QUICforge: Client-side Request Forgery in QUIC

Yuri Gbur
Technische Universität Berlin
Berlin, Germany
yuri.gbur@posteo.de

Florian Tschorsch
Technische Universität Berlin
Berlin, Germany
florian.tschorsch@tu-berlin.de

***Abstract*—The QUIC protocol is gaining more and more traction through its recent standardization and the rising interest by various big tech companies, developing new implementations. QUIC promises to make security and privacy a first-class citizen; yet, challenging these claims is of utmost importance. To this end, this paper provides an initial analysis of client-side request forgery attacks that directly emerge from the QUIC protocol design and not from common vulnerabilities. In particular, we investigate three request forgery attack modalities with respect to their capabilities to be used for protocol impersonation and traffic amplification. We analyze the controllable attack space of the respective protocol messages and demonstrate that one of the attack modalities can indeed be utilized to impersonate other UDP-based protocols, e.g., DNS requests. Furthermore, we identify traffic amplification vectors. Although the QUIC protocol specification states anti-amplification limits, our evaluation of 13 QUIC server implementations shows that in some cases these mitigations are missing or insufficiently implemented. Lastly, we propose mitigation approaches for protocol impersonation and discuss ambiguities in the specification.**

I. Introduction

The QUIC protocol is an innovative development of transport layer stream abstraction. It combines the capabilities of TCP and TLS 1.3 to reduce the amount of required round-trip times (RTTs) during the connection setup. It can achieve a true 0-RTT connection setup for known endpoints, improving performance in high-latency networks [1]. With recent standardization efforts of the QUIC protocol by the IETF in 2021 [2]–[5] and through the support of many well known companies like Apple, Cloudflare, Facebook, Google, and Mozilla, QUIC is gaining more traction. Lastly, QUIC's importance increased by choosing it to be the core protocol of the new HTTP/3 standard. The adoption of QUIC results in one of the biggest changes to the web's protocol stack [6] and spawned the development of various new implementations [7].

In order to achieve compatibility with the Internet protocol stack, QUIC was built on top of UDP [3]. While providing transport layer functionality, QUIC is technically an application layer protocol with its own addressing scheme [1]. QUIC's addressing allows the underlying UDP port and IP address to change, while the connection persists. The QUIC protocol handles the migration of endpoints. To this end, a server has to send UDP datagrams to an unknown endpoint. The same holds true for the handshake, where the first message is always directed to an unvalidated endpoint.

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23072
www.ndss-symposium.org

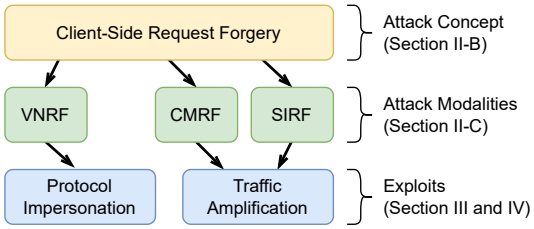


Fig. 1: Scope of the paper: feasibility analysis of client-side request forgery (RF) for three RF attack modalities, which can be used for two different exploits.

As a consequence, QUIC seems particularly vulnerable to address spoofing and request forgery. The specification acknowledges the vulnerabilities and provides first security considerations [3]. While the importance of QUIC is undeniable, research on attacks in general and for request forgery in QUIC in particular is still in its infancy [7]–[14]. In particular, the manifold new implementations together with a rather new protocol design require more in-depth security analysis.

In this paper, we take a detailed look at *client-side request forgery in QUIC* and provide the first feasibility analysis. To this end, we focus on request forgery attacks initiated by a QUIC client (the attacker). In this scenario, request forgery induces a QUIC server (the victim) to send packets that the attacker controls. The attacker can use the server's position in the network to gain higher privileges (e.g., ambient authority) and to gain access to greater resources (e.g., bandwidth). Specifically, we evaluate two request forgery attack modalities introduced in the QUIC specification [3]. They are based on the connection migration (CMRF) and version negotiation mechanism (VNRF) in QUIC. In addition, we introduce a third request forgery attack modality utilizing server initial messages in the standard handshake (SIRF). They all result in the QUIC server to issue an "unintended" request to a target host.

We analyze the feasibility of client-side request forgery in QUIC for two main exploits: First, *protocol impersonation*. Since request forgery happens on the transport layer, it enables an attacker to mimic protocol messages of other application layer protocols, similar to cross-protocol request forgery. Second, *traffic amplification*. An attacker can utilize the imbalance of the size between forged messages and the messages required to trigger the request forgery in amplification denial of service (DoS) scenarios. A summary of our paper's scope is illustrated in Figure 1.

As a result, we provide an analysis of the controllable attack space. In particular, we show that request forgery exploiting the version negotiation is indeed vulnerable to protocol impersonation. To this end, we provide a proof of concept of the attack vector, which can induce the victim to send valid DNS requests. We further evaluate 13 open-source QUIC implementations and show that they are generally vulnerable to at least one request forgery attack modality. Besides the general vulnerabilities, we identify pitfalls in the specification that lead to ambiguities, which are demonstrated by non-compliance with anti-amplification limits in nine of the 13 implementations. Lastly, we discuss changes to the specification that can mitigate the impact of request forgery attacks.

The remainder is structured as follows. In Section II, we introduce our threat model and describe three request forgery attack modalities in QUIC. With these modalities in mind, we analyze the QUIC protocol's vulnerability to protocol impersonation in Section III and to traffic amplification in Section IV. In Section V, we evaluate the attack modalities against 13 open source implementations followed by a general discussion in Section VI. Before we conclude our work in Section VIII, we discuss related work in Section VII.

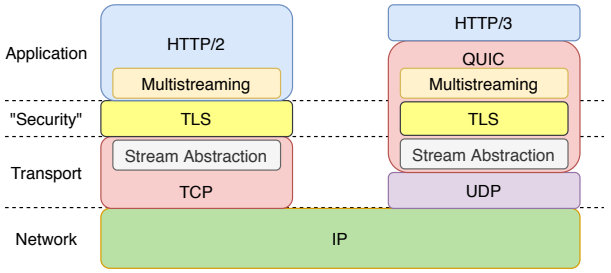


Fig. 2: HTTP/2 and HTTP/3 Protocol Stack Comparison.

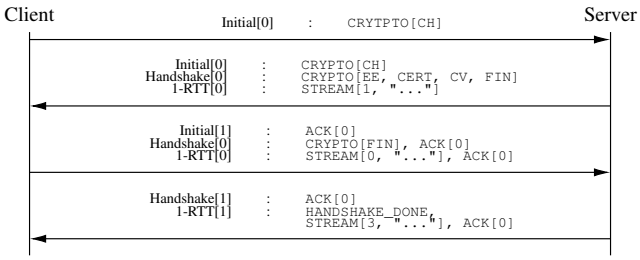


Fig. 3: QUIC 1-RTT handshake [3].

II. Request Forgery in QUIC

In this section, we provide the relevant groundwork on and describe three client-side request forgery attack modalities in QUIC. While connection migration request forgery (CMRF) and version negotiation request forgery (VNRF) are based on security considerations of the QUIC specification, server initial request forgery (SIRF) is a new modality, which we introduce here. We utilize the terms endpoint, QUIC packet, frame, address, CID, and stream as they are defined in RFC 9000 [3].

A. QUIC Basics

The adoption of QUIC results in one of the biggest changes to the web's protocol stack illustrated in Fig. 2. QUIC is a connection oriented protocol, i.e., the client and server have a shared state that is used to provide data reliably and in order to the application. To identify connections, QUIC introduces connection IDs (CIDs) as endpoint identifiers to allow for network-path changes of the UDP port and IP address. In the current version, CIDs are variable in length, but not more than 20 bytes, and must not contain any information that can be used to correlate CIDs. Therefore, all CIDs for a connection have to be generated independently at random. The length of a CID is communicated in the handshake or over `NEW_CONNECTION_ID` frames during the connection. Regular packets have a short header format containing the destination CID (DCID) only. Thus, the endpoint needs to remember the corresponding lengths. A peer can store multiple CIDs for future use for a connection.

The QUIC handshake(Fig. 3) combines the transport layer handshake and the TLS cryptographic handshake. The initial packets resemble the 3-way handshake of TCP, while the TLS parameters are carried "piggybacked" in `CRYPTO` frames. All packets in the handshake use the long header format and contain a source CID (SCID) as well as a DCID with the corresponding lengths. As an alternative to a server's initial packet, the server can send a retry packet containing a token. This token has to be mirrored in a second initial packet from the client to validate the path. The normal handshake without a retry does also perform path validation implicitly. By using retry packets, however, the server can avoid sending the significantly larger cryptographic information to an unvalidated endpoint [3]. If a server receives an unknown version, it will answer with a version negotiation packet, providing a list of supported versions. Version negotiation packets must always have the version identifier `0x00000000` [3].

One fairly unique feature of QUIC is connection migration. The usage of CIDs allows connections to survive endpoint address changes after a connection is established. If a migrated endpoint is detected by the server, it has to perform a path validation to the new host. To validate the path, the server sends a `PATH_CHALLENGE` frame, containing a token that has to be mirrored by the client in a `PATH_RESPONSE` frame [3].

