Workgroup: Network Working Group
Internet-Draft: draft-joseph-tls-turbotls-00
Published: 5 November 2023
Intended Status: Informational
Expires: 8 May 2024
Authors: D. Stebila, University of Waterloo
D. Joseph, SandboxAQ
C. Aguilar-Melchor, SandboxAQ
J. Goertzen, SandboxAQ

TurboTLS for faster connection establishment

Abstract

This document provides a high level protocol description for handshaking over UDP in the Transport Layer Security (TLS) protocol (version independent). In parallel, a TCP session is established, and once this is done, the TLS session reverts to TCP. In the event that the UDP handshaking portion fails, TurboTLS falls back to TLS-over-TCP as is usually done, resulting in negligible latency cost in the case of failure.

Discussion of this work is encouraged to happen on the TLS IETF mailing list tls@ietf.org or on the GitHub repository which contains the draft: https://github.com/PhDJsandboxaq/draft-ietf-turbotls-design/.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 May 2024.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction
   1.1. Terminology
   1.2. Motivation for handshaking over UDP
   1.3. Scope
   1.4. Goals
2. Transport Layer Security
3. Construction for TLS
   3.1. Protocol diagram TLS
   3.2. Protocol diagram TurboTLS
   3.3. Fragmentation
      3.3.1. Client request-based fragmentation
   3.4. TLS-over-TCP fallback
      3.4.1. Early data, post-handshake messages, and TCP fallback
   3.5. TurboTLS support advertisment
   3.6. Specification: Handshake embedding into UDP
   3.7. Specification: UDP Tombstone
4. Security Considerations
   4.1. Transparent proxying
   4.2. Denial-of-Service
      4.2.1. Attacks *on* TurboTLS servers
      4.2.2. Attacks *leveraging* TurboTLS servers
5. References
   5.1. Normative References
   5.2. Informative References
Authors' Addresses

1. Introduction

This document gives a construction for TurboTLS [ABGGJS23], which at its core is a method for handshaking over UDP in TLS before switching back to TCP for the TLS session. A technique called client request-based fragmentation is described which reduces the susceptibility of TurboTLS servers to being exploited by UDP reflection attacks.

reduce the possibility of portions of the handshake over UDP being filtered by poorly configured middle-boxes, and a fallback procedure to standard TLS-over-TCP (at minimal latency overhead) is provided.

1.1. Terminology

* **UDP** Universal Datagram Protocol: a connectionless transport protocol, whereby packets are sent, but without any codified way of knowing that such packets have been successfully received. This leads to low reliability but can be appropriate where applications are time sensitive.

* **TCP** Transmission Control Protocol: a connection-oriented protocol that ensures the successful delivery of packets. Before a communication over TCP can start in earnest, a connection must be established. This is done via a TCP handshake consisting of a SYN, a SYN ACK and an ACK.

* **TLS** Transport Layer Security: a cryptographic protocol that enables a client and server to authenticate one another, and communicate confidentially. TLS initializes with a handshake where cryptographic primitives are executed and session parameters are agreed upon, and then a session over which applications exchange encrypted communications.

* **QUIC** (not an acronym): a security protocol that embeds TLS functionality directly into UDP-based transport. Due to the drawbacks of UDP, QUIC implements its own reliability, packet reordering, and packet dropping procedures as well as the security properties.

1.2. Motivation for handshaking over UDP

TLS is the most ubiquitous application layer security protocol in use at the time of writing. Other protocols for secure connection establishment have been proposed, and one such widely-used protocol is QUIC. QUIC runs entirely over UDP and merges the transport and security aspects into one specification, handling packet loss, reordering, handshake establishment, and session management all in one protocol. One benefit of QUIC is that it enjoys fast connection establishment because it runs over UDP, whereas running on TCP would mean waiting for a TCP handshake to occur which requires one round trip.

Many will make the choice to move from TLS to QUIC, however some will not for a range of reasons. Deep packet inspection is inhibited in QUIC, and updating some legacy systems can be difficult. TurboTLS aims to provide a method for those who do not want to fully switch to QUIC, to benefit from the fast connection establishment enabled by UDP, but without fundamentally changing the security properties

of TLS, and furthermore enabling implementation via transparent proxying, thus avoiding the need to directly upgrade such systems themselves.

