 127.0.0.1.(Figure 5) Once the datagram is sent, the iptables rule is removed to keep the environment clean.

3. Manual Packet Injection and Timing. By directly transmitting the raw UDP datagram (instead of using a full client handshake), the attacker ensures that the forged request is delivered to the server before the application layer has performed any IP validation. This approach makes it possible to bypass even advanced IP-based restrictions that would normally block unauthorized access.



```
  8 2.968641634 127.0.0.1        127.0.0.1  QUIC  1244 4433   56670  Initial, DCID=c0d51c
  9 2.974455245 127.0.0.1        127.0.0.1  QUIC  1244 56670  4433   Handshake, DCID=fdbc
 10 2.974487619 127.0.0.1        127.0.0.1  QUIC  1222 56670  4433   Handshake, DCID=fdbc
 11 2.974494241 127.0.0.1        127.0.0.1  HTTP3  197 56670  4433   Protected Payload (K
 12 2.976123598 127.0.0.1        127.0.0.1  QUIC  1244 4433   56670  Handshake, DCID=ddd9
 13 2.977362190 127.0.0.1        127.0.0.1  HTTP3  401 4433   56670  Protected Payload (K
 14 2.977869941 127.0.0.1        127.0.0.1  HTTP3   79 4433   56670  Protected Payload (K
 15 2.978399949 127.0.0.1        127.0.0.1  HTTP3  276 56670  4433   Protected Payload (K
 16 2.978714706 127.0.0.1        127.0.0.1  QUIC    76 56670  4433   Protected Payload (K
 17 2.978744986 127.0.0.1        127.0.0.1  QUIC    76 4433   56670  Protected Payload (K
 18 3.984955011 127.0.0.1        127.0.0.1  QUIC    74 4433   56670  Protected Payload (K
136 19.720158…  1.2.3.4          127.0.0.1  QUIC  1244 4433   46095  0-RTT, DCID=aa39f1be
137 19.723972…  192.168.149.129  1.2.3.4    QUIC  1244 46095  4433   Protected Payload (K
138 19.923953…  192.168.149.129  1.2.3.4    QUIC  1244 46095  4433   Protected Payload (K
139 20.323931…  192.168.149.129  1.2.3.4    QUIC  1244 46095  4433   Protected Payload (K
```

Fig. 5: Wireshark capture showing: (1) the initial legitimate HTTP/3 client handshake, including session ticket exchange, and (2) the subsequent attacker connection using 0-RTT early data with a spoofed IP address.

C. Server-Side Protections: IP-Based Filtering

To test the attack under realistic conditions, we updated our aioquic-based server to restrict access to sensitive resources using IP allowlists. The server maintains a list (ALLOWED-IP) of authorized addresses (Figure 6) [14].

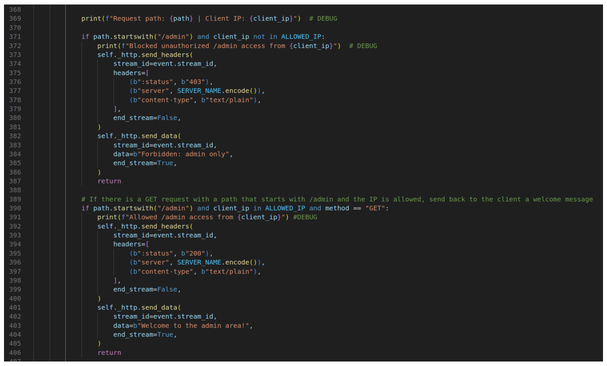
- Any HTTP/3 request for a path beginning with `/admin` is only allowed if the client's IP address is in the allowlist. Otherwise, the server logs the attempt and replies with a 403 Forbidden status and an explanatory message.
- For even stricter protection, endpoints such `/api/allowlist/add` only accept requests from loopback addresses (127.0.0.1/24). These endpoints are often used to update security-critical settings, such as changing which clients can access admin APIs.
- If a client's IP is allowed and the request is a GET, the server replies with a welcome message, confirming successful entry to the admin area.

This setup mirrors real-world web server defenses, where administrative or sensitive APIs are protected by IP filtering or firewall rules.