B. System and Threat Model

Request forgery attacks occur when an attacker is able to trigger a host (victim) to send one or more "unintended" network requests to another host (target). Generally, this is achieved by abusing protocol features or by abusing logic flaws in an application [15]–[17]. In this paper, we explore request forgery attacks initiated by a *client*, i.e., client-side request forgery. Therefore, the words client and attacker as well as server and victim are used synonymously. An attacker can leverage the request forgery for achieving two goals, which are illustrated in Fig. 4: First, utilizing the higher authority of the victim, i.e., internal/restricted network access or higher privileges. Second, utilizing the higher bandwidth available from the server to a target.

For our attacks, we assume that the attacker is able to fully control the content of packets sent to the victim, including IP address and port spoofing. We restrict the attacker to modifications of messages that are still understood by the

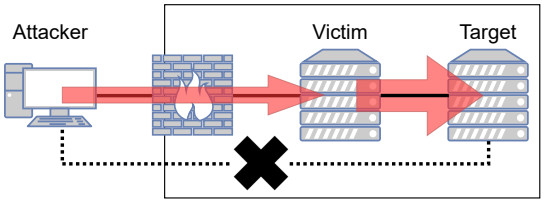


Fig. 4: Threat model showing the advantages an attacker can obtain through client-side request forgery, i.e., gaining higher privileges or greater bandwidth.

victim as valid QUIC packets to emphasize that the examined vulnerabilities stem primarily from the protocol design. While we assume that the victim server speaks and understands QUIC [3], we acknowledge that bugs and failures can occur in the individual server implementations. The target does not need to be capable of speaking QUIC but at least one UDP port expects incoming datagrams. While the target might not be directly reachable from the attacker, the victim must be able to reach it.

C. Request Forgery Attack Modalities

The following sections introduce three client-side request forgery techniques, namely server initial request forgery (SIRF), version negotiation request forgery (VNRF), and connection migration request forgery (CMRF). If not stated otherwise, we assume that the model mentioned above holds true.

1) Server Initial Request Forgery (SIRF): SIRF is probably the most basic request forgery technique for a client in QUIC. Yet, the described technique is currently not mentioned in the QUIC specification. Our concept of a SIRF attack follows the steps depicted in Fig. 5a. The attacking client initiates a QUIC handshake with the victim server according to the protocol definition, using a version the server supports. The source IP and the port, however, are directly spoofed for the first packet. Therefore, the victim assumes that the new connection attempt comes from the target host and continues the handshake by sending a server initial packet or retry packet [3].

2) Version Negotiation Request Forgery (VNRF): VNRF is similar to SIRF but abuses another variant of the QUIC handshake: A QUIC server responds with a version negotiation packet, if an unknown version is contained in the client's initial packet. A malicious client can send a non-existing version identifier to reliably trigger the version negotiation functionality. If the client furthermore spoofs the source of the datagram, the version negotiation is sent to the target, similar to SIRF. The entire message flow is shown in Fig. 5b.

3) Connection Migration Request Forgery (CMRF): This last request forgery technique utilizes QUIC's connection migration functionality. The server is not able to detect if a migrated address comes from a real migration of a client or if the source address was spoofed. In both cases, a `PATH_CHALLENGE` is sent. This allows an attacker to initiate the sending of a UDP datagram, containing a QUIC packet with at least one `PATH_CHALLENGE` frame to an arbitrary IP address and port as shown in Fig. 5c. To perform a CMRF, the attacker needs to initiate a new connection and completes the handshake as intended. The server has to consume a fresh CID from the client CID pool for new paths. Therefore, the original connection needs to stay on the initial endpoints until the server has received at least one `NEW_CONNECTION_ID` frames from the client. When these prerequisites are met, the attacker spoofs the source address of an arbitrary packet. The server detecting the new address initiates the path validation, thereby sending a UDP packet to this address [3].

III. Protocol Impersonation

Since QUIC is technically an application layer protocol, an attacker can target other UDP-based protocols through request forgery. If she is able to control a sufficient amount of data in the forged QUIC packets, she might be able to imitate packets of other protocols. While in literature the attack itself is also called cross-protocol request forgery (CPRF) [18], we will use the term *protocol impersonation* to distinguish between the attack modalities (SIRF, VNRF, and CMRF) and the attack vectors in which they can be utilized.

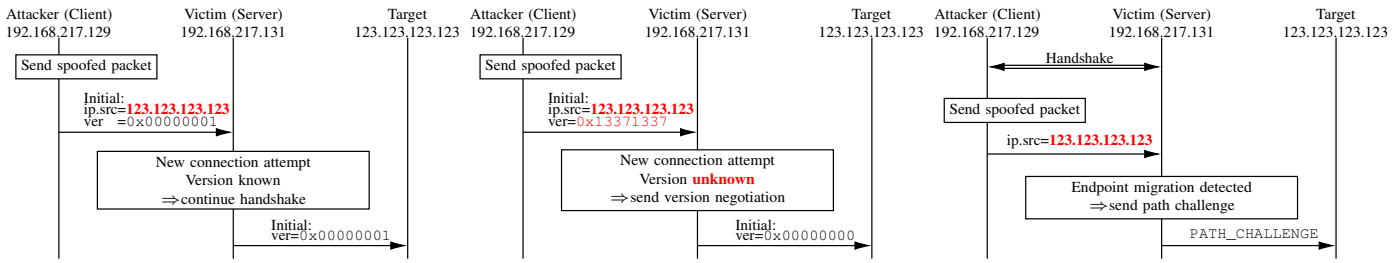
The unencrypted sections of a packet are the primary attack surface for protocol impersonation. An attacker might also be able to modify the ciphertext of encrypted sections [3], but these cryptographic attacks would surpass the scope of this paper. There might also be some parts of the packets that will change based on factors like client-server combinations, versions used, and networks conditions. However, our focus lies on data that can be manipulated *reliably* in most environments.

In the following, we analyze the controllable attack space, i.e., the controllable bits of short and long headers that relate to the attack modalities SIRF, VNRF, and CMRF. The results indicate that the controllable attack space of SIRF and CMRF are clearly limited. The controllable attack space of VNRF, however, is sufficient to perform protocol impersonation. We demonstrate this attack vector by developing a proof of concept for protocol impersonation, which forges valid DNS queries using VNRF.

A. Controllable Bits in Short Headers (CMRF)

During a CMRF attack, the path challenge(s) and possible padding are transmitted in packets with a short header [3]. Therefore, the amount of unencrypted, controllable data is very limited. Fig. 6 lists the detailed structure for short header packets. The first bit indicates the short header format and is fixed [3]. The same is true for the second bit that is required for interoperability with other transport layer protocols as defined in [19]. The spin bit is used for passive latency monitoring and exerts an unpredictable behavior. It can be turned off entirely or will randomly be turned off to ensure connections with disabled spin bit are also commonly observed on the network. Thus, we consider it as not reliably controllable. In the rare case that this single bit would enable a "real-world" attack, it might still be possible to utilize it by using multiple attempts to achieve the correct server mirroring of this bit [3].

Bits four and five are two reserved bits that have to be zero followed by the key phase bit that is used to identify the packet protection keys. All three bits are directly set by the server and not in control of an attacker. The final two bits of the first byte are interpreted as an unsigned integer and indicate the variable length of the packet number that comes after the DCID. The contained value is one less than the actual byte length of



(a) SIRF (b) VNRF (c) CMRF

Fig. 5: Client-side request forgery techniques in QUIC through spoofed source addresses.

```
1-RTT Packet {
        Header Form (1) = 0,
        Fixed Bit (1) = 1,
        Spin Bit (1),
        Reserved Bits (2),
        Key Phase (1),
        Packet Number Length (2),
        Destination Connection ID (0..160),
        Packet Number (8..32),
        Packet Payload (8..),
}
```

Fig. 6: Short header of QUIC packets [3].

```
Initial Packet {
        Header Form (1) = 1,
        Fixed Bit (1) = 1,
        Long Packet Type (2) = 0,
        Reserved Bits (2),
        Packet Number Length (2),
        Version (32),
        Destination Connection ID Length (8),
        Destination Connection ID (0..160),
        Source Connection ID Length (8),
        Source Connection ID (0..160),
        Token Length (i),
        Token (..),
        Length (i),
        Packet Number (8..32),
        Packet Payload (8..),
}
```

Fig. 7: Long header of QUIC initial packets [3].

the packet number field [3]. Assuming a connection can stay open as long as the attacker wants, the packet number can be influenced for CMRF by simply waiting until the packet number has reached a certain value prior to initiating the migration. The packet number length bits directly depend on the packet number, but they will stay the same as long as a value can be encoded into the same number of bytes [3]. The strong interconnection and dependencies of these header fields clearly reduce effective controllability.

The last remaining value is the DCID, which has a variable length integer between 0 and 20 bytes. The content of the DCID is fully controllable by the client. An attacker will likely choose a length of 20 bytes to maximize the controllable space. As an additional limitation, we need to consider that multiple CIDs sent to the server cannot contain the same value [3].

B. Controllable Bits in Long Headers (SIRF and VNRF)

SIRF and VNRF utilize message flows that are part of the QUIC handshake. Both attack techniques are therefore able to control more data than it is possible with CMRF as more unencrypted data is transferred. To this end, QUIC uses long headers. While the long header for initial and version negotiation packets share similarities, there exist distinct differences, which we will dissect separately in the following.