1.3. Scope

This document focuses on TurboTLS [ABGGJS23]. It covers everything needed to achieve the handshaking portion of a TLS connection over UDP, including

* **Construction in principle:** It provides an outline of which flows are sent over UDP, which are sent over TCP and in what order.

* **TLS-over-TCP fallback:** The document describes what to do in the case of failure due to UDP packet loss or filtering. The scheme should revert to TLS-over-TCP incurring a small latency overhead that should be minimal in comparison with standard TLS-over-TCP without a TurboTLS attempt.

* **Client request-based fragmentation:** Due to the impact of post-quantum cryptography such as larger keys certain considerations have to be taken into account. One is that a Server Hello is likely to require multiple UDP packets, thus to eliminate the possibility of reflection attacks and hence serve as a DDoS mitigation, we describe how to create a one-to-one correspondence between Client Hello packets and Server Hello packets.

* **How to implement via a transparent proxy:** The document gives a brief description of how one can implement TurboTLS via a transparent proxy, which has two implications. The first is that it demonstrates clearly that the security of TLS is unchanged, as a server and client can have their entire transcript intercepted by two proxies (one in front of each), which TurboTLS-ify the interaction. Thus the view server and client is unchanged versus standard TLS. The second is that the TLS proxy represents a way for legacy systems to benefit from faster connection establishment without requiring direct upgrades.

* **Performance considerations** Due to the parallelization of the UDP flow and TCP flows, as well as the TCP fallback mechanism, TurboTLS will have some impact on bandwidth requirements. We discuss these briefly, as well as the expected benefit from reducing a round trip when TurboTLS works and the small latency overhead when it doesn't and reverts to TLS-over-TCP.

It intentionally does not address:

* **Protocol design of TLS:** The internal workings and security mechanisms of TLS are not affected by TurboTLS, as can be seen

via the transparent proxying argument. This document does not discuss the design or merits of any version of TLS.

1.4. Goals

* **High performance:** Successful use of TurboTLS removes one round trip and should cut handshaking time by up to 50%. However in the worst case, when the fallback mechanism to TLS-over-TCP is used, there should be only a minimal impact on latency.

* **Ease of implementation:** TurboTLS should be designed such that it is possible to implement in many scenarios where other more invasive upgrades may not be possible, such as switching to QUIC. Transparent proxying should enable this via network proxies, sidecar proxies, or directly modifying the client/server applications.

* **Security:** The design should not create any opportunities for adversaries, either to attack TurboTLS servers or to use them e.g. during a reflection attack. The ratio of received:sent UDP packets, in particular, affects an adversary's chances of carrying out such reflection attacks. The handling of semi-open TCP connections is also important to consider in mitigating DoS attacks.

2. Transport Layer Security

The Transport Layer Security (TLS) protocol is ubiquitous and provides security services to many network applications. TLS runs over TCP. As shown in Figure 1, the main flow for TLS 1.3 connection establishment [TLS13] in a web browser is as follows.

First of all, the client makes a DNS query to convert the requested domain name into an IP address. Simultaneously, browsers request an HTTPS resource record [SBN22] from the DNS server which can provide additional information about the server's HTTPS configuration. Next, the client and server perform the TCP three-way handshake. Once the TCP handshake is complete and a TCP connection is established, the TLS handshake can start; it requires one round trip -- one client-to-server C->S flow and one server-to-client S->C flow -- before the client can start transmitting application data.

In total (not including the DNS resolution) this results in two round trips before the client can send the first byte of application data (the TCP handshake, plus the first C->S and S->C flows of the TLS handshake), and one further round trip before the client receives its first byte of response.

