Which Secure Transport Protocol for a Reliable HTTP/2-based Web Service : TLS or QUIC ?

Antoine Saverimoutou, Bertrand Mathieu, Sandrine Vaton

Which Secure Transport Protocol for a Reliable HTTP/2-based Web Service : TLS or QUIC ?

Antoine Saverimoutou, Bertrand Mathieu
Orange Labs
Lannion, France
Email: {antoine.saverimoutou,bertrand2.mathieu}@orange.com

Sandrine Vaton
Institut Mines Telecom Atlantique
Brest, France
Email: sandrine.vaton@imt-atlantique.fr

***Abstract*—Web browsing protocols are currently gaining the interest of the researchers. Indeed, HTTP/2, an improvement of HTTP/1.1 has been standardized in 2015 and meanwhile, Google proposed another transport protocol, QUIC (Quick UDP Internet Connection). The main objective of the two protocols is to improve end-users quality of experience and communications security. Current HTTP/2-based web servers rely on the standardized TLS (Transport Layer Security) protocol, on top of TCP. Google has developed its own security system, natively integrated within QUIC, and runs on top of UDP. If performance issues, comparing HTTP/2 over TLS/TCP and QUIC/UDP, have been investigated by few researchers, no one studied the security aspects of the two transport protocols. This paper aims at filling this gap and proposes a first security analysis of TLS/TCP and QUIC/UDP. Based on their characteristics, this paper identifies the vulnerabilities of the two protocols and evaluates their impacts on HTTP/2-based web services. This study can enable web servers developers or administrators to either select TLS/TCP or QUIC/UDP.**

Keywords*— *Security, Encryption, TLS, QUIC, HTTP/2

I. Introduction

Web browsing is the main Internet service. It is done with HTTP (Hypertext Transfer Protocol) and HTTP/2, standardized in February 2015 at the IETF [1] [2], increasingly implemented by web servers, and now implemented in all browsers. Even though encrypted and secure connections are not mandatory in HTTP/2 RFCs, it is strongly recommended and it is the default (and imposed) behaviour of major web browsers (Chrome, Firefox and Internet Explorer). Therefore, security is a de facto standard in HTTP/2 communications. To meet this need, HTTP/2 relies on the TLS (Transport Layer Security) protocol, on top of TCP.

Since 2013, Google is investigating another way of delivering web contents and has proposed the QUIC (Quick UDP Internet Connections) protocol [3]. QUIC is expected to be faster and more reliable. QUIC runs on top of UDP and implements its own encryption system. QUIC cryptographic system is different from TLS and can establish connections into only 0 or 1-RTT (Round Trip Time). This is achieved thanks to its built-in encryption system and being UDP-based, instead of TCP, the need for the initial TCP handshake mechanism is removed.

Even if QUIC/UDP is currently used only by Google servers and Chrome/Chromium browsers, it represents an increasing part of the network traffic (many people use Google services). It is thus of prime importance to evaluate the protocol and identify if other web providers could have an interest to move to HTTP/2 over QUIC [4]. The interest can be driven by two aspects : performance and security. Some research studies such as [5] [6] compare the two protocols for a HTTP/2 web server in terms of performance (page load time, object size, server distribution, etc.). But to the best of our knowledge, there is no paper aiming to compare HTTP/2 on top of TLS/TCP and QUIC/UDP focused at security. This is the goal of this paper. We aim at identifying the possible vulnerabilities of both TLS/TCP and QUIC/UDP, evaluating their impacts on HTTP/2-based services. Our main goal is to give hints to web server developers or administrators on whether they should provide their HTTP/2 web service with TLS/TCP or QUIC/UDP.

This paper is structured as follows : We first remind in Section II the main characteristics of TLS/TCP and QUIC/UDP in terms of key exchanges and connection establishment. Then, we list and expose the identified vulnerabilities of the two protocols and their impacts with regards to the offered services in Section III. Finally, after presenting some works in relation to secure network protocols in Section IV, we conclude this paper in Section V.

II. TLS & QUIC security aspects

In this section, we first remind the main security aspects of the TLS and QUIC protocols, mainly in terms of key exchanges and connection establishment.

A. Overview of HTTP/2 over TLS/TCP security

HTTP/2 over TLS/TCP uses the current version 1.2 of TLS [7] which is meant to provide cryptographic security, interoperability, extensibility and relative efficiency on two different levels: the *TLS Record protocol* and the *TLS Handshake protocol*.