1) Initial Packets: Fig. 7 shows the long header for initial packets. The first bit is set to one indicating the long header format. The second fixed bit has to be one in order to allow QUIC to co-exist with other protocols similar to the short header [19]. The next two bits are used to indicate the different type of long header packets. Both are set to zero for initial packets, as are the reserved bits. Once more, the packet number length corresponds to the packet number variable length integer that follows later in the packet. They cannot be influenced in the SIRF use-case as the first messages of a connection setup will contain a freshly initialized counter [3]. Therefore, the first byte as well as two to four bytes at the end of the header are not controllable by an attacker.

The field following the packet number length encodes the version identifier in four bytes. Though this value is dynamic and can be set by the attacker, it has to be a predefined version that the server understands to avoid triggering version negotiation. Except for the rare occasion, where the impersonated protocol can contain exactly the same four bytes at this position in the payload, these bytes will not be usable for an impersonation attack. The version is followed by the CIDs for source and for destination. Each will be preceded by the corresponding length identifiers. During the QUIC handshake, the server will always send the DCID that was chosen by the client. If the attacker wants to maximize the amount of data controlled, she will most likely use the maximum size of 20 bytes, thereby giving the DCID length byte a static value of 20 (`0x14`) [3].

Concerning the SCID, sent by the server, there exist two scenarios. The server either chooses the CID proposed by the client in the first initial packet or sets its own CID. The first scenario, from now on referred to as SIRF+, results in the same amount of controllable bytes for the SCID as for the DCID. In the second scenario, an attacker will not have any control over the SCID bytes and length chosen, reducing the versatility of the attack. Since many of the tested implementations choose a static length for the SCIDs, the positioning of the surrounding bytes stays, at least, predictable.

```
Version Negotiation Packet {
        Header Form (1) = 1,
        Unused (7),
        Version (32) = 0,
        Destination Connection ID Length (8),
        Destination Connection ID (0..2040),
        Source Connection ID Length (8),
        Source Connection ID (0..2040),
        Supported Version (32) ...,
}
```

Fig. 8: Long header of QUIC version negotiation packets [3].

Server initial packets must always contain a token length of zero, as tokens are only used by the client (e.g., after a receiving a retry packet). Therefore, the token field is non-existent, resulting in both fields being reduced to one uncontrollable zero byte. Finally, the length parameter encodes the remaining packet payload length including the packet number. Its length may vary depending on the payload content (e.g., chosen TLS parameters) as well as depending on the included padding [3]. Nevertheless, it is also not controllable by the client.

2) Version Negotiation Packets: Version negotiation packets have a more simple structure than initial packets, shown in Fig. 8. As before, the first bit indicates the long header format. The next seven bits are unused in version negotiation packets and can be set to an arbitrary value by the server [3]. To adhere to RFC 7983 [19], the most significant bit (MSB) of the unused bits should be set to one although it is the only QUIC packet where it can be zero in theory. Nevertheless, looking from an attacker's perspective, the whole seven bits are uncontrollable for protocol impersonation with VNRF. The four version bytes are all set to zero in a version negotiation packet (`0x00000000`). Besides the CIDs, the only remaining part is an array of four byte values containing the identifiers for supported versions, which cannot be influenced by an attacker [3].

Again, the only remaining parts are the CIDs. While the controllable amount of 20 bytes is quite limited for CMRF and SIRF, the CIDs are special for version negotiation. To allow adjustments in the definition of future QUIC versions, the CID lengths are not restricted to 20 bytes. A future QUIC version could use the full range of lengths encodable in the eight bit length identifier resulting in a maximum length of 255 bytes for each CID. Through the possibility of using the full range of lengths, the CID length byte can be fully controlled [3]. An attacker needs to weigh up the advantages of fully controlling these bytes against controlling more bytes in total.

Another unique requirement for version negotiation packets is that a server must mirror both the DCID and the SCID sent by the client (switched for their purpose) [3]. Therefore, the attacker has the ability to always utilize both CIDs for protocol impersonation. Considering the extended length bytes, this results in up to 512 controllable bytes preceded by five and followed by at least four uncontrollable bytes.

C. Comparison of Controllable Attack Space

Our analysis shows that VNRF is the most versatile modality to mount an impersonation attack. The full comparison of all attack modalities is shown in Fig. 9. SIRF,SIRF+, and CMRF have two primary obstacles when trying to craft a payload. First,

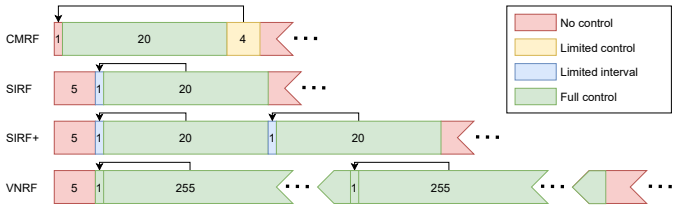


Fig. 9: Comparison of the maximum controllable bytes for protocol impersonation with CMRF, SIRF, SIRF+, and VNRF. Limited control means that the value cannot directly be chosen but can be influenced by actions of the attacker. Limited intervals means that the value cannot be chosen from the entire range that is encodable with the given bytes. Variable lengths that depend on previous values are indicated by arrows.

the overall length of 20 bytes (or in some rare cases 40 bytes) will not suffice to imitate most protocol messages. Second, the very limited byte-range and control for the packet length and CID lengths are not sufficient to manipulate meaningful bytes in the impersonated context. We, therefore, conclude that the capabilities of SIRF,SIRF+, and CMRF are clearly limited for protocol impersonation attacks. In all scenarios, including VNRF, an attacker has to work around the first few header bytes. However, as we will show in the next section, this alone is not enough to prevent impersonation with VNRF.

D. Protocol Impersonation with VNRF

As a proof of concept, our goal in this section is to craft a datagram that can be sent through VNRF to a DNS server and triggers a valid DNS response to the victim for the domain `tu-berlin.de`. We chose to impersonate the DNS protocol, as it is one of the most well known UDP-based protocols, widely used, and allowed in most networks [20]. As we will see, it can also be utilized around the restrictions by the static parts of the version negotiation packet.

Fig. 10 shows the beginning of the handcrafted packet bytes with the QUIC interpretation (above) and DNS interpretation (below). The first byte of the QUIC packet will start with a one, to indicate the long header followed by seven bits with random value. This byte plus the first zero-byte of the version identifier will be interpreted as the query ID. The next two zero-bytes of the version number are interpreted as the DNS flags defined in RFC 1035 [21]. The DNS flags for the two zero-bytes indicate a standard query that is not truncated with no recursion desired. This is a valid flag setting for DNS queries. The last version byte is the first byte of the number of host queries contained in the DNS query. This static zero bytes limit the maximum number of queries, but the remaining second byte will be more than sufficient in most scenarios [21].

The second byte of the number of queries is determined by the DCID length of the QUIC version negotiation packet. For our payload, we chose a value of seven to keep the amount of required hostnames in the query to a minimum, while still being able to skip the remaining bytes of the answer number (`Ans`), the number of authority records (`Auth`), and the additional records number (`Add`). The `Ans` and `Auth` bytes should be zero for a normal query and are not usable to extend the payload. The number of additional records is usually zero for a standard

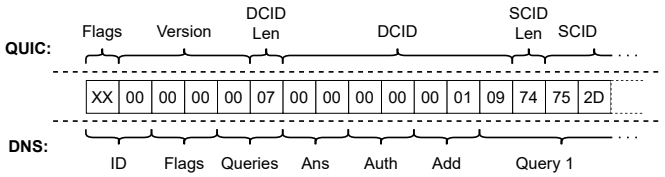


Fig. 10: Byte mapping of a QUIC version negotiation packet to a DNS query. Bytes are depicted in hexadecimal notation.

DNS query. We, however, set the two `Add` bytes to `0x0001` to deal with the remaining version number identifiers of the version negotiation packet that follow the CIDs [3], [21].

With the chosen DCID length of seven, the SCID length is the first byte of the hostname `tu-berlin.de`. The preceding byte `0x09` labels the number of bytes for the following domain level. Each domain level is indicated by a length octet and the top-level domain is terminated by a zero-byte. With the SCID length set to *t* (`0x74`), there are 116 remaining bytes of SCID that can be utilized for the remaining payload. This size is sufficient to include the entire query for `tu-berlin.de`.

Six further queries to the root domain were added to have the required seven queries in total. The hostnames for padding can be arbitrary and are only required to adhere to the DNS specification. The root domain is most suitable to our needs as it consumes as little payload space as possible. The `Add` section query entry was set to the domain root (`0x00`) and the type and class was set to zero. The length of the `Add` entry is set to the length of the remaining SCID payload plus the length of the version identifier array in the version negotiation packet. The amount of version identifiers advertised by the server is static and can be determined by triggering a version negotiation without a spoofed address. The length of the array is multiplied by four, because version identifiers are always 4 byte values and the `Add` entry length is given in bytes. The remaining payload space between the beginning of the `Add` entry and the version identifiers are filled with random bytes. By encoding the remaining QUIC payload in the described manner, the additional record will not make sense to a DNS server. Yet, the whole packet bytes are covered and the forged request is a valid DNS request [21].