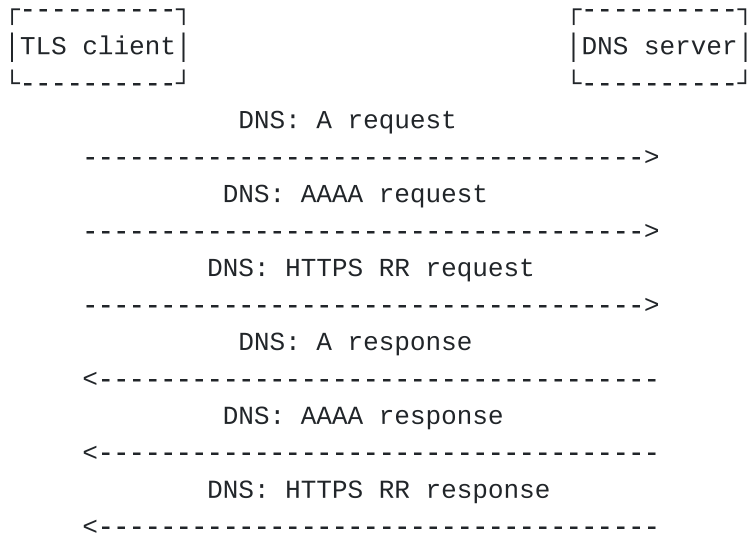
TLS does have a pre-shared key mode that allows for an abbreviated handshake permitting application data to be sent in the first C->S

TLS flow, but this requires that the client and server have a pre-shared key in advance, established either through some out-of-band mechanism or saved from a previous TLS connection for session resumption.

3. Construction for TLS

We first demonstrate protocol diagrams of the handshaking parts of TLS and TurboTLS.

3.1. Protocol diagram TLS



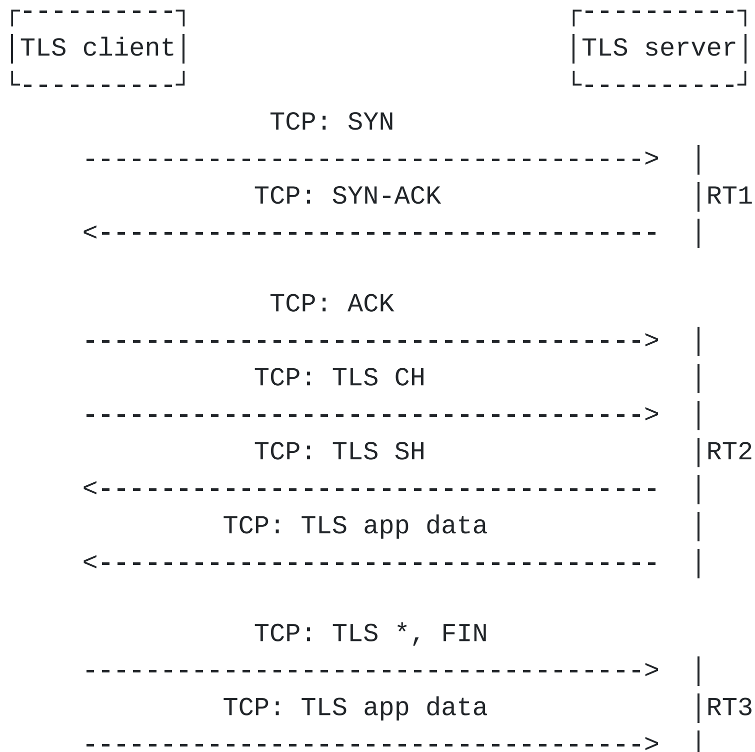
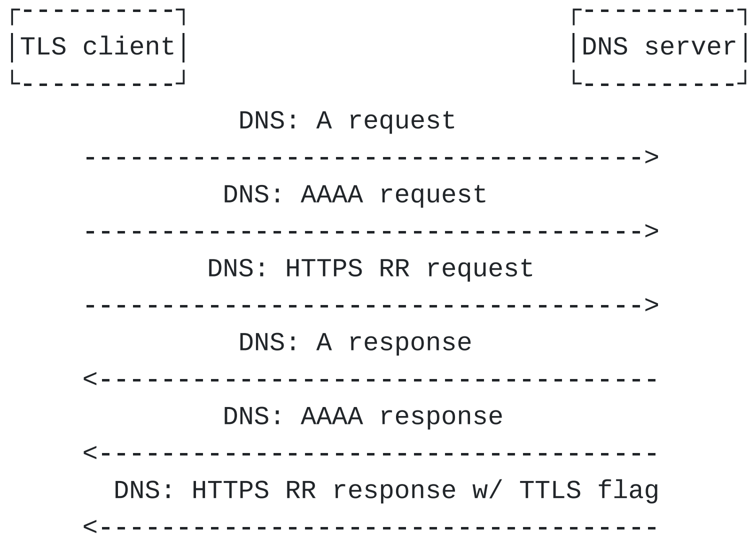


