In-Kernel Cryptography Expansion for eBPF Fast-Relays

Daniel Pfeifer

December 1, 2024

Abstract

In this paper we present an extension for the previously proposed eBPF Fast-Relay setup [1]. The extension adds support for in-kernel de- and encryption of QUIC short-header packets utilizing the ChaCha20-Poly1305 AEAD cipher. Together with a theoretical setup proposal we provide a proof-of-concept implementation for ChaCha20 eBPF-decryption for a Go-based sample program.

1 Introduction

Your introduction goes here.

2 Background

We build upon our previous work on eBPF Fast-Relays [1] and add support for the ChaCha20-Poly1305 AEAD cipher as it is used in the TLS 1.3 protocol and thus also in QUIC. This now brings our initial idea of a fast-relay setup closer to a potential real-world application since cryptography is obviously a crucial part of any production-grade network application.

3 Methodology

ChaCha20, since it is a stream cipher, proves to be an easier-to-use algorithm for in-kernel eBPF implementation since the bitstream blocks, that are used for en- and decryption, can be pre-computed by the user-space program and then handed down to the kernel program via simple eBPF maps. Using the

packet-number and the number of the 64-byte block needed for decryption is sufficient to look-up the correct bitstream as shown by our proof-of-concept implementation. This approach, however, will need one additional info-field for the lookup-key once multiple connections are handled by the same eBPF program. Adding a simple integer bitstream-id to which all the connection ids of one connection will map will be sufficient to mitigate this issue. Such a mapping could be handled by the userspace or by the eBPF program itself.

4 Results

Your results go here.

5 Discussion

In order for our setup to be fully TLS 1.3 compliant, since TLS 1.3 does not allow the user to manually select one of the allowed ciphersuites, we would need to add support for all other ciphersuites that are potentially used in the protocol. This would mean implementing AES-GCM and AES-CCM support for eBPF en- and decryption. The issue here is that AES is a block cipher and thus the ChaCha approach of pre-computing the bitstream blocks and simply applying an XOR operation to the packet is not possible. More research would need to be conducted to find a suitable approach for this problem that stays eBPF-verifier compliant.

6 Conclusion

Your conclusion goes here.

7 References

References

[1] Daniel Pfeifer. *eBPF-Assisted Relays for Multimedia Streaming*. Accessed: Dec 1st, 2024. 2024. URL: `https://github.com/danielpfeifer02/tum-thesis-bsc-fast-relays.git`.