Fig. 11 shows the packet capture of a VNRF-based protocol impersonation with a payload as described above. To demonstrate the validity of the forged packet, the entire QUIC traffic (left) is also decoded as DNS (right) in Wireshark. To execute a real DNS query, the spoofed address was set to the Google DNS server 8.8.8.8:53. Accordingly, Fig. 11a shows the packet types, Fig. 11b shows the first header byte and version, and Fig. 11c shows the CIDs in the payload. While the initial packet is a malformed DNS packet, the right pane shows that the version negotiation packet is indeed interpreted as a valid DNS request to `tu-berlin.de` (Fig. 11d) and results in a valid DNS response containing the IP addresses (see Fig. 11e).

This approach can be generalized to most domain names. The only thing that changes is the primarily queried domain name, which affects the length of the remaining payload. If a domain begins with a character or number (in ASCII) that translates to smaller bytes (e.g., 0=`0x30`) the length becomes limited. That is, if the the overall length exceeds this value, the exploit will not work. It, however, can be assumed that domain names meeting these special conditions are fairly rare.

We include the generalized proof of concept, i.e., forge DNS requests for arbitrary domain names, in our attack script which can be found in our accompanied repository [22]. The only parameter that has to be set manually is the number of versions supported by the attacked QUIC server. For protocol impersonation of DNS requests to return the domain's IP address, the DNS server must be a recursive resolver or must have a respective cached domain entry.

The proof of concept above shows that protocol impersonation with VNRF can be used with a "real-world" protocol. For example, such DNS queries could potentially be utilized to reduce uncertainty in the timing of DNS cache poisoning attacks. Due to the uncovered restrictions, creating a payload for other protocols than DNS will likely require a lot of debugging and manual tweaking to find a valid combination of bytes. There will be some payloads that cannot be realized within the existing boundaries, as described above. Nevertheless, we are convinced that a number of correct datagrams can be crafted through multiple iterations and creativity in utilizing the specification of the targeted protocol. As previously discussed, though, the additional restrictions of CMRF and SIRF make them unfeasible in the context of protocol impersonation.

E. Mitigation: Reducing Controllability

As the ability to perform request forgery stems from the protocol design, there is no inherent mechanism to fully avoid it. We propose the following protocol changes to reduce the controllability of the payload for CMRF, SIRF, and VNRF.

1) CID Reflection: A server should always choose a new CID in the handshake as its own DCID, which does not require changes to the specification. Besides deploying a fresh CID, servers should also select the length parameter randomly to reduce predictability for payload placement. The length has to remain within the specified limits and has to be large enough to ensure a sufficient entropy of the CIDs.

As shown, the version negotiation mechanism at its current state poses the biggest threat to protocol impersonation. While it is appreciable to provide room for changes in future versions, i.e., variable CID lengths in version negotiation packets, the mechanism requires some revision. One could rely on removing the mirroring of the client-provided DCID as described above. While this halves the amount of controllable bytes, it does not entirely mitigate the issue of controllable bytes. Therefore, we further propose the following approaches in addition to not mirroring the DCID proposed by the client.

2) Hash-based CID Generation: For the remaining client-controlled CIDs, we propose a mechanism that gives the server control over the client-controlled bytes of the DCID. A server could always (or at least for unvalidated paths) reflect the return value of a one-way (hash) function for a newly proposed CID. Thereby, the client could still influence the "randomness" of the CID but cannot control the content anymore. In order to utilize such a secured value for protocol impersonation, it would require an attacker to calculate the inverse for a payload, which should be infeasible for one-way functions [23].

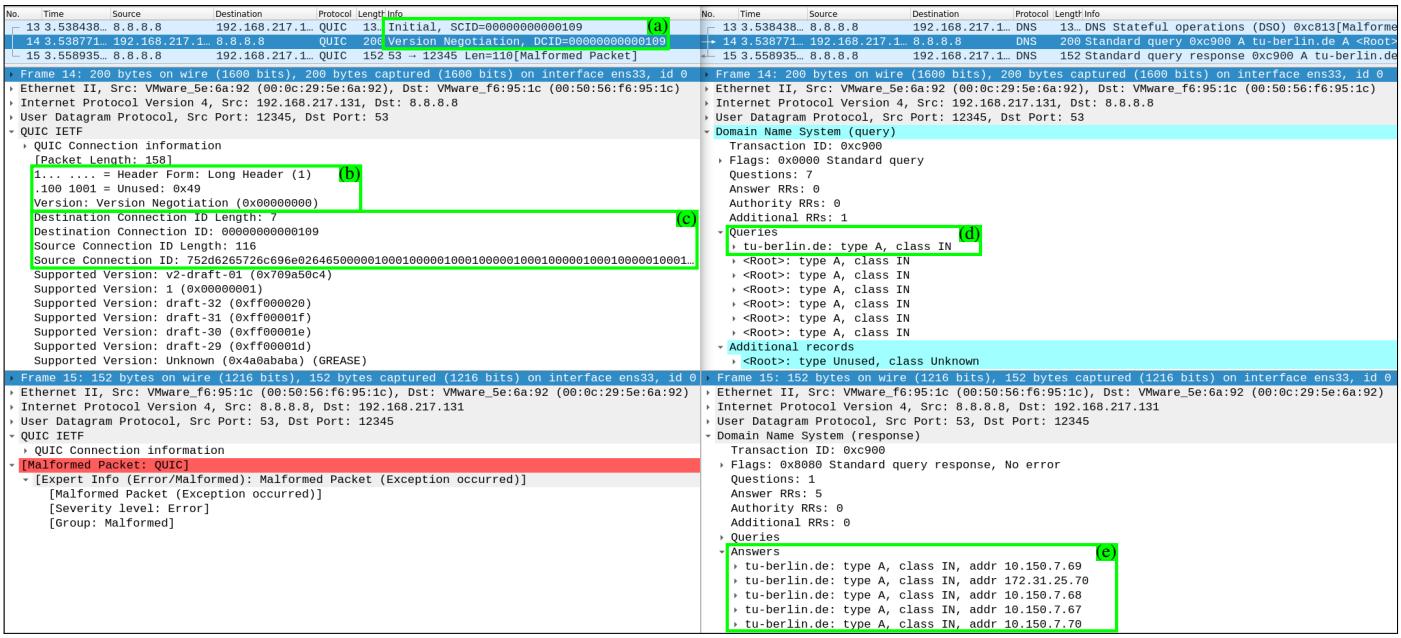


Fig. 11: Wireshark traffic capture of a forged DNS request using VNRF-based protocol impersonation; the QUIC interpretation is shown on the left and the DNS interpretation on the right.

Yet, the mechanism might conflict with routing and load balancing that is based on CIDs and the ability of a client to control them. Deployments might require consistency in length or for certain parts of a CID to route the packets to the correct endpoint [3]. Mechanisms that rely on static lengths or parts of a CID should be avoided in our opinion, as they make it easier to correlate connections. For hashed CIDs, the client is also able to calculate the value beforehand and use it for certain routing strategies.

3) Masking: Another approach that does leave control in the hands of the client is a masking mechanisms. A similar approach is used for client-to-server masking in Websockets [24], [25]. The QUIC headers need to be extended by a field that contains a masking value (e.g., 32-bit). This masking value is randomly generated by the server and the entire remaining header is XORed with a mask generated form this value. With this masking strategy, a client is still able to choose the DCID reflected by the server and it can base routing and load balancing on the value if required. However, as the masking is chosen by the server, the resulting payload of a request forgery attack is no longer controllable by an attacker. The data received by the target will appear random.

IV. Traffic Amplification

Another impact of request forgery attacks can be traffic amplification, which we will analyze in this section. The conditions for traffic amplification attacks arise if the forged packets are larger than the ones sent by the attacker. Such amplification attacks are measured by the bandwidth amplification factor (BAF), which is usually calculated as follows [26]:

$$\text{BAF} = \frac{\text{\# bytes from victim to target}}{\text{\# bytes from attacker to victim}}$$

The QUIC protocol specifies an anti-amplification limit that requires to not send more than three times the amount of data received on any unvalidated path [3]. To be able to directly compare amplification through request forgery with this limit within QUIC, we define and use a path amplification factor (PAF) for our measurements:

$$\text{PAF} = \frac{\text{\# bytes from victim to target}}{\text{\# bytes from attacker to victim with spoofed address}}$$

For SIRF, the PAF value is similar to the classical BAF definition. For CMRF, the packets sent by the attacker during the handshake are not calculated in the PAF. To get the corresponding BAF these bytes have to be added to the divisor. In the following, we first describe potential amplification vectors, before discussing mitigations.

A. Amplification Vectors

1) Minimum Path Requirements: The specification requires that "QUIC *must not* be used if the network path cannot support a maximum datagram size of at least 1200 bytes" [3]. Therefore, initial packets and packets containing `PATH_CHALLENGES` should be padded to 1200 bytes to perform Path Maximum Transmission Unit Discovery (PMTUD), ensuring that the path supports datagrams that are large enough to support QUIC. Despite the requirements of being able to transmit 1200 byte datagrams, some packets can actually be quite small during a connection. We have observed datagrams containing only one QUIC packet with one `ACK` frame that were as small as 73 bytes. If a server would send a padded path challenge with 1200 bytes for such a datagram on a new path, the BAF will be $\frac{1200}{73} \approx 16.44$, violating the anti-amplification rule.