Figure 1: TLS-over-TCP handshake

3.2. Protocol diagram TurboTLS



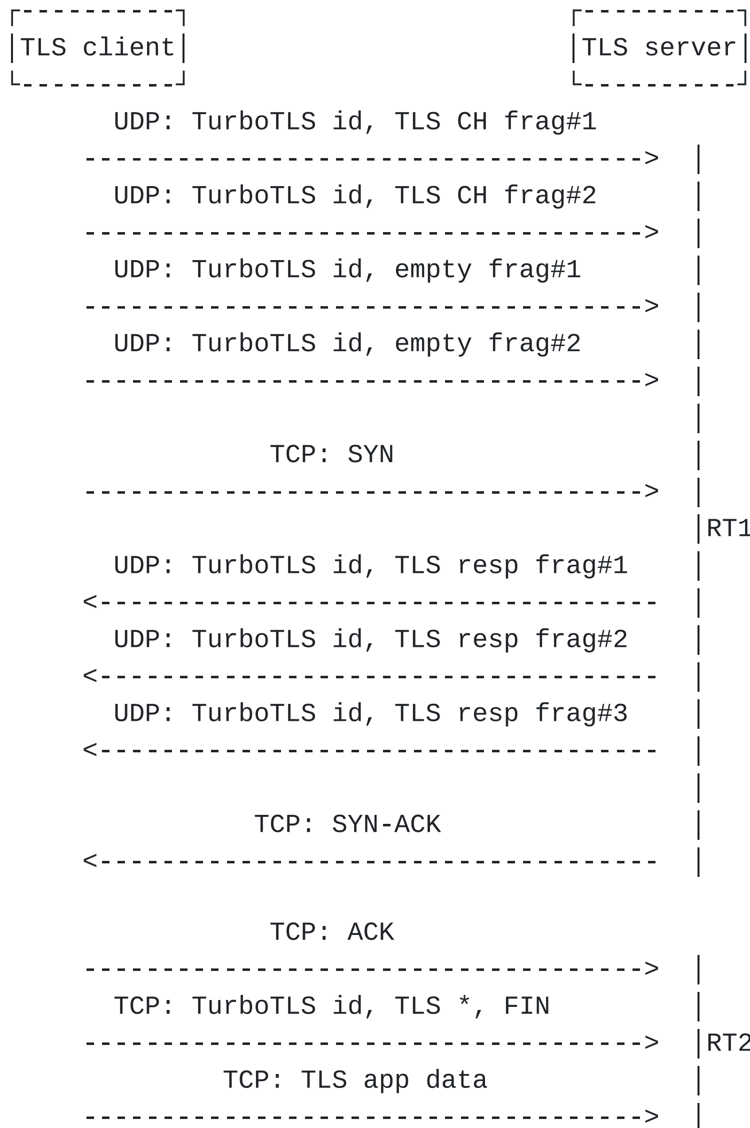


Figure 2: TurboTLS Handshake

As described in Figure 2, TurboTLS sends part of the TLS handshake over UDP, rather than TCP. Switching from TCP to UDP for handshake establishment means we cannot rely on TCP's features, namely connection-oriented, reliable, in-order delivery. However, since the rest of the connection will still run over TCP and only part of the handshake runs over UDP, we can reproduce the required functionality in a lightweight way without adding latency and allowing for a simple implementation.

3.3. Fragmentation

One of the major problems to deal with is that of fragmentation. TLS handshake messages can be too large to fit in a single packet -- especially with long certificate chains or if post-quantum algorithms are used.

Obviously the client can fragment its first C->S flow across multiple UDP packets. To allow a server to link fragments received across multiple UDP requests, we add a 12-byte session ID field, containing a client-selected random value *id* that is used across all TurboTLS fragments sent by the client. The session ID is also included in the first message on the established TLS connection as part of the tombstone message to allow the server to link together data received on the UDP and TCP connections. To allow the server to reassemble fragments if they arrive out-of-order. The client will send the last in-order UDP packet it received's sequence number as a part of the tombstone message so that the server can detect packetloss.

