Penetration testing of HTTP/3 Servers for the detection of new vulnerabilities

Shaan Kumar, Naresh Kumar, Nikita
Department of Computer Science and Engineering
Indian Institute of Technology (BHU), Varanasi
Varanasi, Uttar Pradesh, India
{shaan.kumar.cse19, naresh.kumar.cse19, nikita.student.cse19}@itbhu.ac.in

Dr. Mayank Swarnkar
Department of Computer Science and Engineering
Indian Institute of Technology (BHU)
Varanasi, Uttar Pradesh, India
mayank.cse@iitbhu.ac.in

***Abstract*— HTTP/3 is the new-age protocol, which is based on QUIC, and increasingly being used by big internet companies like Google and Facebook etc. QUIC, a general transport protocol, is based on UDP (User Datagram Protocol), and implements almost all features provided by TCP. It is reliable and efficient.**

In this research paper, we have tried to find vulnerabilities in servers that support HTTP/3 like nginx, aioquic, Couldfare, Openlitespeed. We also compared their performance based on Packet RX Time and Handshake Time.

***Index Terms*— HTTP/3, QUIC, UDP, Vulnerabilities**

I. INTRODUCTION

The Hypertext Transfer Protocol (HTTP) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser. [1]

HTTP/3, the latest major version of the Hypertext Transfer Protocol, the successor to HTTP/2, was published in 2022. It is now used by over 25% of websites and is supported by many web browsers (over 75% of users). HTTP/3 uses QUIC instead of TCP for the underlying transport protocol. Like HTTP/2, it does not obsolesce previous major versions of the protocol. Support for HTTP/3 was added to Cloudflare and Google Chrome first, and is also enabled in Firefox. HTTP/3 has lower latency for real-world web pages, if enabled on the server, load faster than with HTTP/2, and even faster than HTTP/1.1, in some cases over 3× faster than HTTP/1.1 (which is still commonly only enabled). [1]

QUIC is a new multiplexed transport built on top of UDP. HTTP/3 is designed to take advantage of QUIC's features, including lack of Head-Of-Line blocking between streams.

The QUIC project started as an alternative to TCP+TLS+HTTP/2, with the goal of improving user experience, particularly page load times. The QUIC working group at the IETF defined a clear boundary between the transport(QUIC) and application(HTTP/3) layers, as well as migrating from QUIC Crypto to TLS 1.3.

Because TCP is implemented in operating system kernels and middleboxes, widely deploying significant changes to TCP is next to impossible. However, since QUIC is built on top of UDP and the transport functionality is encrypted, it suffers from no such limitations. [2]

Key features of QUIC and HTTP/3 over TCP+TLS and HTTP/2 are:

- Reduced connection establishment time - 0 round trips in the common case
- Improved congestion control feedback
- Multiplexing without head of line blocking
- Connection migration
- Transport extensibility
- Optional unreliable delivery

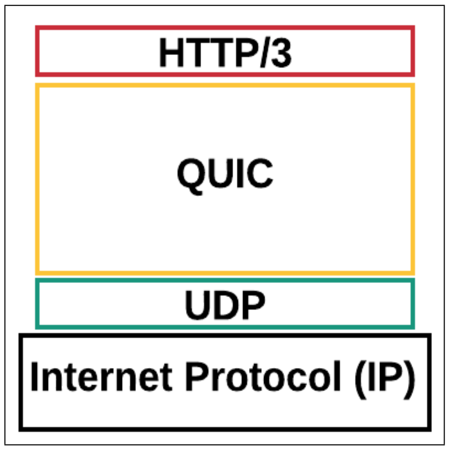

Fig. 1. HTTP/3 Underlying Structure

HTTP/3 is the latest version of HTTP protocol that is used. This aims to increase the security and efficiency of communication in client-server system. It is not a major reform from HTTP/2 but a better implemented protocol.

This paper is structured as follows: Section II provides details on the background and related work. Section III provides details on the problem statement that this paper is trying to solve. The section's following explains the motivation and experiments performed to analyse the performance of HTTP/3 supporting test servers.