Addressing this issue, the QUIC specification allows a first initial path validation without or with less padding, if anti-amplification limits cannot be met under the PMTUD requirements. In this case, an additional path validation has to be performed as soon as the path is successfully validated in

the sense that the endpoint is a legitimate migrated client. The second padded validation ensures that PMTUD requirements are also met [3]. We have identified these varying size checks as an area that is prone to errors. Besides the padding requirements, a server is allowed to already start sending data to the new endpoint before the address is validated. If not checked correctly, the data sent can also violate the anti-amplification limit.

2) Unbalanced Handshake Sizes: For SIRF and VNRF, the handshake mechanism can be abused. A server has to send at least one client-initial packet (or retry) after receiving a connection attempt. The TLS parameters required by the server (e.g., certificates) are usually larger than the ones from the client. If the resulting datagram(s) is/are larger than 3600 bytes, it is impossible to adhere to the anti-amplification limit even though the client initial packets are padded to a size of 1200 bytes. The specification [3] does not explicitly mention how this situation should be resolved in general. It only states that during the handshake, the anti-amplification limit has to be respected. A client implementation that considers this issue could include supplementary padding into the initial packet in order to increase the amount of data the server is allowed to send. Retry and version negotiation packets are always smaller than the client initial packet due to their limited amount of content [3] and cannot be used for amplification.

3) Reliability: The impact of the introduced conflicts can be worsened through reliability mechanisms of QUIC. To ensure that the path challenge succeeds, a server might send multiple `PATH_CHALLENGE` frames in a burst. Alternatively, a server might re-send the path challenges, if no path response is received. If all these packets are padded, the amplification issue increases.

For normal unacknowledged QUIC packets of a connection, there is a timeout and retry mechanism (not to be confused with QUIC's tokenized retry). This mechanism must be disabled for the server initial packets, because they will nearly always conflict with the anti-amplification limit. To avoid a deadlock in such a situation, the client is obliged to implement a probe timeout (PTO) after which it has to send another initial message [3]. Yet, if the server performs retries, an attacker can amplify the issue of unbalanced handshake sizes.

B. Anti-Amplification Mechanisms

The QUIC specification, in principle, contains all necessary details to prevent traffic amplification as already mentioned above. However, the conflicts between PMTUD and anti-amplification limits can become a pitfall. In Section V, we will show that many QUIC implementations indeed struggle to avoid them. We therefore advise reorganizing and adjusting the QUIC specification as follows [3]:

1) The mitigations mentioned in the security considerations chapter only should be additionally mentioned and emphasized in the chapters introducing the related mechanisms.
2) The amount of required decisions during path validation and during the handshake should be reduced. For example, the path validation could always be performed in two steps. The first packet would validate the path without padding, while the second packet would only ensure the PMTUD limit by including a `PADDING` frame.

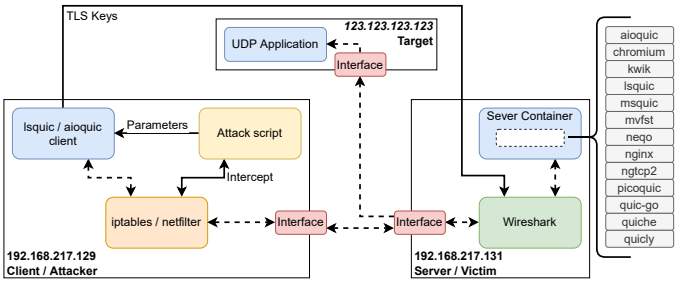


Fig. 12: The evaluation setup, showing the different involved components. The network traffic flow is shown as dashed lines and the control flow as solid lines.

3) Client implementations should be "encouraged" to include padding into initial packets. If a server is not able to respond to a client initial packet within the amplification limit, it should be obliged to send a retry to validate the path first.

Concerning unvalidated paths in the handshake, as exploited in SIRF, the specification mentions the retry mechanism as an effective protection against unnecessarily calculating the (expensive) key exchange information on the server. While our evaluation supports this statement and we propose the usage of retries as described above, we do not believe that this mechanism is preferable in most scenarios, since it introduces another RTT to the handshake. By doing so, it negates one of the primary "selling points" of QUIC, which is the latency reduced connection setup. This opinion is also supported by other research that comes to the same conclusion about reasonableness of this mechanism [8].

In summary, we consider the existing anti-amplification mechanisms sufficient to avoid amplification attacks if and only if it is also implemented correctly for all edge cases. In the following we evaluate how existing implementations cope with the identified amplification challenges and which of them are vulnerable to SIRF, VNRF and CMRF.

V. Evaluation

In this section, we evaluate a series of open-source QUIC implementations In particular, we evaluate their vulnerability to the outlined attack vectors and, thus, whether they adhere to the specification sufficiently. We start by introducing our attack setup (Fig. 12) and evaluating the vulnerability of each server implementations for each attack technique. For the servers that show amplification issues we perform a detailed analysis on their PAF values. Finally, we evaluate benefits and challenges of the mitigation approaches introduced in Section III-E.

A. Server Selection

We decided to use open source software only as the ability to comprehend the application logic is essential to detect vulnerabilities and to understand existing security features. The implementations are all listed in the QUIC working group's GitHub [27]. In our evaluation, we consider all listed implementations that support QUIC version 1. Unfortunately, we had to exclude some of the implementations due to interoperability issues and bugs in the implementations. The

remaining 13 implementations were integrated in our attack setup as described below. We acknowledge that some of the implementations are experimental or maintained by a small team, which stands in contrast to some other projects that are developed and supported by large companies with large teams. We also acknowledge that some implementations were not developed to be deployed in production. Nonetheless, we did include software from all categories as we believe it is important that they all adhere to the specification. The versions/commits used for each server are available in the respective docker containers (see below).

For the implementations that turned out to be not compliant with the specification, and therefore are prone to exploits, we informed the developers in a responsible disclosure. The respective contact persons received a draft version of our paper in August 2022 in order to give them enough time to address the issues, before the research will be publicly released.

B. Setup and Attack Implementation

On the client virtual machine (VM), we use a custom Python attack script that utilizes `netfilter` queues with the `NetfilterQueue` and `scapy` library to intercept and spoof packets. This approach allows us to spoof packets for arbitrary QUIC clients without altering the source code. For communicating over QUIC, the attack script primarily uses the `lsquic` HTTP client in release version 3.0.4. as it provides various command line options for different settings.

Although VNRF requires less interoperability settings, it needs some additional adjustments to the client implementation. Besides the integration of a non-existent version (i.e., `0x13371337` in our case), the extension of a CID up to a length of 255 bytes had to be implemented. We implemented the aforementioned adjustments into `aioquic`, as Python handles buffer length natively which minimized the programming efforts in contrast to the C-base `lsquic` implementation Please refer to our GitHub repository [22] for more information about the attack setup and implementation.

On the server VM, we chose to run the QUIC server binaries in a containerized Docker setup to avoid package conflicts and to create a more flexible setup. `chromium` has a special setup within the container that it is started in proxy mode and forwards requests to a simple python backend server. We were forced to deviate from the other setups as the chromium test server only supports proxy or cached mode and the cache mode did not work within our required parameters. All servers are deployed in their default configuration. The Dockerfiles, can be found in our GitHub repository [22]. Furthermore, we provide the used containers in our DockerHub [28] to provide an easy way to reproduce our experiments. Since most of the QUIC implementations provide HTTP servers as test binaries only, we have created large files (≈700MB) for each server that can be requested to keep a QUIC connection open if needed. The limitations and abilities of the client and server are in line with our threat model.

In order to gain more insight into the communication, we used a development version of Wireshark (v3.7.0) to monitor network traffic at the server with the latest QUIC packet analysis capabilities that are up to date with the specification development. If needed, the TLS encryption keys were exported from the `lsquic` client and loaded into Wireshark.

TABLE I: Evaluation of the three request forgery attacks against the 13 server implementations.

Client	CMRF				SIRF			VNRF	
	Vuln.	Pad.	New CID	PAF>3	Vuln.	PAF>3	Ref. CID	Vuln.	CID>20
aioquic	✓	✗	✗	✗	✓	✓	✗	✓	✗
chromium	✓	✗	✗	✓	✓	✓	✗	✓	✓
kwik	✗	-	-	-	✓	✗	✗	✓	✓
lsquic	✓	✓	✓	✓	✓	✗	✗	✓	✗
msquic	✓	✗	✓	✗	✓	✓	✗	✓	✓
mvfst	✓	✓	✗	✓	✓	✓	✗	✓	✗
neqo	✓	✗	✓	✓	✓	✓	✗	✓	✓
nginx	✓	✗	✓	✓	✓	✗	✗	✓	✗
ngtcp2	✓	✗	✗	✗	✓	✗	✗	✓	✓
picoquic	✓	✗	✓	✓	✓	✗	✗	✓	✓
quic-go	✗	-	-	-	✓*	✗	✗	✓	✓
quiche	✗	-	-	-	✓*	✗	✗	✓	✓
quicly	✗	-	-	-	✓	✓	✗	✓	✓
Total	**9**	**2**	**5**	**6**	**11**(13)	**6**	**0**	**13**	**9**

* Sends retry packet instead of server initial packet

C. Vulnerability Analysis of CMRF, SIRF, and VNRF

Tab. I shows relevant factors for the request forgery attacks for each implementation: The first column for each attack technique (Vuln.) indicates whether the implementation is generally vulnerable. The varying impact through protocol impersonation and traffic amplification is listed in the remaining columns for each of the three attack techniques. In summary, we observe that nine out of the 13 implementations are vulnerable to CMRF, and all 13 implementations are vulnerable to SIRF and VNRF.