TLS Record Protocol: This protocol negotiates a private, reliable connection between the client and the server, where symmetric cryptography keys are used to ensure a private connection. The connection is secured through the use of hash functions generated by using a Message Authentication Code (MAC).

TLS Negotiation Protocol: The protocol allows authenticated communication to start between the client and the

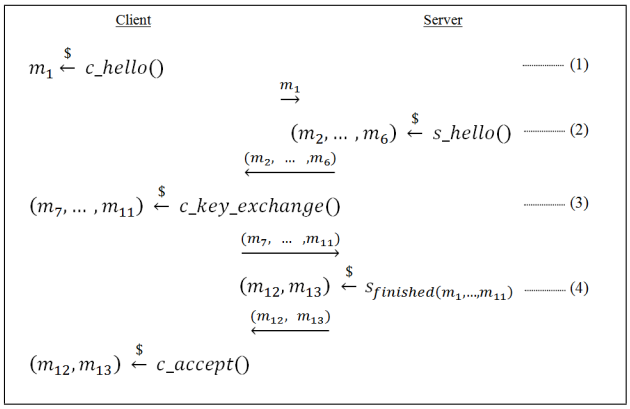

Fig. 1. TLS 1.2 negotiation (Ephemeral Diffie-Hellman)

server. The handshake uses asymmetric encryption, where two separate keys are used, a public key used for encryption and a private key for decryption, to produce a newly-created shared-key. The session then uses this freshly produced shared-key to perform a symmetric encryption, which yields to a feasible secure connection.

Two types of handshakes are available in TLS 1.2, *Static RSA* and *Ephemeral Diffie-Hellman*. We hereunder give a description of the main messages (out of 13 messages) sent during the TLS negotiation with Ephemeral Diffie-Hellman (DH) key exchange and client authentication as per Figure 1. The TLS negotiation is made in 2-RTT (full TLS negotiation for a *first-time* connection) or in 1-RTT (abbreviated TLS negotiation for a *repeat* connection).

1) Client hello: The Client, C, sends a c_hello message (Fig 1-1), m_1, to the server, S, in order to identify itself where

$$c\_hello = (sslv_c, r_c, s_{id}, cs\_list_c, ca\_list_c, ext_c)$$

- $sslv_c$ is the version of Secure Socket Layer (SSL) [8] the client, C, tries to use for negotiating with the server, S.
- r_c is the random bytes generated by C to produce a master key for encryption.
- s_{id} is a *session id* used during a *repeat* connection,
- cs_list_c is a list of cipher suites, containing all encryption algorithms that the browser on client side is willing to support,
- ca_list_c is a list of compression algorithms supported by the client,
- ext_c optional, consists of a list of extensions that can be used to improve the security of the negotiation.

2) Server hello and Server Key Exchange : The server, S, responds with a s_hello message (Fig 1-2), where:

- m_2 has the same structure as m_1 except that at most one cipher suite and one compression method is present,
- m_3 may contain a chain of certificates, starting from the TLS server certificate up to a direct child of a root certificate,
- m_4 is the *Server Key Exchange* message containing the Diffie-Hellman key exchange parameters,
- m_5 is the Certificate Request message, containing a list of certificate types that C can use,
- m_6 consists of a constant tag with byte value 14 and a length value 0.

3) Client key exchange and client finished : Having received messages $(m_2, ..., m_6)$, the signature is verified by C. If verification fails, C rejects and aborts. Otherwise, C is able to complete the key exchange and compute the cryptographic keys (Fig 1-3) through $(m_7, ..., m_{11})$ where

- m_7 is the Client Certificate and contains a signing certificate $cert_c$ with the public key pk_c,
- m_8 is the Client Key Exchange and contains the Diffie-Hellman share, T_c,
- m_9 is the Certificate Verify which is used to authenticate the client C,

The client then computes the *master secret*, ms, stored for the lifetime of the TLS session. The *Client Finished* message, fin_c, is then derived by using the ms together with two random nonces. The keys are afterwards handed to the *TLS Record Layer* which ensures a private connection between the client C and the server S through the use of hash functions where

- m_{10} is the *Change Cipher Spec*, signaling the start of encryption being sent to S,
- m_{11} consists of an authenticated encryption of the fin_c message.