II. BACKGROUND AND RELATED WORK

HTTP/3.0 is based on QUIC, and provides improved performance in comparison to HTTP/2.0 which implemented multiplexing, which means that multiple requests and responses can be sent and received asynchronously. Now, as each frame has a stream ID linked with it, which eliminates the need for client and server to process HTTP requests and responses in sequential order. Thus, resolving HOL blocking at HTTP layer. But, if a packet is lost, then all the subsequent packets need to be held in receiver's TCP buffer, till the lost packet is retransmitted and reaches the receiver. As this happens at the TCP layer, HTTP layer can't know the reason for slow transmission, but just wait before the whole sequence of data is transmitted to the receiver. This is known as HOL blocking at TCP layer.

So, as to solve this, Google introduced a new protocol called as QUIC which is implemented on UDP and encrypted with TLS, multiplexed, and low-latency transport protocol over UDP. In comparison to HTTP/2, QUIC has an in-built security using TLS, and requires atleast TLS v1.3, whereas, it's optional with HTTP/2, and it can still use TLS v1.2. So, a lot of research done for HTTP/2.0 is not meaningful when doing research on HTTP/3.0

IETF version of QUIC is significantly different from gQUIC, which was proposed by Google. A basic difference is that gQUIC transports HTTP/2 frames, whereas IETF's QUIC is more of a general-purpose transporting HTTP/3.0 and DNS. So, a lot of the research work that was done on gQUIC may not be applicable on IETF's QUIC. [3]

As it is a new protocol, there is not a lot of research done that provides a good idea about security of servers implementing

Some of the possible attacks that can be performed on QUIC are 0-RTT, ACK, request forgery and DoS attacks. QUIC is vulnerable to 0-RTT (Zero Round Trip) attacks, where the attacker sends congestion information, using a legitimately acquired address validation token, to a victim that has the IP address used to acquire the token. It is also vulnerable to possible ACK attacks, in which case the opponent sends acknowledgment packets for the QUIC packet it never received. It can then cause the congestion controller to give permission to the end points to send packets at a rate higher than original network throughput. It can also experience a request forgery attack, in which case the attacker has access to the requests that a peer sends to a victim, and then may exploit the authorizations of a potential peer. Also, QUIC can suffer from DoS attacks, in which an attacker sends an enough number of packets to keep hold of the open connection with the attacked system, to deplete its resources. [4]

In the presence of attackers, QUIC may not be able to attain 0-RTT connections. The adversary can make it fall-back to TCP/TLS or cause the client and server to have an inconsistent view of their handshake, which could lead to inconsistent states and more latency. Furthermore, such simple attacks could also be used to mount DoS attacks. [5]

There is very less research being done till now on IETF QUIC, and because of the complexity in design of QUIC in comparison to TCP+TLS, many of the researches have not provided any concrete results. There have been papers where penetration testing was performed, and behaviour of different servers was analyzed and compared based on parameters like flow control, multiplexing and prioritization, packetization and congestion control. [6]

In our research, we have analysed performance of different servers based on Packet RX Time and Handshake Time. Packet RX Time - the time between the first packet sent, and the first packet received, whereas Handshake Time - the time between when the first packet is sent and when the handshake is completed.

III. PROBLEM STATEMENT

Penetration testing of HTTP/3 servers for detection of new vulnerabilities. This paper tries to find vulnerabilities in publicly available HTTP/3 test servers like nginx, Cloudflare, aioquic, Openlitespeed.

IV. MOTIVATION

HTTP/3 is a very recent HTTP protocol. As this protocol is vulnerable, hackers can exploit the vulnerabilities. The adoption of HTTP/3 is increasing day by day. That's why analysing the risk and benefit which comes with HTTP/3 is a very important task that needs to be done. In the past, few types of research have been done that find vulnerability in HTTP/3. This was the main motivation for our work in the penetration testing of HTTP/3.

V. PROPOSED WORK

We will try to do penetration testing of HTTP/3 supporting test servers like nginx, Cloudflare, aioquic, Openlitespeed. We will analyze the performance of these servers on the basis of **Handshake Time** & **Packet RX Time**. We will compare these servers to decide which server is best to use in general.