However, during the initial implementation, we encountered a misconfiguration in the placement of the

IP filtering logic. The filtering was originally applied at the QUIC transport layer, before the HTTP/3 request parsing stage. This approach proved ineffective for our purposes, as the QUIC layer operates without visibility into HTTP-level details such as request paths. Consequently, the server could not reliably enforce path-specific restrictions (e.g., allowing `/admin` only for certain IPs) because these checks require information available only at the HTTP layer.

To resolve this issue, we moved the IP validation logic from the QUIC transport level to the HTTP/3 application layer, where complete request context including the target path and method is available. This change ensured that IP-based restrictions could be accurately applied to specific endpoints, aligning our setup more closely with real-world deployment practices and enabling a precise evaluation of the attack's effectiveness.



```python
print(f"Request path: {path} | Client IP: {client_ip}")  # DEBUG

if path.startswith("/admin") and client_ip not in ALLOWED_IP:
    print(f"Blocked unauthorized /admin access from {client_ip}")  # DEBUG
    self._http.send_headers(
        stream_id=event.stream_id,
        headers=[
            (b":status", b"403"),
            (b"server", SERVER_NAME.encode()),
            (b"content-type", b"text/plain"),
        ],
        end_stream=False,
    )
    self._http.send_data(
        stream_id=event.stream_id,
        data=b"Forbidden: admin only",
        end_stream=True,
    )
    return

# If there is a GET request with a path that starts with /admin and the IP is allowed, send back to the client a welcome message
if path.startswith("/admin") and client_ip in ALLOWED_IP and method == "GET":
    print(f"Allowed /admin access from {client_ip}") #DEBUG
    self._http.send_headers(
        stream_id=event.stream_id,
        headers=[
            (b":status", b"200"),
            (b"server", SERVER_NAME.encode()),
            (b"content-type", b"text/plain"),
        ],
        end_stream=False,
    )
    self._http.send_data(
        stream_id=event.stream_id,
        data=b"Welcome to the admin area!",
        end_stream=True,
    )
    return
```

Fig. 6: Implementation of IP-based blocking logic in the aioquic HTTP/3 server. Requests to protected endpoints are only allowed if the client's source IP address is present in the allowlist; otherwise, the server returns HTTP 403 Forbidden.

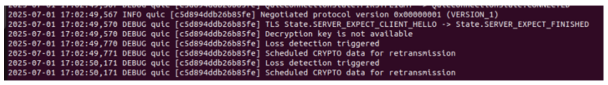
D. Session Ticket Validity and the Need to Restart the Client

During our experiments, we observed that the QUIC session ticket obtained by the client can only be used once for a 0-RTT resumption. After the attacker uses the session ticket to initiate a spoofed 0-RTT connection, that ticket is invalidated or considered "consumed" by the server according to the QUIC and TLS-1.3 specifications.(Figure 7)

More specifically, as defined in RFC-9000 [3] and RFC-8446 [12], QUIC session tickets (also called PSK tickets in TLS) are intended to provide forward secrecy and mitigate replay attacks. To do so, servers typically enforce a **single-use** policy for session tickets that allow 0-RTT data. Once a ticket has been used for early data, it cannot be used again for another resumption. This mechanism is crucial to prevent attackers from replaying the same 0-RTT payload multiple times.

As a consequence, every time we wish to replay the attack, we must relaunch the client in "client" mode to establish a fresh connection and obtain a new session ticket from the server. Only then can the attacker role use the new ticket for another spoofed 0-RTT connection attempt.

This behavior reflects proper QUIC server security, as it ensures that each 0-RTT attempt is uniquely tied to a single session ticket, and that the replay window for early data is tightly limited.



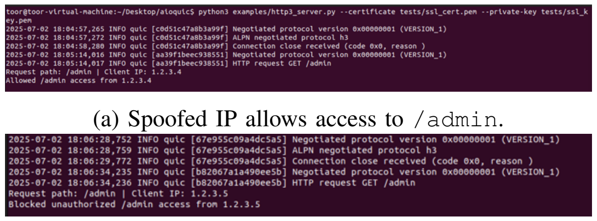
```
2025-07-01 17:02:49,567 INFO quic [c5d894ddb26b85fe] Negotiated protocol version 0x00000001 (VERSION_1)
2025-07-01 17:02:49,570 DEBUG quic [c5d894ddb26b85fe] TLS State.SERVER_EXPECT_CLIENT_HELLO -> State.SERVER_EXPECT_FINISHED
2025-07-01 17:02:49,570 DEBUG quic [c5d894ddb26b85fe] Decryption key is not available
2025-07-01 17:02:49,770 DEBUG quic [c5d894ddb26b85fe] Loss detection triggered
2025-07-01 17:02:49,771 DEBUG quic [c5d894ddb26b85fe] Scheduled CRYPTO data for retransmission
2025-07-01 17:02:50,171 DEBUG quic [c5d894ddb26b85fe] Loss detection triggered
2025-07-01 17:02:50,171 DEBUG quic [c5d894ddb26b85fe] Scheduled CRYPTO data for retransmission
```