4) Server Finished : The server verifies the signature received in m_9. If verification succeeds, S calculates its *master secret*, ms, and computes the encryption and MAC keys. S can then decrypt m_{11}, and verify fin_c. If verification succeeds, the server computes the *Server Finished message*, fin_s over all plaintext messages. Messages (m_{12}, m_{13}) are then sent to C (Fig 1-4) where

- m_{12} is the encoded flag, $flag_{enc}$,
- m_{13} is the encryption of fin_s.

The client, C verifies the fin_s received from S (by using its pre-computed encryption and MAC keys obtained before calculating m_{10} and m_{11}).

Encrypted Payload Transmission: The keys obtained can now be used in order to transmit payload data in the *TLS Record Layer* by using a stateful length-hiding symmetric encryption scheme [9].

If the client C has recently connected to the server S and the certificates are still valid, C uses the session id, s_{id}, stored in its cache in order to make an abbreviated TLS negotiation. As per Figure 1, C goes through only steps *1* and *4*.

B. Overview of HTTP/2 over QUIC/UDP security

The QUIC-Crypto protocol[1] is part of QUIC that provides transport security to a connection, where two session keys are

[1]https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit

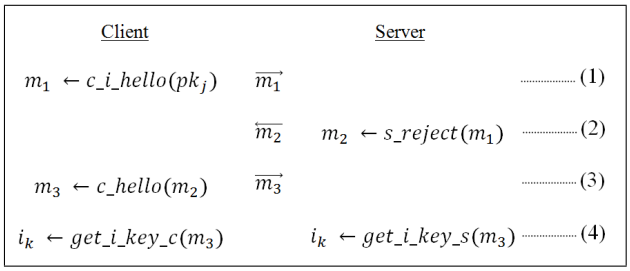

Fig. 2. Initial Key Agreement phase of QUIC

used. To meet the 1-RTT, the parties first agree on an initial session key during the *Initial Key Agreement* phase, which can be used to exchange data until the final exchange key is set, detailed as follows:

- **Initial Key Agreement:** Each party sets its *initial key material*, i_k, which is used for encryption and decryption,
- **Initial Data Exchange:** Client (C) and Server (S) exchange their initial data, encrypted and authenticated,
- **Key Agreement:** Consists of one message. S generates a new Diffie-Hellman (DH) value and sends its public DH value to C. C verifies authenticity of the server's new DH public value and both parties at this point derive the session key material, s_k,
- **Data Exchange:** Consists of two packets. C and S use the session key material, s_k, to encrypt and authenticate their remaining data.

We describe hereunder the *Initial Key Agreement* phase as per Figure 2 in detail since it is where security flaws appear because at the very start of that process, the headers are not encrypted between the client and the server and an attacker, acting as a MiTM (Man-in-The-Middle) may have access to that exchanged information on the communication channel. An overview of the three other phases can be found in [10].

1) Client connection initiation phase: The client sends the message, m_1 (Fig 2-1), to the server he wants to connect to, where

$$m_1 = (IP_c, IP_s, port_c, port_s, cid, 1)$$

- $cid \xleftarrow{\$} \{0,1\}^{64}$, a random generated id by the client C,

2) Server $s\_reject(m_1)$ response: The server, S, responds to the client with the message m_2 (Fig 2-2) by running $s\_reject(m_1)$ where

$$m_2 = (IP_c, IP_s, port_c, port_s, cid, 1, scfg^t_{pub}, stk)$$

- stk, is an authenticated-encryption block of the client's IP and a timestamp, to be used later by C to identify itself to S during the initial key agreement phase as well as during additional 0-RTT connection requests,
- $scfg^t_{pub}$ contains the server's DH public values with an expiration date and a signature $prof$ over all the public values under the server's secret key s_k.

At this very stage the 1-RTT connection establishment has been realized. Initial keys can be derived and the parties can start exchanging data until the final key is set.

3) Client verifies m_2: On receiving m_2, C checks that $scfg^t_{pub}$ is authentic and not expired since it possesses the public key of S. C then generates a nonce and its own DH values (Fig 2-3) by running $c\_hello(m_2)$, which is sent to S through m_3, where

$$m_3 = (pkt\_info, cid, 2, stk, scid, nonc, pub_c)$$

4) Initial Key Material derivation: Both C and S derive their initial key material, i_k, which will be used during the *Initial Data Exchange* and *Key Agreement* phase.