The Padding column (Pad.) lists whether the server enlarges the first packet containing a `PATH_CHALLENGE` to 1200 bytes, as required for PMTUD. We observe three implementations performing padding up to this size. We will discuss our results on amplification (PAF $>$ 3) in more detail in Section V-D.

Besides padding and amplification, Tab. I lists the New CID, Ref. CID, and CID $>$ 20 column. The New CID column indicates whether a new CID is consumed by the server for a new path. In order to perform protocol impersonation with CMRF, an attacker has to know in advance which CID is used to be able to inject the payload into the correct one. Since QUIC requires that no CIDs are reused for one connection, it is not possible to transmit the same payload in the handshake and in `NEW_CONNECTION_ID` frames [3]. Only five of the implementations that support connection migration consume a new CID as required. This is itself a violation of the QUIC specification [3].

SIRF can control twice as many bytes if the DCID proposed by the client is mirrored by the server. The evaluation results of this behavior is indicated by the Ref. CID column. All tested implementations use a fresh CID, thereby limiting the controllable attack space. The last column, CID $>$ 20, indicates whether the server responds to a client initial packet with an unknown version identifier and CIDs longer than 20. If a server does not accept longer CID values, the protocol impersonation impact of VNRF is significantly reduced, but the implementation is violating the QUIC specification by doing

so (cf. Sec. III-D). Of the 13 evaluated implementations, nine responded to the unknown version `0x13371337` with CID lengths up to 255 bytes.

D. Traffic Amplification Analysis

In this section we evaluate the amplification that occurs on the spoofed paths. The individual amplification factors are depicted in Fig. 13. We used two PAF measurements for CMRF. The first value describes the amplification for only one spoofed packet containing a single `ACK` frame. For the second measurements, all pending frames from the client are also transmitted with the spoofed address. We continue the sending for up to three minutes or until the connection is terminated by one of the endpoints, e.g., due to a missing `PATH_RESPONSE`. For SIRF, there is one PAF measurement as the client always sends only one initial packet. If one of the PAF measurements surpasses an anti-amplification limit of three, the implementation was marked as vulnerable to amplification in Tab. I. Since version negotiation packets are always smaller than the initial packet from the client, no evaluation for VNRF was performed in regards to amplification. Fig. 13 further lists the actual BAF values for CMRF if they surpass they also surpass the anti-amplification limit to indicate "real-world" DoS viability.

`chromium`, `lsquic`, `mvfst`, `neqo`, and `picoquic` do not adhere to the anti-amplification limit of QUIC for CMRF. They do not limit the responses to three times the amount of data on an unvalidated path for a single spoofed packet, initiating the connection migration. Especially `chromium`, `lsquic`, and `mvfst` show significant PAF values up to 374.44. While `lsquic` launches multiple path challenges for redundancy and performs the 1200 bytes padding on the initial path challenge, `mvfst` already transfers too much stream data before the path is validated. `neqo` is the closest to adhering to the specification by first initiating a path challenge with no padding followed by a padded one. However, the padded challenge is sent prematurely without waiting for the response of the first one. Thus, the second path challenge violates the anti-amplification limit.

`chromium` induces a very unexpected behavior during the connection migration. It does not send path challenges at all but starts sending stream data directly. This results in significant amplification in both scenarios, as the `STREAM` frames are large in comparison to the pending acknowledgments. During the responsible disclosure process, we were able to identify the reason for the amplification together with the QUIC development team from Google. Due to compatibility issues with certain client implementations, the path validation is turned off by default and the older gQUIC mechanism is used. Both, path validation and strictly adhering to the amplification limit, can be turned on by setting respective compile flags. However, this does not affect the measured amplification factors as the compared default configuration stays vulnerable.

For `mvfst`, which also sends stream data prematurely, we observe that it also surpasses the anti-amplification limit. However, for `chromium` and `mvfst`, the PAF is lower as more data is sent through pending ACKs. Amplification does not occur for the remaining implementations when the pending packets are also sent, as the servers do not send an additional path challenge for each received packet.

Fig. 14 compares the sent bytes over time for the implementations that surpass the anti-amplification limit for CMRF. It shows that a majority of the data is sent directly after initiating the connection migrations in one big burst. `lsquic` additionally transfers retries of the initial path challenge, resulting in small chunks of additional data over time. This behavior could be less interesting for attackers utilizing amplification in a distributed DoS (DDoS) scenario. As not all of the amplification is performed immediately after the connection migration is initiated.

The byte rate for the total amplification is plotted in Fig. 14c. It shows that there is also a significant burst of data sent to the target but not a lot of data sent over time. The amplification for `chromium` happens delayed even though the spoofed packet was sent at the same time. The reason for this was not comprehensible for us but it could be an artifact of the proxy setup. The delay makes `chromium` less favorable for attacks as the connections to the victim potentially have to be kept open for longer.

For SIRF, most implementations stick to the anti-amplification limit. `aioquic`, `msquic` and `quicly` encounter the issue described in Section IV, i.e., the server initial packet slightly surpasses the allowed anti-amplification limit as the TLS parameters are too large. `mvfst` and `neqo` exert the faulty behavior concerning reliability in QUIC also introduced in Section IV. In these implementations, the standard retry mechanism for ongoing connections is also present for the handshake messages The server initial packet is sent multiple times resulting in high PAF (BAF) values for these three implementations.

`quic-go` and `quiche` are also special for the SIRF evaluation. Both server implementations send a retry packet instead of the server initial packet for new connection attempts to verify the path before the server initial packet is calculated. Retry packets are always smaller than the client initial packet and do not result in an amplification condition [3].

In total, all 13 implementations were affected by at least one of the request forgery techniques with varying impact. We were able to utilize protocol impersonation through VNRF with nine server implementations. As elaborated above, we argue that all 13 implementations should be vulnerable to this attack vector, if they would be compliant with the specification. The strong suit of VNRF is that it is based solely on the QUIC protocol definition. To this end, our results challenge the early perception by the QUIC working group that request forgery is not that serious [29]. We are convinced that request forgery in QUIC can have a significant impact. We demonstrated opportunities for violating the amplification limits within five of the implementations. The measured PAF values up to 374.44 surpass the anti-amplification limit by a significant amount. The actual BAF values usable for traffic amplification attacks were measured up to 18.28 for `chromium` (CMRF) and up to 22.1 for `mvfst` (SIRF). Even though the actual BAF values are surpassed by some other protocols, e.g., NTP or in some cases DNS [26], they will likely still be very relevant. As a core web protocol, QUIC will be readily available and widely admitted by firewalls. These factors make it more feasible for traffic amplification than other protocols.

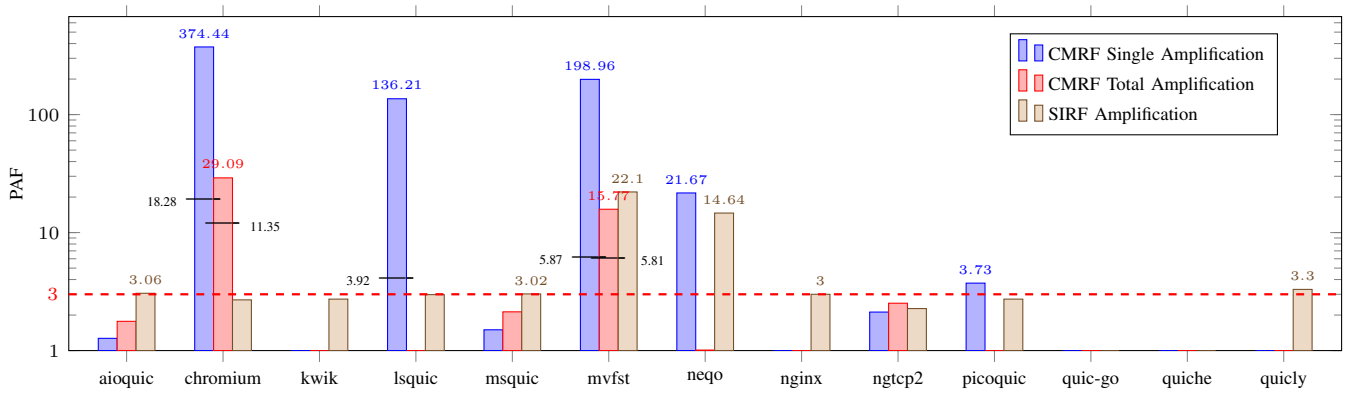
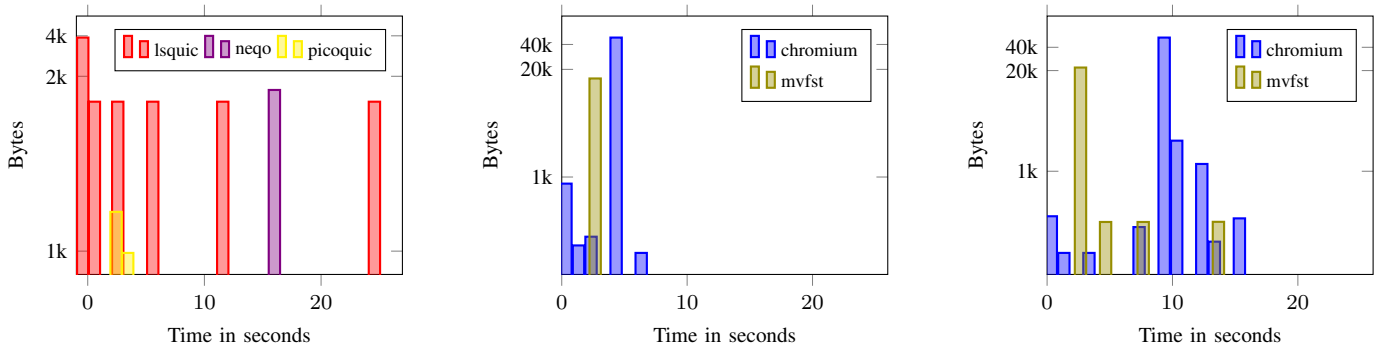


Fig. 13: Evaluation of the maximum PAF observed for CMRF and SIRF for each implementation. CMRF is evaluated through the amplification for only the initial path challenge and for all the packets sent to the spoofed address until the connection is terminated. The dotted red line at PAF = 3 depicts the anti-amplification limit required by QUIC. For servers where also the actual BAF values are larger then three, the BAF is depicted as a black line within the respective column.