Fig. 7: Terminal output showing the client acquiring a new session ticket before each spoofed 0-RTT attack attempt. After each use, the previous ticket becomes invalid and must be replaced.

E. Bypassing Defenses with 0-RTT IP Spoofing

Despite these protections, our attack demonstrates that 0-RTT IP spoofing can bypass even carefully configured allowlists. By sending a 0-RTT request with a spoofed source IP (Figure 8) (such as a loopback address), the attacker can:

- Access administrative endpoints restricted to internal clients.
- Update the server's allowlist by forging requests to endpoints like `/api/allowlist/add`, which should only accept local traffic.



```
toor@toor-virtual-machine:~/Desktop/aioquic$ python3 examples/http3_server.py --certificate tests/ssl_cert.pem --private-key tests/ssl_k
ey.pem
2025-07-02 18:04:57,265 INFO quic [c0d51c47a8b3a99f] Negotiated protocol version 0x00000001 (VERSION_1)
2025-07-02 18:04:57,272 INFO quic [c0d51c47a8b3a99f] ALPN negotiated protocol h3
2025-07-02 18:04:58,280 INFO quic [c0d51c47a8b3a99f] Connection close received (code 0x0, reason )
2025-07-02 18:05:14,016 INFO quic [aa39f1beec938551] Negotiated protocol version 0x00000001 (VERSION_1)
2025-07-02 18:05:14,017 INFO quic [aa39f1beec938551] HTTP request GET /admin
Request path: /admin | Client IP: 1.2.3.4
Allowed /admin access from 1.2.3.4
```

(a) Spoofed IP allows access to `/admin`.

```
2025-07-02 18:06:28,752 INFO quic [67e955c09a4dc5a5] Negotiated protocol version 0x00000001 (VERSION_1)
2025-07-02 18:06:28,759 INFO quic [67e955c09a4dc5a5] ALPN negotiated protocol h3
2025-07-02 18:06:29,772 INFO quic [67e955c09a4dc5a5] Connection close received (code 0x0, reason )
2025-07-02 18:06:34,235 INFO quic [b82067a1a490ee5b] Negotiated protocol version 0x00000001 (VERSION_1)
2025-07-02 18:06:34,236 INFO quic [b82067a1a490ee5b] HTTP request GET /admin
Request path: /admin | Client IP: 1.2.3.5
Blocked unauthorized /admin access from 1.2.3.5
```

(b) Non-allowed IP is denied access to `/admin`.

Fig. 8: Terminal output showing: (a) successful access to the `/admin` endpoint using a spoofed loopback IP, and (b) access denied when using a non-allowed IP.

The key reason this attack works is that QUIC, using UDP, allows early data to be sent and processed before the handshake is fully complete and before the server can reliably validate the true source IP at the application layer. The forged request is accepted and executed as if it was coming from a trusted client.

F. Verification and Evidence of Attack Success

To confirm the effectiveness of the attack, we analyze both server logs and protocol traces. When the spoofed packet is accepted, the server log shows the action as coming from the forged IP address. Access to endpoints and actions reserved for loopback or authorized IPs is granted, even though the real client was external. Although the attacker cannot receive the actual server response (since it is sent to the spoofed address), the full set of logs and traces from both the client and the server confirm the attack's impact and demonstrate the vulnerability.

G. *Limitations, Implications, and Potential for Abuse*

Our results highlight a fundamental limitation of IP-based allowlists for HTTP/3 servers that support 0-RTT. Because early data is handled before full handshake validation, the server cannot distinguish between a legitimate local client and an attacker using IP spoofing with a previously acquired session ticket. This attack vector is not only relevant for admin endpoints, but could also be used to:

- Bypass rate-limiting or per-IP quotas by flooding requests from spoofed addresses.
- Poison application logs, making tracking and forensic investigation more difficult.
- Exploit internal-only endpoints or APIs that trust the client IP, potentially leading to further privilege escalation or lateral movement inside a network.
- Combine with other protocol-level attacks, such as replay attacks, to further extend the impact.