Similarly, the server can fragment its first S->C flow across multiple UDP packets. One additional problem here however is that the S->C flow is typically larger than the C->S flow (as it typically contains one or more certificates), so the server may have to send more UDP response packets than UDP request packets. As noted by [SW19] in the context of DNSSEC, many network devices do not behave well when receiving multiple UDP responses to a single UDP request, and may close the port after the first packet, dropping the request. Subsequent packets received at a closed port lead to ICMP failure alerts, which can be a nuisance.

3.3.1. Client request-based fragmentation

A method proposed by Goertzen and Stebila [GS22] for DNSSEC is called request-based fragmentation. In the context of large resource records in DNSSEC, [GS22] had the first response be a truncated response that included information about the size of the response, and then the client sent multiple additional requests, in parallel, for the remaining fragments. This ensured that there was only one UDP response for each UDP request.

Maintaining one-to-oneness of UDP packets can prevent reflection attacks. We therefore offer an adaptation of that method for TurboTLS: the client can, in its first C->S flow, fragment its own C->S data across multiple UDP packets. Additionally it sends (in parallel) enough nearly-empty UDP requests for a predicted upper bound on the number of fragments the server will need to fit its response. This preserves the model of each UDP request receiving a single UDP response reducing the potential for DDoS amplification attacks.

TODO: we solicit advice on the potential impacts of this method on middlebox compatibility, and whether the benefit in DDoS protection is offset by other presently unknown factors.

3.4. TLS-over-TCP fallback

UDP does not have reliable delivery, so packets may be lost. Since the first TurboTLS round-trip includes the TCP handshake, we can immediately fall back to TCP if a UDP packet is lost in either direction. This will induce a latency cost of however long the client decides to wait for UDP packets to arrive before giving up and assuming they were lost.

In an implementation, the client delay could be a fixed number of milliseconds, or could be variable depending on observed network conditions; this need not be fixed by a standard. We believe that in many cases a client delay of just 2ms after the TCP reply is received in the first round trip will be enough to ensure UDP responses are received a large majority of the time. In other words, by tolerating a potential 2ms of extra latency on *X%* of connections, we can save an entire round-trip on a large proportion *(100-X%)* of the connections. This mechanic was not implemented in the experimental results presented here and constitutes future work.

3.4.1. Early data, post-handshake messages, and TCP fallback

As part of the TLS 1.3 specification, a server is able to send encrypted application data and connection maintenance related messages after it sends its server finished message. One could wait until the TCP connection is established and is associated with the correct UDP handshake. This would remove the benefit that TurboTLS offers as it requires the server to wait for the TCP connection to finish being established. We therefore propose that all post-handshake messages and early data message attempt to be transmitted over UDP. These messages should therefore be wrapped with the standard TurboTLS headers (session ID and index) to ensure that can be associated with the correct TLS session. Once the TCP connection is established, the client's first message should include the index of the last in order UDP based packet that was received. The server

can then determine what needs to be retransmitted over the reliable TCP connection.

In the best case scenario, these early data and post-handshake messages arrive one round trip sooner than they would than in TCP-based TLS, and in the worst cast arrive at the same time as TCP-based TLS. However, this fallback method comes at the cost of requiring additional memory usage by the server to store the messages sent over UDP until it has verified they have been delivered.

3.5. TurboTLS support advertisment

To protect servers who do not support TurboTLS from being bombarded with unwanted UDP traffic, it would be preferable if clients only used TurboTLS with servers that they already know support it. Clients could cache this information from previous non-TurboTLS connections, but in fact we can do better. Even on the first visit to a server, we can communicate server support for TurboTLS to the client, without an extra round trip, using the HTTPS resource record in DNS [SBN22]. Today when web browsers perform the DNS lookup for the domain name in question, they typically send three requests in parallel: an A query for an IPv4 address, an AAAA query for an IPv6 address, and a query for an HTTPS resource record [SBN22]. Servers can advertise support for TurboTLS with an additional flag in the HTTPS resource record and clients can check for it without incurring any extra latency.