If client C already had a connection with server S, in the time period τ_t, then C does not need to send a c_i_hello but can instead initiate another connection request with S via a c_hello packet. Upon reception, S verifies that the nonce is fresh, the stk valid and $scid$ known and not expired, leading to 0-RTT. If these conditions are not met, S goes to the 1-RTT connection establishment phase.

III. Identification of Vulnerabilities and Impact on HTTP/2 web-servers over TLS/TCP & QUIC/UDP

This section lists the identified vulnerabilities of the two protocols, when being used with HTTP/2, based on an active or passive attack model, where the threats are classified as interception, interruption, modification and fabrication. For each identified attack, we first point out the attack model and threat, describe the protocol vulnerability which allows the attack to be feasible as well as how complex it is to be set up and what are the drawbacks regarding the end-user's QoE (Quality of Experience) and QoS (Quality of Service) provided by the service. We take into account attacks for which no immediate solutions exist.

A. Impacts of TLS/TCP vulnerabilities on service

1) Slow read attacks: This slow-http attack is part of active attack models, where interruption leads to DDoS (Distributed Denial of Service) and is defined as being the Slowloris DDoS [11] coupled to Slow POST method, where the main objective is to bypass policies that filter slow-deciding clients, by reading HTTP responses slowly and at the same time open multiple active connections with a targeted server by sending partial HTTP requests. Through SYN packets, an attacker deliberately advertises a small receive window. On request received, the server generates the response which is sent to its socket and the server's kernel further delivers the data to the attacker. The server's sending rate being throttled, the attacker is asked to increase its receive-window but no action is fulfilled. Automatic mechanisms being absent on a TLS/TCP server to auto-tune flow control credits for both stream and connection flow controllers to adapt the receive-window, the responses are delivered slowly to the attacker. Concurrently, with the help of Slowloris DDoS, multiple partial requests sent by attacker make the server to over-poll its socket for write readiness which leads to a drastic increase in the server's resources, inducing a Denial of Service.
The attack's implementation is straightforward and traditional detection systems are powerless since they allow the partial

packets (which are not malformed packets) to go through their security policy. If unmitigated or undetected, this attack can last for long periods of time. During the attack laps time, the QoS offered by the website is downgraded and the legitimate client's QoE is impacted due to an unresponsive website.

2) Logjam: Logjam [12] is part of active attack models, where modification through a MiTM challenges asymmetric ciphers and relies on a flaw in the way TLS composes DHE and DHE-EXPORT. To comply with the 1990s-era U.S export restrictions on cryptography, many TLS servers are still configured with two groups, i.e. a strong 1024-bit group for regular Diffie-Hellman Ephemeral (DHE) key exchanges and a 512-bit group for legacy DHE-export. When the server selects DHE-EXPORT for a handshake, it proceeds by issuing a signed *Server Key Exchange* message. Provided the client also offers DHE, an attacker acting as MiTM can re-write the $client\_hello$ message, m_1 (Fig 1-1) which is $g^a mod\ p$ to offer a corresponding DHE-EXPORT ciphersuite, among the different cipher suites present in cs_list_c. A spoofed s_hello message $(m_2, ..., m_6)_{spoofed}$ (Fig 1-2) which is $g^b mod\ p$ can also be written by the attacker to replace the choosen DHE-EXPORT ciphersuite. On receipt, the legitimate client interprets the $s\_hello_{spoofed}$ message. At this stage both client and server have different handshake transcripts, and if the attacker has important computing power resources, he can calculate b in close real time (i.e. $\log b$ from $g^b mod\ p$), and can derive the master secret, ms (based on the shared keys $g^{ab} mod\ p$ exposed in Fig1-3) as well as the connection keys to complete the handshake with the client and hence read and write freely application data, pretending to be the server.
Since vulnerable TLS connections can be downgraded to 512-bit export-grade cryptography, both the client and the server can be impacted by this attack, where the attacker can read and modify any data passed over the connection. As long as no emphasis on downgrade protection is brought in TLS, any client connecting to a web service (where DHE_EXPORT ciphers are supported) can have his session highjacked, where access to the desired service is prevented and bears risks of having his personal information collected and eventually forged. The server's QoS on the other side is decreased as it can serve less legitimate clients and the latters experience a downgraded QoE through an unresponsive webpage.