(a) Single amplification for server with 10k bytes maximum peak.

(b) Single amplification for server with more than 10k bytes peak.

(c) Total amplification.

Fig. 14: Amplification rate over time with CMRF for server implementations violating the anti-amplification limit. Byte measurements are accumulated to one second intervals.

E. Removing Control of CIDs

In Section III-E we introduce two major approaches for reducing the controllabilitly of CIDs. Before evaluating the pro and cons for both techniques, we acknowledge that both mechanisms change the protocol significantly and will require a new QUIC version.

F. Hash-based CID Generation

As a proof of concept, we have implemented a hash-based CID generation for version negotiation for the `aioquic` server. Since version negotiation is separate from other messages flows, the CID handling could be altered without affecting the remaining protocol. As a one-way hash function, we decided to use the `SHAKE` sponge function. Since sponge functions are designed to support variable lengths, the are ideal for the QUIC CID scenario. The generation of 20 byte values is possible, while preserving the possibility for longer CIDs in the future. In our test implementation, we utilized `SHAKE256` as it is one of the underlying algorithms for the current `SHA3` standard and as many libraries support it [30].

In an experimental performance test, we compared `aioquic`'s CID generation based on `os.urandom()` with the hashing performance of `hashlib.shake_256()`. To this end, we generated 10,000 CIDs for increasing lengths up to 255 bytes and measured the the computation time. The average difference over all values was only around $859\,ns$, Based on our experience with the `aioquic` implementation, we are convinced that the effort to implement the described mechanisms is reasonable. As we use `SHA3`, all common programming languages have libraries that support `SHAKE256`. In most places of the codebase, the current CID variables can be replaced with the hashed value. The major difference is that the original value has to be passed to the generation of initial packets and to `NEW_CONNECTION_ID` frames.

G. Masking

Since masking requires more changes to the header structure, it is not easily integrable into current implementations. This makes the hash-based mitigation approach the favorable method from a development standpoint as it requires only minor changes to the codebase. A hash-based CID generation, however, has

a greater impact on the current capabilities of CIDs than masking. Certain routing and load balancing strategies using CIDs becomes impossible. From a performance perspective, we come to the conclusion that the hash-based CID generation is favorable as well. To this end, we performed a basic performance analysis, masking a common-sized initial header (50 bytes) with the `numpy.bitwise_xor()` function. 10,000 repetitions resulted in an average masking time of $702\,ns$. While XOR-ing is clearly faster than hashing, it has to be performed for every packet and not only on the few generated CIDs.

Unfortunately, both mitigation strategies break direct interoperability of the current and the new QUIC version. A final decision on which method is preferable is subject of future discussions. In order to maintain backwards compatibility, server implementations would still need to understand the original CID usage as part of version negotiation with older versions. If version negotiation is required to use the controllable CID design for older QUIC versions, VNRF as described above persists. Abandoning the current QUIC version to make the protocol resistant against request forgery comes with huge implications for integration and development.

VI. Discussion

In the following section, we discuss additional mitigation approaches through network controls and other possible attack scenarios through request forgery. In general, if an attacker wants to utilize the attacks vectors, she needs to determine which QUIC implementation is used by the server. Therefore, we also present first thoughts to identify and distinguish the different implementations.

A. Network Control

The mitigation approaches above are intentionally focused on in-protocol solutions. While they can drastically reduce the impact of request forgery, they do not entirely prevent that a forged UDP datagram can be transmitted to an arbitrary host, even if it contains uncontrollable data only. To completely avoid UDP based request forgery through QUIC, the only option for network operators is to utilize strong network controls. One approach would be to deploy a rate-liming for packets on unvalidated paths in order to reduce the impact of amplification vectors. As a second general measure, it would be advisable to limit the internal IP address ranges that a server is allowed to reach. Besides limiting IP addresses, a server could maintain a denylist of standardized UDP ports to avoid request forgery to common UDP based protocols. In general, it should be possible to block port zero up to 1023 as these system ports should never be used as client source ports. Yet, network operators should be aware that common stateful transport layer network controls in QUIC are less effective than they are for TCP based protocols [12]. Furthermore, many available deep-packet inspection tools lack maturity in regards of QUIC analysis [9] and load balancing tools struggle with the limited insights into packets [7]. Future research about securing networks in the presence of QUIC will be key to a wide adoption.

B. Server-Side Attacks

The focus of our paper lies on client-side attacks. An attacker acting as a server cannot utilize CMRF, SIRF, and VNRF. Connections are always initiated by the client, thereby SIRF and VNRF are prevented. CMRF is neither possible as connection migration is currently only allowed for clients.

The specification mentions one server-side attack that also directly stems from the protocol design. The technique abuses the `preferred_address` parameter in the handshake, specifying an address to which the client migrates after the handshake. For this approach, only the DCID is fully controllable for protocol impersonation, similar to the CMRF attack technique. The hash-based CID generation, introduced in Section III-E, and the anti-amplification limit would also mitigate the protocol impersonation and DoS attack scenarios for this server-side attack.

C. Protocol Impersonation via Encrypted Data

An additional protocol impersonation vector that was out of scope of this paper could be to influence the ciphertext of encrypted messages. As already mentioned, a server might already start sending application data after path challenge. Depending on how much of the cleartext is controllable an attacker might be able to create a ciphertext that is meaningful in the context of another protocol. If the cleartext is not controllable but a priori known, an attacker might still be able to influence the key-material sufficiently to achieve her goals for protocol impersonation. While we believe that there exists some interesting research concerning these cryptographic attacks, we also think that the "real-world" impact will be minor. They are often bound to very specific pre-requisites that might not be present in most scenarios [31]–[33].

D. Victim Detection

As discussed in our evaluation, the 13 implementations require different client settings and the impact will vary based on the targeted server software. Therefore, we are convinced that it would be beneficial to look at server/victim detection mechanisms in future research. General identification of QUIC usage can be performed efficiently by scanning IP addresses with client initial packets that trigger a version negotiation [34]. Existing fingerprinting approaches that aim to differentiate between various implementations primarily incorporate the trivial indicators from the handshake, e.g., like packet length, CID lengths, and CID changes [35]. Some of our evaluation results (cf. Section V-B) could be directly integrated into fingerprinting mechanisms to improve the accuracy. The varying interoperability settings and responses to the attack scenarios create a disjunct set of identifiers for each of the 13 implementations. Especially connection migration behavior could be a valuable addition to the existing approaches. Additionally, the TLS parameters of the handshake may be a valid extension as long as they are static for implementations and not configurable by administrators. As more individual HTTP/3 servers on the same QUIC libraries will be created, a hybrid approach between QUIC and HTTP/3 will probably have the most accurate results [36].

In addition to the general detection of a host's QUIC implementation, it would be necessary for an attacker to verify that the identified server is placed in a vulnerable environment. For our experiments we always had the advantage to have full insight into everything that happened on the attacker's, victim's,

and target's site. In a "real-world" scenario, an attacker would probably not be able to observe the victim and the target. Furthermore, the evaluated request forgery attacks are always blind, i.e., answers by an impersonated protocol will not be forwarded to the client.

One possibility to decide whether a server is vulnerable to request forgery is to target an attacker-owned machine. A successful attack will be visible by watching incoming packets on this machine. Yet, such an evaluation does not enable an attacker to evaluate whether mechanisms are in place that prevent the addressing of other target IPs (e.g., internal IPs). However, it could be a valid strategy to identify generally vulnerable servers in the wild.

VII. Related Work

QUIC gained significant attention in the past years, particularly with respect to security aspects. While some contributions started to investigate early versions of QUIC [37], [38], research on attacks in the current version is still sparse. Nawrocki et al. [8] performed an analysis of the current state of DoS attacks in QUIC. They also touched amplification mechanisms related to request forgery but discarded them as unfeasible. The general concepts for two of the attacks vectors that we evaluated, namely CMRF and VNRF, are considered in the QUIC specification [3]. To the best of our knowledge, however, our work is the first in-depth analysis of request forgery for QUIC.