VI. EXPERIMENTAL SETUP

The experimental setup consisted of following components(Fig. 2):

- Scapy was used to modify the QUIC packets.
- Python was used to write script for the flooding.
- Wireshark was used to analyze the packet information.

- For test servers we used Nginx, aioquic, Cloudflare, Openlitespeed.
- For measuring HANDSHAKE time and PACKET RX time, we used `http3check.net`.

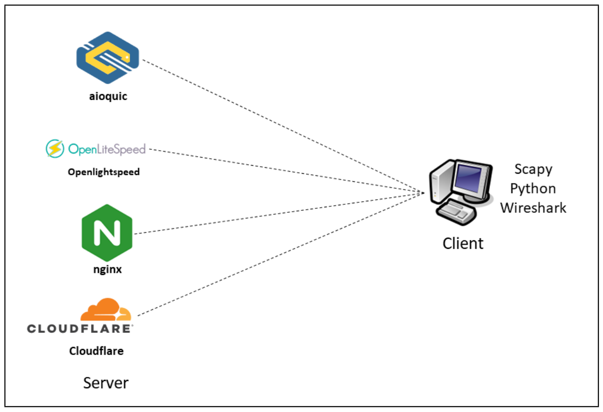

Fig. 2. Experimental Setup

VII. Work Done and Results Analysis

Multiple QUIC packets are exchanged between the client and server through a single UDP Datagrams. QUIC packet consists of a QUIC header and frames. QUIC has two different types of headers - long and short. Long header is used before the connection is set up and short header after that.

Long header includes the following fields such as - Header Form, Fixed Bit, Version ID, Destination Connection ID and Source Connection ID.

Short header consists of the following fields - Header Form, Fixed Bit, Packet Number Length, Destination Connection ID, Packet Number and Packet Payload.

In this research, we observe the Packet RX Time and Handshake Time when parameters, mentioned below, are modified in the QUIC header and then the server is flooded with these packets:

1) Set Version number to zero
2) Set Version number to a positive value
3) Changing fixed bit in the public flag
4) Changing packet number length in the public flag
5) Buffer Overflow

The readings of Handshake Time & Packet RX Time observed are shown in **Table I** & **Table II** respectively.

The % change in Packet RX Time & Handshake Time observed from normal behaviour for all the experiments is shown graphically in Fig. 3 for **Experiment 1**, Fig. 4 for **Experiment 2**, Fig. 5 for **Experiment 3**, Fig. 6 for **Experiment 4**, Fig. 7 for **Experiment 5**.

VIII. Conclusions

After performing all the experiments, we noted the Handshake Time & Packet RX Time for each experiment. Now, we tried to analyze the servers on the basis of average percentage change in Packet RX Time & Handshake Time, to decide which one is better compared to others

Server name	Normal(Avg)	Exp1	Exp2	Exp3	Exp4	Exp5
aioquic	181.94	187.77	273.57	431.86	190.81	472.26
Isquic-Openlightspeed	20.42	22.43	23.87	31.67	25.56	23.53
Quic-nginx	175.22	157.99	162.01	167.91	157.75	164.06
Cloudflare-Quiche	157.40	156.11	165.95	767.24	157.09	758.94

TABLE I

HANDSHAKE TIME FOR SERVERS ON DIFFERENT TESTS

Server name	Normal(Avg)	Exp1	Exp2	Exp3	Exp4	Exp5
aioquic	91.67	94.64	94.85	91.17	98.120	88.68
Isquic-Openlightspeed	9.64	11.24	22.07	9.34	15.21	12.18
Quic-nginx	85.74	76.11	78.87	79.96	75.98	78.96
Cloudflare-Quiche	155.66	153.62	164.19	755.77	155.35	757.20