3) Lucky Thirteen attack: The Lucky 13 attack targets ciphersuites which use Cipher Block Chaining (CBC) mode encryption, and first goes through a passive attack model where an attacker acting as MiTM intercepts in clear text the TLS handshake message (e.g on an unsecured WiFi network) between a legitimate client and the server, followed by an active attack model where interruption is carried out through modified ciphertext injection (m_2 in Fig 1-2 is modified). The attack succeeds on chosen message lengths and when the HMAC-SHA1 MAC algorithm is used, where there is an alignment of the TLS headers, plaintexts and MAC tag bytes with the blocks ciphers boundary and the hash compression function's block boundary. TLS messages containing at least 2 bytes of correct padding will be processed slightly faster than those containing 1 byte of correct padding. By repeating the attack and use of adequate statistical processing, noise from the network jitter is isolated and padding conditions differentiated.
Through this attack's success plaintext of authentication cookies are retrieved and through replay attacks, the attacker gains access to personal information, or the authentication cookie can itself be forged (e.g by modifying the *session id*, s_{id} (Fig 1-1) used for *repeat* connection). The client trying to connect to a particular web service with the forged cookie although having a valid timestamp will be prevented from authentication, thus downgrading his QoE. Authenticated data being able to be recovered through this attack, any exchange made between the server and legitimate clients can be collected by third parties, thus decreasing the degree of trust of the web service itself.

4) Broken RC4 as primary ciphersuite: Rivest Cipher 4 (RC4) is a fast stream cipher for SSL/TLS connections where it does not need padding, being immune to TLS attacks like BEAST[2] and Lucky Thirteen, being pretty fast and yielding less computations or hardware requirements. But if a plaintext is encrypted with many different RC4 keys, an attacker acting as a MiTM can intercept the plaintext and a bunch of totally random looking ciphertexts can be obtained. An attacker having important computing power and making proper statistical analysis of different portions of the ciphertexts will obtain frequent appearing values. By getting several different encryptions of the same message, under different keys, these small deviations from random can lead the attacker to what was originally encrypted.
When the client's connection is encrypted with RC4, each time he makes a fresh connection to a web service, he automatically sends a new encrypted copy of the same cookie. Furthermore, if the session is renegotiated by using a different key, the attacker builds up the list of ciphertexts needed. Having access to this information, the attacker can perform actions under the victim's name and gain access to personal information. On the long run, the QoE of the user will be downgraded since his personal information can be used to perform unappropriate actions on his behalf.

B. Impacts of QUIC/UDP vulnerabilities on service

Most of QUIC packet headers and payloads are encrypted and authenticated, except during the connection establishment phase and more particularly at the *Initial Key Agreement* phase (Figure 2). Considering active attack models where interruption, fabrication and modification occurs, we take into account an adversary who acts as MiTM and listens to the communication channel, being closer to the client. During the hanshake process, the attacker can learn the server's current state, $scfg^t_{pub}$, as well as the source-address token, stk. By replaying these two spoofed-forged values, i.e. the server's $scfg^t_{pub}$ to the client and the client's stk to the server, the

[2]http://resources.infosecinstitute.com/ssl-attacks/#gref

		Impact on Client	Impact on Server
TLS/TCP	*Slow Read attack*	QoE decreased	Unresponsive to legitimate requests
	Logjam	Session hijacked, personal information collected and can be forged	Service access and popularity reduced
	Lucky Thirteen	No access to web service	Web service trust degree impacted
	Rivest Cipher 4	Forgery of personal information	Web service trust degree decreased
QUIC/UDP	*Replay attack*	Connection establishment failure	No impact as server ends connection
	Packet Manipulation	Connection establishment failure	No impact as connection ends
	Crypto Stream Offset	Deprived from web service access and fall back to TLS/TCP	No impact, broken byte-stream ends connection

TABLE I
VULNERABILITIES & IMPACT ON HTTP/2-BASED WEB SERVERS OVER TLS/TCP & QUIC/UDP

core particularity of QUIC putting forth at most 1-RTT can turn to be a roadmap leading to the protocol's inefficiency.