The field of cross-protocol attacks for TCP was established in a paper by Jochen Topf [15]. The whitepaper by Tanner Prynn [18] is the only occurrence we found that investigates cross-protocol request forgery (CPRF) (here protocol impersonation) *and* mentions QUIC. The author concludes that QUIC is not usable in such kind of attacks as his approach (besides using an earlier QUIC version) concentrated on the control over the protocol from within a browser. We instead focus on a more general attack vector and consider general protocol alterations that can lead to protocol impersonation.

The fact that QUIC is built on top of UDP, while still inheriting transport layer functionality, introduces novel request forgery attacks, which would not be possible in a similar way in TCP/TLS. Well-researched application layer attacks like server-side request forgery (SSRF) and cross-site request forgery (CSRF) are primarily based on application logic flaws. The forgery capabilities in QUIC directly stem from the protocol design. UDP is therefore not covered in most request forgery research in terms of protocol impersonation. However, request forgery is very prominent with UDP for amplification DoS attacks. The work by Christian Rossow [26] provides an excellent overview of known amplification attack vectors in UDP-based protocols and their DoS impact. Specifically for QUIC, the results by Nawrocki et al. [8] confirm the feasibility of resource depletion attacks through client initial flooding by showing it being utilized in the wild. On-path 0-RTT attacks that utilize the packets that are not integrity protected are explored by Cao et al. [39]. In our work, we emphasize the importance of DoS research by providing a first analysis of amplification attacks utilizing QUIC.

As previously discussed, network controls play an important role in reducing the impact of request forgery attacks. Lehlan Decker [9] shows how common deep packet inspection tools cope with QUIC. He comes to the same conclusion as we did in our previous work [12] that the tools struggle in the presence of QUIC. Thimmaraju et al. [7] demonstrate related issues while load balancing QUIC traffic. Kühlewind et al. [40] propose an architectural solution to the transparency issues experienced by middleboxes. By introducing a "path layer" for transport independent signaling, middleboxes gain more insight into transport related information for fully encrypted protocols like QUIC. These insights in network controls are therefore orthogonal to our work and key to improve network security.

VIII. Conclusion

In this paper, we analyzed the feasibility of client-side request forgery attacks in QUIC by evaluating the capabilities and controllable attack space. We showed that the theoretical request forgery attack strategies, mentioned in the QUIC specification and introduced by us, can have a significant impact on the security of networks. The techniques analyzed are server initial request forgery (SIRF) and version negotiation request forgery (VNRF) based on the handshake, as well as connection migration request forgery (CMRF) based on the connection migration mechanism. We built a custom test environment and evaluated our attack vectors against 13 different QUIC server implementations. Our results indicate that all servers were affected by at least one of the request forgery techniques.

We discovered that protocol impersonation can only be utilized with VNRF as it enables an attacker to fully imitate packets of other protocols in the forged payload, e.g., DNS queries. The high impact through VNRF could be shown with nine implementations that had the required behavior implemented. However, the exploited behavior is well-defined and required by the current QUIC specification making every implementation vulnerable in theory.

Further discrepancies between the specification and the implementations were observed in other functionalities too. In multiple server implementations, the anti-amplification limits conflicted with network probing requirements or retry mechanisms were mistakenly used during the handshake. These issues result in significant path amplification factor (PAF) values of up to 374.44 for CMRF and 22.1 for SIRF. The related bandwidth amplification factor (BAF) values of 18.28 for CMRF and 22.1 for SIRF make DoS traffic amplification attacks feasible.

Based on the issues identified, we propose two mitigation strategies (hashing and masking) that remove the client's control over the forged payloads. In regards of DoS protection, the QUIC specification already contains sufficient anti-amplification mechanisms that are effective against the evaluated attack vectors. Efforts should be made to reduce ambiguity within the conflicts of packet duplication for reliability and network requirements.

QUIC is one of the more promising approaches to improve the network stack of the web. Yet, some features of the protocol are entirely new concepts and they are not yet scrutinized sufficiently to ensure that they meet the statements concerning their security properties. It will be therefore essential to perform further security research on QUIC before a wide adoption is advisable.

REFERENCES

[1] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 183–196.

[2] M. Thomson, "Version-Independent Properties of QUIC," RFC 8999, May 2021.

[3] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021.

[4] M. Thomson and S. Turner, "Using TLS to Secure QUIC," RFC 9001, May 2021.

[5] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," RFC 9002, May 2021.

[6] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-34, Feb. 2021, work in Progress.

[7] K. Thimmaraju and B. Scheuermann, "Count me if you can: Enumerating QUIC servers behind load balancers," *Conference on Networked Systems 2021 (NetSys 2021)*, Sep. 2021.

[8] M. Nawrocki, R. Hiesgen, T. C. Schmidt, and M. Wählisch, "Quicsand: Quantifying quic reconnaissance scans and dos flooding events," in *Proceedings of the 21st ACM Internet Measurement Conference*, ser. IMC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 283–291.

[9] L. Decker, "QUIC & The Dead: Which of the Most Common IDS/IPS Tools Can Best Identify QUIC Traffic," *SANS Institute*, May 2020.

[10] M. Thomson, "QUIC Security," 2020.

[11] J. Zhang, L. Yang, X. Gao, and Q. Wang, "Formal analysis of quic handshake protocol using proverif," in *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2020, pp. 132–138.

[12] K. Y. Gbur and F. Tschorsch, "A QUIC(K) way through your firewall?" *CoRR*, vol. abs/2107.05939, 2021. [Online]. Available: https://arxiv.org/abs/2107.05939

[13] M. Soni and B. S. Rajput, "Security and performance evaluations of quic protocol," in *Data Science and Intelligent Applications*, K. Kotecha, V. Piuri, H. N. Shah, and R. Patel, Eds. Singapore: Springer Singapore, 2021, pp. 457–462.

[14] K. Elmenhorst, B. Schütz, N. Aschenbruck, and S. Basso, "Web censorship measurements of http/3 over quic," in *Proceedings of the 21st ACM Internet Measurement Conference*, ser. IMC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 276–282.

[15] J. Topf, "The HTML Form Protocol Attack," Aug. 2001. [Online]. Available: https://www.jochentopf.com/hfpa/hfpa.pdf

[16] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 75–88.

[17] N. Jovanovic, E. Kirda, and C. Kruegel, "Preventing cross site request forgery attacks," in *2006 Securecomm and Workshops*, 2006, pp. 1–10.

[18] T. Prynn, "Cross-protocol request forgery," *NCC Group Whitepaper*, Oct. 2018.

[19] M. Petit-Huguenin and G. Salgueiro, "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)," RFC 7983, Sep. 2016.

[20] B. R. Sanjay and S. D. Pushparaj, "Dns amplification amp; dns tunneling attacks simulation, detection and mitigation approaches," in *2020 International Conference on Inventive Computation Technologies (ICICT)*, Feb. 2020, pp. 230–236.

[21] P. Mockapetris, "Domain names - implementation and specification," RFC 1035, Nov. 1987.

[22] Y. Gbur, "QUICforge." [Online]. Available: https://github.com/yurigbur/QUICforge

[23] J. Holmgren and A. Lombardi, "Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications)," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, Oct. 2018, pp. 850–858.

[24] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, 2011.

[25] L.-S. Huang, E. Y. Chen, A. Barth, E. Rescorla, and C. Jackson, "Talking to yourself for fun and profit," *Proceedings of W2SP*, pp. 1–11, 2011.

[26] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," *Network and Distributed System Security (NDSS)*, Feb. 2014.

[27] QUIC Working Group, "Implementations," Aug. 2021. [Online]. Available: https://github.com/quicwg/base-drafts/wiki/Implementations

[28] Y. Gbur, "YuKonSec Dockerhub." [Online]. Available: https://hub.docker.com/u/yukonsec

[29] M. Thomson, "Request forgery attacks - Issue 3995," Aug. 2020. [Online]. Available: https://github.com/quicwg/base-drafts/issues/3995

[30] W. May and P. Pritzker, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Aug. 2015.

[31] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018.

[32] Y. Sheffer, R. Holz, and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)," RFC 7457, Feb. 2015.

[33] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt, "DROWN: Breaking TLS using sslv2," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 689–706.

[34] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, "A first look at quic in the wild," in *Passive and Active Measurement*, R. Beverly, G. Smaragdakis, and A. Feldmann, Eds. Cham: Springer International Publishing, 2018, pp. 255–268.

[35] M. Schwarz, "One Protocol, Different Versions: Determining QUIC implementations Through Fingerprinting," May 2021. [Online]. Available: https://docs.google.com/presentation/d/1ZCsEE7zpPgku-tICPEo53g-i1xrUW8P0Av4XcVJkNrM/edit#slide=id.p

[36] J.-P. Smith, P. Mittal, and A. Perrig, "Website fingerprinting in the age of quic," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, pp. 48–69, 04 2021.

[37] C. Pearce and C. Vincent, "HTTP/2 & QUIC - Teaching Good Protocols to do Bad Things," 2016.

[38] 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.

[39] X. Cao, S. Zhao, and Y. Zhang, "0-rtt attack and defense of quic protocol," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019, pp. 1–6.

[40] M. Kühlewind, T. Bühler, B. Trammell, S. Neuhaus, R. Müntener, and G. Fairhurst, "A path layer for the internet: Enabling network operations on encrypted protocols," in *2017 13th International Conference on Network and Service Management (CNSM)*, 2017, pp. 1–9.