TABLE II

PACKET RX TIME FOR SERVERS ON DIFFERENT TESTS

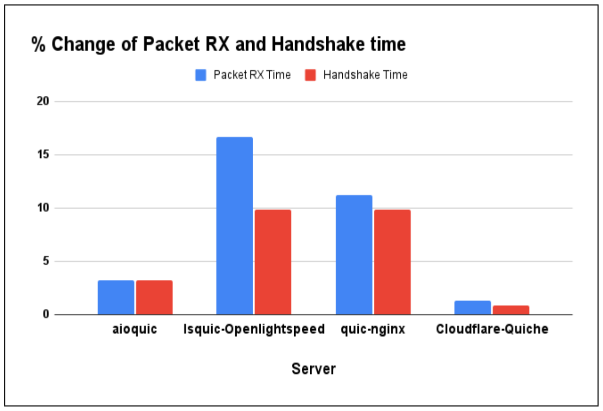

Fig. 3. Experiment 1

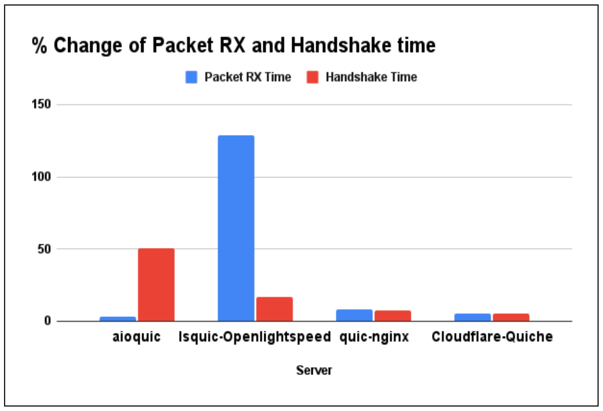

Fig. 4. Experiment 2

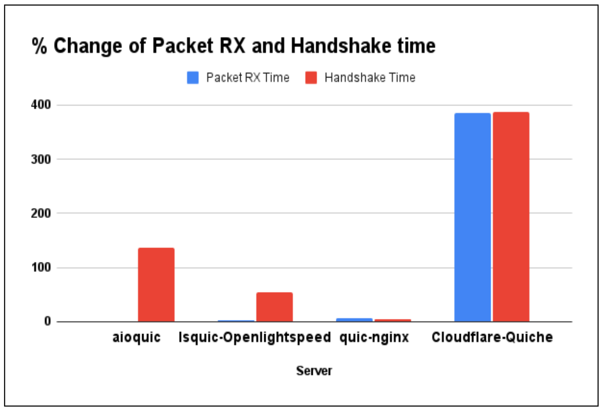

Fig. 5. Experiment 3

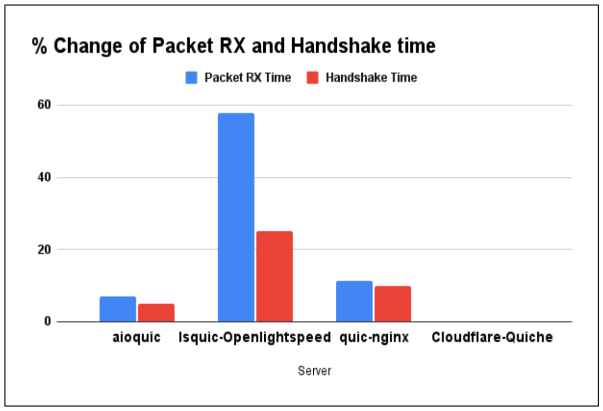

Fig. 6. Experiment 4

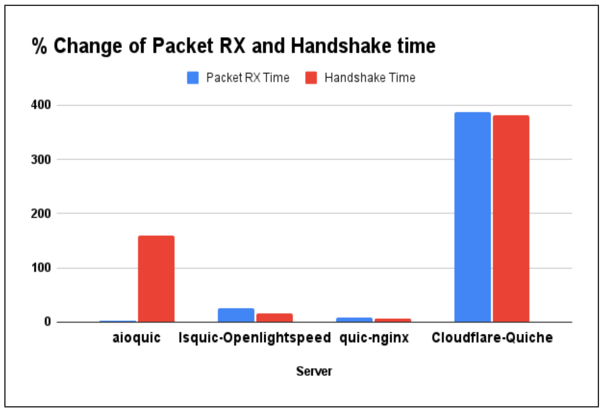

Fig. 7. Experiment 5

in general. Average percentage change in Packet RX Time & Handshake Time is shown in Table III.

The average percentage graph for Handshake Time & Packet RX Time is shown in (Fig. 8) & (Fig. 9) respectively.