1) Replay attacks: Replay attacks are part of active attack models where fabrication occurs when an adversary alters the information between the client and server in the connection establishment phase, acting as a MiTM, being geographically close to the client. After having obtained the $scfg^t_{pub}$ and stk, the adversary waits for the client to connect. When the client sends a c_i_hello message (Fig 2-1) to the server, the adversary sends back a $spoofed\_s\_reject$ message (Fig 2-2), $\bar{m_2}$, to the client (before the server responds) with a $stk_{spoofed}$. On receipt, the client verifies that $scfg^t_{pub}$ is not expired. The client then generates a nonce and its own Diffie-Hellman (DH) value, with the $stk_{spoofed}$ value, which is sent to the server. The server receives the c_i_hello message, but since the stk is not legit, the validation fails and the server sends a $legit\_s\_reject$ message to the client, so that the latter can re-calculate its c_i_hello message. At this very phase, the client moves backwards in the negotiation process to re-calculate the nonce and own DH value.
Through replay attacks, an adversary can prevent a client from performing the connection establishment phase with the server. To counter optimistic ACK attacks, each QUIC packet includes an entropy bit in its header (being UDP-based), together with a hash of these bits in the QUIC ACK frames. When the client acknowledges the $spoofed\_s\_reject$ message to the legit server, the entropy bit does not match the server's entropy bit in its legit response, which prevents the entropy hash in this ACK to validate and connection with the server will be ended.
Furthermore, the QUIC-Crypto handshake takes place within a special byte-stream reserved for connection establishment. All c_hello, s_reject and s_hello messages (exposed in Figure 2) occur within the context of this byte-stream having offset and length attributes. Since the attacker's s_reject message size has a very high probability at being different from the server's response, the byte-stream is effectively broken. Messages being at offsets will be dropped or buffered forever, and after a delay of ten seconds, the client's connection will be ended, thus having a direct impact on the client's QoE.

2) Packet Manipulation attacks: Non-encrypted QUIC packets are not protected against adversarial manipulation. If an attacker has access to the communication channel, acting as MiTM, bits of unprotected parameters such as the connection id, cid, or source address token, stk, can be flipped through fabrication and lead the client and server to derive different initial keys which would ultimately lead the connection establishment to fail.
The server automatically proposes the client to re-negotiate the connection establishment, where the low RTT put forth by the protocol is doomed, since the adversary can one more time proceed to packet manipulation and lead to an infinite connection establishment loop for the client. The client's QoE is drastically reduced until the webpage abandonment.

3) Crypto Stream Offset: All handshake messages are part of a logical byte-stream in QUIC. If an adversary acting as MiTM has access to the communication channel and can inject random data to interrupt the byte-stream, the attacker is able to break the byte-stream itself and prevent further connection establishment. When further processing of the handshake messages happen, the legitimate user is found denied of access to the desired web service and case-wise forced to fall back to TLS/TCP leading with a higher RTT than expected. Through both of these scenarios, even if the server's QoS is not impacted, the client's QoE is reduced.

C. Comparison of TLS/TCP & QUIC/UDP vulnerabilities on service

Through Table I, a client making use of a web service being HTTP/2 based on TLS/TCP has its QoE decreased and personal information can be collected and forged. The server is impacted upon its responsiveness, leading to a decrease in number of clients it can serve and may crash.
For a client making use of a web service being HTTP/2 based on QUIC/UDP, although his connection establishment is impacted and may even fall back to TCP/TLS which shrinks his QoE, the early connection is safe since no sensitive information is exchanged until connection establishment succeeds. On the server side, connection of the client is ended if any verification step fails, making the server still fully available and safe to legitimate visitors, ensuring promised QoE.

IV. Related Work

Some research papers address specific security issues of the TLS protocol, namely use of SSL to break TLS [13], cross protocol attacks on TLS [14] and key-encapsulation mechanism extraction from the TLS handshake protocol [15]. We can find very few presenting the QUIC security aspects, [16] for multi key exchanges or [10] for a cryptographic analysis. But all of those papers are different than ours, since they address specific points. Regarding papers studying security aspects as a whole, we can mention [17] and [18], which explore the security aspects, the protocols and different types of attacks and countermeasures. But these papers focus on the Internet of Things domain and not the Web. The network protocols are different: MQ Telemetry Transport (MQTT) and the Constrained Application Protocol (CoAP), and not TLS or QUIC. In [19], the authors present the main challenges for Big Data environments regarding security and privacy, but encryption and secure communications is only one item, and only mentions TLS, not QUIC. With the same use-case as us, we can cite [20] which addresses the websites protection, but the paper only presents the main attacks and possible actions to take into consideration to avoid attacks, and do not investigate the network transport protocols such as TLS or QUIC. In [21], the author presents an analysis of the vulnerabilities of TLS, but without relationship upon their impacts on the web server and QUIC is not addressed. To the best of our knowledge, our paper is the first one aiming at comparing the two protocols at the security level and evaluating the impacts for a HTTP/2-based web service.