As HTTP/3 becomes more widely deployed and 0-RTT is adopted for performance, the risk posed by this type of attack will likely increase. Servers must not rely solely on client IP validation for access control, especially for critical or sensitive endpoints.

H. *Recommendations and Practical Security Guidance*

To defend against this vulnerability, we recommend several concrete measures:

- **Disable 0-RTT for Critical Endpoints:** For endpoints that change security settings, allow admin access, or perform sensitive actions, do not permit 0-RTT resumption.
- **Implement Anti-Replay Mechanisms:** Ensure that early data cannot be replayed or re-used by an attacker. Use strict anti-replay windows and session tracking.
- **Avoid Trusting IP Before Handshake Completion:** Only check and enforce IP-based access control after the handshake is validated, not based on early packets or UDP source IP.
- **Use Stronger Authentication:** Where possible, add mutual authentication or token-based protection to sensitive endpoints.

I. *Future Work and Research Directions*

Our proof-of-concept shows that an attacker can use HTTP/3 0-RTT to bypass IP-based protections. However, there are still many areas to explore. In future work, we could develop detection systems that look for unusual patterns in 0-RTT requests—such as a mismatch between the claimed IP address and the real one, or strange timing behavior. Another idea is to test different defense methods, like adding small puzzles for the client to solve, limiting how many times a session ticket can be used, or checking session tokens more strictly.

It would also be interesting to apply this attack model in blind conditions, where the attacker receives no response but infers success from side effects, such as logs or secondary behaviors. A related approach would be to test large lists of API endpoints and admin paths—hoping to find hidden or misconfigured services that rely on IP filtering. In addition, the attack could be extended to scan an entire subnet of IP addresses to discover which ones are authorized or whitelisted by the server, helping to map internal network structures.

Finally, we could improve our attack tool to simulate multiple fake IPs or to chain attacks inside the network, helping researchers and developers build stronger protections.

J. *Conclusion*

In summary, our experiments demonstrate that HTTP/3's 0-RTT feature, while valuable for performance, can create new opportunities for IP spoofing and access control bypass if not carefully managed. Our enhanced attack client shows that even robust server-side protections can be defeated in this scenario. As next-generation web protocols evolve, careful implementation and security awareness are critical to avoid re-introducing well-known vulnerabilities in new forms.

References

[1] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, Jun. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2616

[2] M. Thomson and C. Benfield, "HTTP/2," RFC 9113, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9113

[3] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[4] M. Bishop, "HTTP/3," RFC 9114, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9114

[5] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the conference of the ACM special interest group on data communication*, 2017, pp. 183–196.

[6] E. Chatzoglou, V. Kouliaridis, G. Kambourakis, G. Karopoulos, and S. Gritzalis, "A hands-on gaze on http/3 security through the lens of http/2 and a public dataset," *Computers & Security*, vol. 125, p. 103051, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822004436

[7] Y. Joarder and C. Fung, "Exploring quic security and privacy: A comprehensive survey on quic security and privacy vulnerabilities, threats, attacks and future research directions," *IEEE Transactions on Network and Service Management*, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10677379

[8] A. S. Balaji, V. Anil Kumar, P. Amritha, and M. Sethumadhavan, "Quicloris: a slow denial-of-service attack on the quic protocol," in *International Conference on Signal Processing and Integrated Networks*. Springer, 2022, pp. 85–94. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-99-1312-1_7

[9] X. Cao, S. Zhao, and Y. Zhang, "0-rtt attack and defense of quic protocol," in *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9024637

[10] M. Duke, "QUIC Version 2," RFC 9369, May 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9369

[11] CiscoCXSecurityLabs, "tlsplayback," https://github.com/CiscoCXSecurity/tlsplayback.git, 2018, accessed: 2025-01.

[12] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[13] M. Wedl, "http3-ip-spoofing," https://github.com/MWedl/http3-ip-spoofing/tree/main?tab=readme-ov-file, 2024, accessed: 2025-01.

[14] Y. Chicheportiche and T. Malka, "Http/3 attack - 0rtt spoof attack," https://github.com/YannChich/HTTP-3-Attacks.git, 2024-2025.