USENIX Security '26 Artifact Appendix: Breaking the Boundaries: Analyzing QUIC Frame-Packet Interactions With QUIC-Attacker

Nurullah Erinola, Marcel Maehren, Marcus Brinkmann, and Jörg Schwenk

Ruhr University Bochum

A Artifact Appendix

A.1 Abstract

In our paper, we developed probes to explore how different QUIC server implementations handle the coalescence and fragmentation of payloads, covering both valid and invalid combinations of datagrams, packets, and frames. Already at this basic level, we observe significant differences between implementations, some of which pointing towards exploitable vulnerabilities. Previous QUIC research tools were not designed to implement such probes. To address this limitation, we presented *QUIC-Attacker*, a testing framework that provides maximum freedom on the sending side of QUIC.

This artifact appendix aims to show the functionality of *QUIC-Attacker*, which we developed to conduct our study. The experiments involve the reproduction of our most important findings, including the eight DoS vulnerabilities caused by unhandled exceptions and the exploitable injection vulnerabilities in Kwik and XQUIC.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

We are not aware of any security, privacy, or ethical concerns arising from running the experiments described below.

A.2.2 How to access

We provide our artifacts via Zenodo.[1]

A.2.3 Hardware dependencies

At least 60 GB of free disk space is required to build Docker images.

A.2.4 Software dependencies

The experiments require the ability to run bash scripts and basic Linux CLI tools (curl, unzip, and grep). In addition, Python 3 and Docker must be installed as part of the setup. We tested this artifact on Ubuntu 24.04 LTS Desktop.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Download and unzip the `artifact_experiments.zip` achieve from the Zenodo repository:

`https://doi.org/10.5281/zenodo.21451785`

Then, run the `setup.sh` script from the root directory of the extracted artifact experiments directory:

```
./setup.sh all
```

The script performs the following steps:

1. Downloads all remaining archives from Zenodo.
2. Unzips the downloaded archives.
3. Builds QUIC-Attacker and the test suite.
4. Builds the Docker images required for the experiments.

A.3.2 Basic Test

Testing the QUIC-Attacker. To verify that QUIC-Attacker can be executed correctly, run the following command:

```
./setup.sh test-attacker
```

If everything works correctly, this should print the parameter list of QUIC-Attacker with usage instructions.

Testing the Test Suite. To verify that the test suite can be executed correctly, run the following command:

```
./setup.sh test-suite
```

If everything works correctly, this should print the parameter list of the test suite with usage instructions.

[1] `https://doi.org/10.5281/zenodo.21451785`

A.4 Evaluation workflow

A.4.1 Major Claims

We claim to provide QUIC-Attacker, which can be used to evaluate various QUIC implementations, especially with respect to coalescing and fragmentation. In addition, we claim to provide the QUIC-Docker-Library, which can be used to easily start different QUIC implementations with Docker. In the following, we exemplary describe our claims for our most important findings: eight DoS vulnerabilities caused by unhandled exceptions and two exploitable injection vulnerabilities.

(C1): We claim to have found eight crashes in three QUIC server implementations (LSQUIC, Neqo, and quiche^C). These crashes are described in Sections 7.2.3 and 7.3 of our main paper.

(C2): We claim to have found exploitable injection vulnerabilities in two QUIC server implementations (XQUIC and Kwik). These vulnerabilities are described in Sections 7.2.1 and 7.2.2 of our main paper.

(C3): We claim to provide a proof of concept exploit for the injection vulnerability in XQUIC (cf. Section 7.2.1 and Figure 4 of our main paper).

A.4.2 Experiments

(E1): *Executing the DoS analysis [10 human-minutes]: this experiment executes the required QUIC handshakes with QUIC-Attacker against three QUIC implementations to reproduce the presented crashes.*

Preparation: Run the `E1.sh` script from the root directory of the extracted artifact experiments directory:

```
./E1.sh start
```

The script starts the three QUIC servers.

Execution: Run the `E1.sh` script again to open the logs of the QUIC servers:

```
./E1.sh logs
```

Note that the logs are intentionally reduced to make the crashes easier to detect. In a second terminal, use the `E1.sh` script again to execute the required QUIC handshakes. Each of the following commands executes one QUIC handshake and must be run sequentially. The table below also maps each command to the tested implementation and the corresponding test identifier used in our main paper:

Command	Test	Implementation
`./E1.sh exp-1.1`	F7	Neqo
`./E1.sh exp-1.2`	S1	Neqo
`./E1.sh exp-1.3`	S2	Neqo
`./E1.sh exp-1.4`	S3	Neqo
`./E1.sh exp-1.5`	S1	quiche^C
`./E1.sh exp-1.6`	S2	quiche^C
`./E1.sh exp-1.7`	S3	quiche^C
`./E1.sh exp-1.8`	S3	LSQUIC

Results: For each execution, inspect the logs. Each executed QUIC handshake crashes the tested server and triggers a restart. The first four executions trigger crashes of Neqo, the next three executions trigger crashes of quiche^C, and the final execution triggers a restart of LSQUIC. Since the experiment consists of eight handshakes, you should observe eight restarts in total.