V. Conclusion

In this paper, we have highlighted the main vulnerabilities of two secure transport protocols, the well-known and largely used TLS/TCP and the newcomer, proposed by Google, QUIC/UDP. This analysis is done mainly for a web service, using HTTP/2 on top of these two secure protocols. Our analysis shows that QUIC/UDP allows to have a more reliable server, since the server can not crash or reach a state where the web service offered to end-users is largely degraded. Although the QUIC/UDP attacks mainly impact end-users, the impact on the server is limited, whereas TLS/TCP attacks target more precisely the server, with the aim to make a DDOS. With TLS and the current QUIC specification, web designers should prefer to deliver their contents using QUIC/UDP rather than TLS/TCP regarding security issues.

Acknowledgement

This work is partially funded by the French National Research Agency (ANR) BottleNet project, No ANR-15-CE25-0013-001.

References

[1] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," Internet Requests for Comments, RFC Editor, RFC 7540, May 2015, http://www.rfc-editor.org/rfc/rfc7540.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7540.txt

[2] R. Peon and H. Ruellan, "HPACK: Header Compression for HTTP/2," Internet Requests for Comments, RFC Editor, RFC 7541, May 2015.

[3] J. Iyengar, I. Swett, R. Hamilton, and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2," Internet Engineering Task Force, Tech. Rep. draft-tsvwg-quic-protocol-02, Jan. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02

[4] R. Shade and M. Warres, "HTTP/2 Semantics Using The QUIC Transport Protocol," IETF Draft, IETF, Draft, July 2016. [Online]. Available: https://datatracker.ietf.org/doc/draft-shade-quic-http2-mapping/

[5] P. Biswal and O. Gnawali, "Does quic make the web faster?" in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

[6] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui, "QUIC: Better For What And For Whom?" in *Proceedings of IEEE International Conference on Communications (ICC)*, May 2017.

[7] T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Tech. Rep. 5246, Aug. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5246.txt

[8] A. O. Freier, P. Karlton, and P. C. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0," RFC 6101, Tech. Rep. 6101, Aug. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6101.txt

[9] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of tls-dhe in the standard model," in *Advances in Cryptology – CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, R. Safavi-Naini and R. Canetti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 273–293.

[10] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How Secure and Quick is QUIC? Provable Security and Performance Analyses," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, May 2015.

[11] E. Damon, J. Dale, E. Laron, J. Mache, N. Land, and R. Weiss, "Hands-on denial of service lab exercises using slowloris and rudy," in *Proceedings of the 2012 Information Security Curriculum Development Conference*, ser. InfoSecCD '12. New York, NY, USA: ACM, 2012, pp. 21–29.

[12] W. Bokslag, "The problem of popular primes: Logjam," vol. abs/1602.02396, 2016.

[13] 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, 2016, pp. 689–706.

[14] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel, "A cross-protocol attack on the tls protocol," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 62–72. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382206

[15] H. Krawczyk, K. G. Paterson, and H. Wee, *On the Security of the TLS Protocol: A Systematic Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 429–448.

[16] M. Fischlin and F. Günther, "Multi-stage key exchange and the case of google's quic protocol," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 1193–1204.

[17] I. Andrea, C. Chrysostomou, and G. Hadjichristofi, "Internet of things: Security vulnerabilities and challenges," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, July 2015, pp. 180–187.

[18] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the internet of things: A survey of existing protocols and open research issues," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1294–1312, thirdquarter 2015.

[19] Y. Gahi, M. Guennoun, and H. T. Mouftah, "Big data analytics: Security and privacy challenges," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, June 2016, pp. 952–957.

[20] D. Gillman, Y. Lin, B. Maggs, and R. K. Sitaraman, "Protecting websites from attack with secure delivery networks," *Computer*, vol. 48, no. 4, pp. 26–34, Apr 2015.

[21] M. S. Haque, "Web server vulnerability analysis in the context of transport layer security (tls)," *IJCSI International Journal of Computer Science Issues*, vol. 13, no. 5, pp. 11–19, 2016.