3.6. Specification: Handshake embedding into UDP

Rather than relying on IP fragmentation, and the issues that may arise from IP/UDP fragmentation, we fragment at the application layer and send a new UDP packet each time the packet size would exceed a predefined *safe* size. This predefined size will need to account for the various headers and metadata being sent in each packet. In DNS, for example, the recommended maximum payload size is 1232 bytes to account for IPv6, and UDP headers [DNSUDP]. As previously mentioned, TurboTLS UDP packets contain a session ID, and a sequence number. These Turbo-headers will be prefixed to the payload of each UDP packet being set. Both client and server UDP communication streams have their own distinct sequence counters to maintain ordering in either direction.

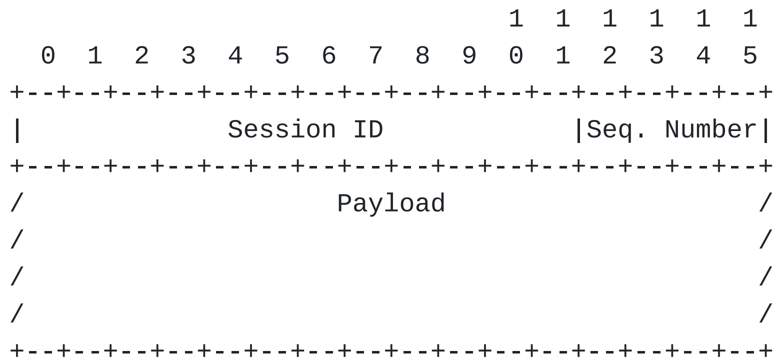


Figure 3: TurboTLS UDP packet layout

3.7. Specification: UDP Tombstone

As part of joining the UDP and TCP streams together once a TCP connection is established, the cleint sends a tombstone message as the first message over TCP. This message contains the session ID as well as the sequence number of the last in order UDP packet that it received. The server will then use this message to determine which handshake a given TCP stream should be associated with, as well as what data needs to be retransmitted due packet loss.

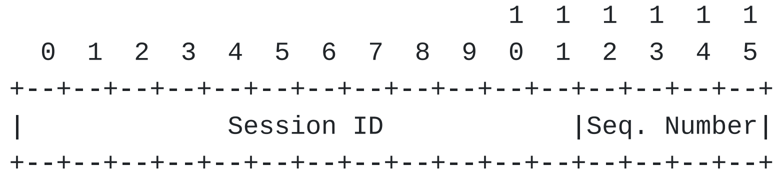


Figure 4: TurboTLS tombstone packet layout

4. Security Considerations

4.1. Transparent proxying

TurboTLS benefits from a nice feature: TurboTLS makes no change whatsoever to the content of a TLS handshake, only changes the delivery mechanism. As a result, all cryptographic properties of TLS are untouched. In fact, it is possible to implement TurboTLS without changing the client or server's TLS library at all, and instead use transparent proxies on both the client and server side to change the network delivery from pure TCP in TLS to UDP+TCP in TurboTLS. Of course in such a construction the initial client or server, who does not know TurboTLS, will observe two round trip times, but if each proxy is close to its host (say on the same machine), then the two round trip times will be negligible, and the higher latency client--server distance will only be covered over one round trip.

4.2. Denial-of-Service

We now consider the implications for TurboTLS of various types of denial-of-service and distributed denial-of-service attacks, including whether a TurboTLS server is a victim in a DoS attack or being leveraged by an attacker to direct a DDoS attack elsewhere. TurboTLS runs on top of both TCP and UDP so we have to consider attacks involving both protocols.

4.2.1. Attacks *on* TurboTLS servers

The most significant TCP DoS attack is the SYN flood attack where a target machine is overwhelmed by TCP SYN messages faster than it can process them. This is because a server, upon receiving a SYN, typically stores the source IP, TCP packet index number, and port in a `SYN queue', and this represents a half-open connection. An attacker could flood the server with SYN messages thereby exhausting its memory. The server cannot just arbitrarily drop connections because then legitimate users may find themselves unable to connect. There are multiple protections against SYN flood attacks, such as:

* Allocating only very small amounts (micro blocks) of memory to half-open connections.

* Using TCP cryptographic cookies [Ber05] [Sim11] whereby the sequence number of the ACK encodes information about the SYN queue entry so that the server can reconstruct the entry even if it was not stored due to having a full SYN queue. TCP cookies enjoy support in the Linux kernel -- this and other such mitigations are already sufficient to protect TurboTLS from SYN floods.