(E2): *Executing the injection vulnerability analysis [10 human-minutes]: this experiment executes a minimum set of required QUIC handshakes with QUIC-Attacker against two QUIC implementations to demonstrate the missing frame-packet validation.*

Preparation: Run the `E2.sh` script from the root directory of the extracted artifact experiments directory:

```
./E2.sh start
```

The script starts the two QUIC servers.

Execution: Use the `E2.sh` script again to execute the minimum set of required QUIC handshakes. Each command evaluates one QUIC implementation and must be run sequentially. The table below also maps each command to the tested implementation and the corresponding test identifier used in the main paper:

Command	Test	Implementation
`./E2.sh exp-2.1`	F1 & F4	Kwik
`./E2.sh exp-2.2`	F1 & F4	XQUIC

Results: Each execution performs four QUIC handshakes with the tested server.[2]

For Kwik, the script should produce the following:

```
1 FrameTests - INITIAL_PACKET(MAX_STREAM_DATA_FRAME): CC
2 FrameTests - HANDSHAKE_PACKET(MAX_STREAM_DATA_FRAME): CC +
      ↪ TLS FIN
3 FrameTests - INITIAL_PACKET(STREAM_FRAME): DONE + TLS FIN
4 FrameTests - HANDSHAKE_PACKET(STREAM_FRAME): DONE + TLS FIN
```

Each line summarizes the server's response to one tested frame-packet combination. All four combinations are forbidden, and a QUIC implementation must treat them as protocol violations by terminating the connection. However, Kwik terminates the connection with a `CC (CONNECTION_CLOSE)` frame only for the first two combinations. For the other two combinations, the handshake completes without such a frame. This behavior indicates missing frame-packet validation.

For XQUIC, the script should produce the following:

```
1 FrameTests - INITIAL_PACKET(MAX_STREAM_DATA_FRAME): DONE +
      ↪ TLS FIN
2 FrameTests - HANDSHAKE_PACKET(MAX_STREAM_DATA_FRAME): DONE +
      ↪ TLS FIN
3 FrameTests - INITIAL_PACKET(STREAM_FRAME): DONE + TLS FIN
4 FrameTests - HANDSHAKE_PACKET(STREAM_FRAME): DONE + TLS FIN
```

[2] A complete analysis requires testing all frame-packet combinations described in the main paper. For the artifact evaluation, we reduced the analysis to four combinations because these are sufficient to observe a deviation from the specification.

In all four combinations, XQUIC neither sends a `CC (CONNECTION_CLOSE)` frame nor otherwise terminates the connection. This behavior also indicates missing frame-packet validation.

(E3): *Executing the proof of concept exploit [5 human-minutes]: this experiment executes the presented confusion attack against the XQUIC implementation.*

Preparation: The Chrome container in this experiment needs access to the host's X server to display its graphical interface. For this purpose, grant local Docker containers access to the X server by running:

```
xhost +local:docker
```

Execution: Run the `E3.sh` script from the root directory of the extracted artifact experiments directory:

```
./E3.sh start
```

The script start the XQUIC server, the Chrome browser, and the attacker script. Note, that Chrome opens an initial welcome window, which can be skipped. It then automatically issues a `GET` request to `/cat` on the server.

Results: Due to the injected frame by the attacker, the response delivered to the browser corresponds to `/dog` instead. This is visible in the browser, which displays the content `DOG DOG DOG...` instead of the expected `CAT CAT CAT...`.

A.5 Notes on Reusability

We contribute QUIC-Attacker as part of the TLS-Attacker suite (https://github.com/tls-attacker/TLS-Attacker). Our extensions have been merged and released as part of TLS-Attacker v7.8.0 (https://github.com/tls-attacker/TLS-Attacker/releases/tag/v7.8.0). We also provide the QUIC-Docker-Library on GitHub (https://github.com/tls-attacker/QUIC-Docker-Library). Both projects will be maintained and extended in their respective repositories. For updates, please refer to the GitHub repositories.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.