Our experiment of penetration testing shows that the order for using these servers is: On the basis of

- Handshake Time: **nginx** > **openlightspeed** > **aioquic** > **cloudflare**
- Packet RX Time: **aioquic** > **nginx** > **openlightspeed** > **cloudflare**

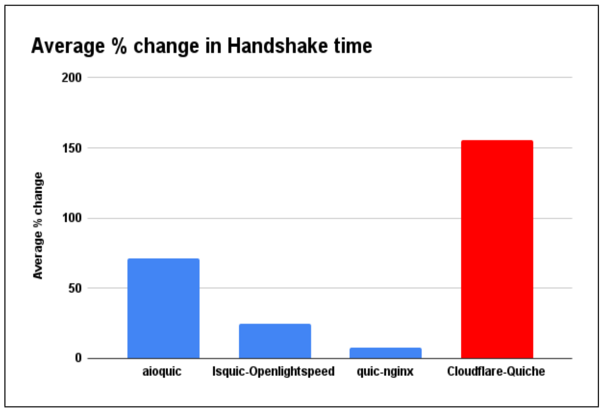

Fig. 8. Average Handshake Time Graph

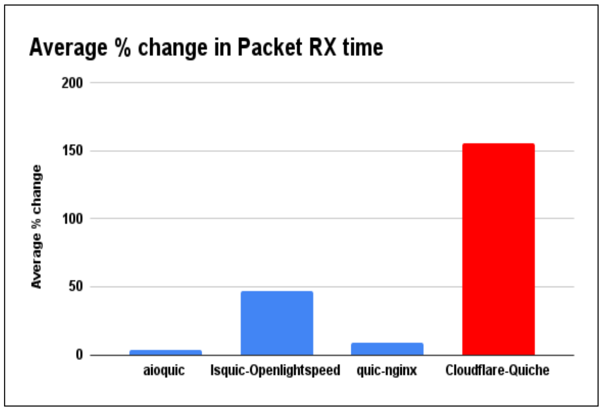

Fig. 9. Average Packet RX Time Graph

Server	Average % change	
	Packet RX Time	Handshake Time
aioquic	3.52	71.07
lsquic-Openlightspeed	46.54	24.48
quic-nginx	9.05	7.58
Cloudflare-Quiche	155.80	155.21

TABLE III
AVERAGE PERCENTAGE CHANGE IN READINGS FROM NORMAL BEHAVIOUR

ACKNOWLEDGMENT

We would like to thank Dr. Mayank Swarnkar for his valuable comments and suggestions to improve the quality of the project. We are also grateful to him for helping us review our work regularly. We would also like to thank the Department of Computer Science, IIT BHU for providing us with this opportunity.

REFERENCES

[1] En.wikipedia.org. (2022). Hypertext Transfer Protocol. [online]. Available: `https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#cite_note-rfc9110-1`. [Accessed 3 Dec. 2022].
[2] Chromium.org. (2022). QUIC, a multiplexed transport over UDP. [online]. Available: `https://www.chromium.org/quic/`. [Accessed 3 Dec. 2022].
[3] Alexander Yu and Theophilus A. Benson. 2021. Dissecting Performance of Production QUIC. In Proceedings of the Web Conference 2021 (WWW '21). Association for Computing Machinery, New York, NY, USA, 1157–1168. https://doi.org/10.1145/3442381.3450103
[4] Efstratios Chatzoglou, Vasileios Kouliaridis, Georgios Karopoulos et al. Revisiting QUIC attacks: A comprehensive review on QUIC security and a hands-on study, 01 July 2022, PREPRINT (Version 1) available at Research Square [https://doi.org/10.21203/rs.3.rs-1676730/v1]
[5] R. Lychev, S. Jero, A. Boldyreva and C. Nita-Rotaru, "How Secure and Quick is QUIC? Provable Security and Performance Analyses," 2015 IEEE Symposium on Security and Privacy, 2015, pp. 214-231, doi: 10.1109/SP.2015.21.
[6] Robin Marx, Joris Herbots, Wim Lamotte, and Peter Quax. 2020. Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity. In Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ '20). Association for Computing Machinery, New York, NY, USA, 14–20. https://doi.org/10.1145/3405796.3405828