In general there are several vectors to consider for resource exhaustion attacks on a server running TurboTLS. The server needs to maintain a buffer of received UDP packets containing fragments of a TLS CH message.

* To avoid memory exhaustion attacks, a server can safely bound the memory allocated to this buffer and flush old entries on a regular basis (e.g. after two seconds).

  - In the worst case, a legitimate client whose UDP packets are rejected from a busy server or flushed early will be able to fall back to vanilla TLS over TCP, and will incur negligible latency loss (compared to TLS over TCP) in doing so, because TurboTLS starts the TCP handshake in parallel to the first C->S UDP flow.

* An attacker spoofing IP addresses and sending well-formed CH messages could also try to exhaust a server's CPU resources by causing a large amount of cryptographic computation.

  - Again, a server under attack can limit the CPU resources allocated to UDP-received CH messages, and then fall back to vanilla TLS over TCP. In the worst case, legitimate clients affected by this and having to fall back to vanilla TLS over TCP will incur negligible latency loss compared to TLS over TCP since the TCP handshake has already been started in parallel.

4.2.2. Attacks *leveraging* TurboTLS servers

UDP reflection attacks present another threat. Typical defenses against these are: - blocking unused ports, - rate limiting based on expected traffic loads from peers (exorbitant traffic loads are likely to be malicious), - blocking IPs of other known vulnerable servers. However such defenses are provided by middleboxes and therefore do not affect the protocol.

The one-to-oneness of the UDP request/response significantly reduces the impact of any amplification attack which tries to utilize a TurboTLS server as a reflector: an attacker would have to send one UDP packet for every reflected packet generated by the server, meaning that initial requests and responses are of comparable sizes, making the amplification factor so low that it would be an ineffective use of resources. Furthermore, the UDP requests ultimately must contain a fully formed CH before the server responds, limiting the amplification factor.

5. References

5.1. Normative References

[ABGGJS23] Carlos Aguilar-Melchor, Thomas Bailleux, Jason Goertzen, Adrien Guinet, David Joseph, and Douglas Stebila, "TurboTLS: TLS connection establishment with 1 less round trip", n.d., <https://arxiv.org/abs/2302.05311>.

[TCP] Postel, J., "Transmission Control Protocol", RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/rfc/rfc793>.

[TLS12] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/

RFC5246, August 2008, <https://www.rfc-editor.org/rfc/rfc5246>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

5.2. Informative References

[Ber05] Daniel J Bernstein, "SYN cookies", 1 December 2005, <https://cr.yp.to/syncookies.html>.

[DNSUDP] Axel Koolhaas and Tjeerd Slokker, "Defragmenting DNS - Determining the optimal maximum UDP response size for DNS", 11 September 2020, <https://indico.dns-oarc.net/event/36/contributions/776/>.

[GS22] Douglas Stebila and Jason Goertzen, "Post-Quantum Signatures in DNSSEC via Request-Based Fragmentation", 25 November 2022, <https://link.springer.com/chapter/10.1007/978-3-031-40003-2_20>.

[SBN22] Benjamin M Schwartz, Mike Bishop, and Erik Nygren, "Service binding and parameter specification via the DNS (DNS SVCB and HTTPS RRs)", 11 October 2022, <https://datatracker.ietf.org/doc/draft-ietf-dnsop-svcb-https/11/>.

[Sim11] Simpson, W., "TCP Cookie Transactions (TCPCT)", 2011, <https://www.rfc-editor.org/rfc/rfc6013>.

[SW19] Linjian Song and Shengling Wan, "ATR: Additional Truncated Response for Large DNS Response", 10 September 2017, <https://datatracker.ietf.org/doc/html/draft-song-atr-large-resp-00>.

Authors' Addresses

Douglas Stebila
University of Waterloo

Email: dstebila@uwaterloo.ca

David Joseph
SandboxAQ

Email: dj@sandboxaq.com

Carlos Aguilar-Melchor
SandboxAQ

Email: carlos.aguilar@sandboxaq.com

Jason Goertzen
SandboxAQ

Email: jason.goertzen@sandboxquantum.com