Scaling the Performance of Modern TLS Traffic Monitoring without Compromising the Security

Junghan Yoon[*†] *KAIST* Seunghyun Do[*†] *KAIST* Duckwoo Kim[*†] *KAIST* Taejoong Chung[*] *Virginia Tech* Kyoungsoo Park[*] *KAIST*

1 Introduction

Transport Layer Security (TLS) is increasingly popular in the modern Internet [7] as it ensures secure private network communication. While this is beneficial to endpoints, it poses a serious problem in traffic monitoring. Today's TLS middlebox intentionally operates as a man in the middle (MITM) – it impersonates as *any* site that a client intends to visit and inspects the stream while it relays the encrypted traffic between the two endpoints. The impersonation is typically realized by installing a custom root certificate at a client [2], which nullifies the key properties of TLS beyond giving up confidentiality, i.e., no end-to-end authentication nor content integrity is guaranteed.

There have been a number of research works [3–5] that attempt to address the problem. These proposals enable explicit middlebox authentication while they extend the key exchange process so that endpoints (and/or middleboxes) can readily detect illegal modification by non-privileged middleboxes. Unfortunately, none of these systems consider performance as their primary goal, and they often incur significant overheads. This is problematic as the CPU advancement has stagnated while the network bandwidth keeps increasing.

We tackle the performance issue from a new perspective. Our key observation is that traffic monitoring middleboxes in the client side are often predominantly read-only [3]. For read-only middleboxes, we argue that using a generalized architecture like mbTLS [4] or maTLS [5] is highly inefficient due to the complexity in key sharing and extra computation. Also, they operate by terminating a connection from a client and initiating a separate connection to the server (which we call "split-connection"). However, split-connection incurs a huge overhead as it not only relays the traffic between two TCP connections but it also translates the encrypted content from one endpoint to the other.

[*]{cerotyki, acornhight, 0731kdw14, tijay00, kyoungsoo}@gmail.com
[†]student

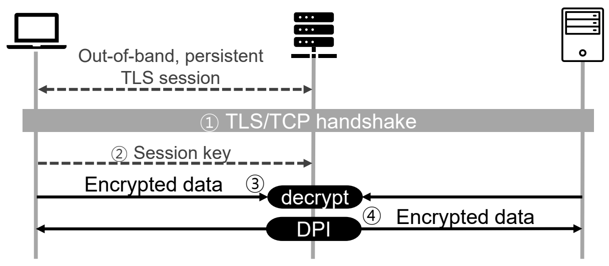

Figure 1: Operations of an monTLS middlebox

2 Approach

Basic scheme. We present monTLS, a highly-scalable TLS traffic monitoring architecture that preserves the end-to-end TLS properties except confidentiality. The key idea of monTLS is very simple – monTLS keeps the original end-to-end TLS connection intact while it allows TLS endpoints to securely share the session keys with TLS middleboxes for traffic monitoring. Figure 1 shows the overview of the operations. In the setup phase, an monTLS middlebox establishes an out-of-band persistent TLS session with each client. This TLS session is used for secure key delivery and explicit middlebox authentication. Then, the middlebox operates by forwarding packets between the two endpoints rather than relaying the content on two TLS connections. After a client initiates a TLS handshake with a server (①), it shares the session keys with the middlebox over the secure channel (②). Then, the middlebox reassembles each TLS record from incoming TCP packets, decrypts the record if it collects a complete one, and runs a monitoring function (by deep packet inspection) on the plaintext. The middlebox can block or terminate the connection if the content turns out to be malicious. Underneath the table, the middlebox keeps relaying the original packets to the other endpoint (④). In case a new incoming packet completes a TLS record, the middlebox forwards the packet only after the DPI function confirms that the content is innocent. While a partial TLS record that contains malicious content can be delivered to the other endpoint, this would not be delivered to the upper layer as the TLS standard dictates that a received TLS

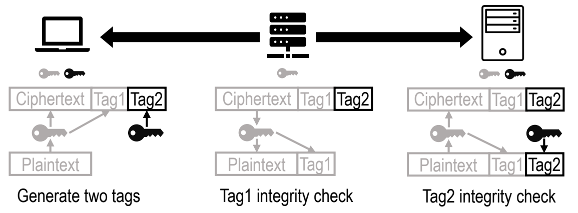

Figure 2: Private tag generation for end-to-end content integrity

record must be verified in full before being delivered to the upper layer [1]. Overall, our base design focuses on high performance – the problem is reduced to monitoring a single TCP connection without re-encryption. The middlebox does not run congestion/flow control nor need to ensure reliable data delivery between the two endpoints. All packets are forwarded without re-encryption nor being queued up for a complete TLS record.

Support for end-to-end content integrity. While the basis scheme ensures end-to-end authentication, it does not guarantee end-to-end content integrity (a.k.a. data authentication). As TLS 1.3 mandates authenticated encryption with additional data (AEAD), sharing the key with a middlebox implies that the middlebox can modify the data but still generate a correct tag even if it is read-only. To address this problem, we propose adding a second tag to the TLS record where the key for this tag is kept secret only to endpoints. While this requires updating the handshake process, it can be implemented as an extension. Using the second tag, the receiver can detect illegal content modification in the middle while the middlebox can verify content integrity with the original tag (see Figure 2).

Implementation. We implement monTLS middlebox as an application of mOS [6], which provides a set of events and APIs for monitoring the flow-reassembled TCP content. To fully utilize CPU in the middlebox, we leverage a SoC SmartNIC (Bluefield-2) as a scalable session key receiver. The key receiver on the SmartNIC kernel is implemented as an epoll-based OpenSSL server.

3 Preliminary Performance

We compare the performance of an monTLS middlebox against existing solutions. For the baseline, we use nginx TLS proxy (v1.22.0) to represent an existing MITM middlebox (splitTLS), and we also compare against mcTLS[1] as a secure split-TLS middlebox. For the middlebox, we use a machine equipped with 16-core Xeon Gold 6326 @ 2.90GHz and Bluefield-2 dual-port 100GbE NIC, and we make sure that neither clients nor servers become the bottleneck. The TLS middleboxes decrypt the content but do not run any DPI applications like pattern matching. Clients request objects of a varying size with 4,000 concurrent, persistent TLS connections with either ECDHE_RSA_AES_256_GCM_SHA384 (v1.3) or DHE_RSA_AES_256_CBC_SHA256 (v1.2) [2]. Figure 3 shows that monTLS improves the performance of TLS traffic monitoring by 2.5x to 4.5x over splitTLS (nginx) in the GCM mode, and by 1.5x to 3.0x in the CBC mode. The performance improvement over mcTLS is much larger – by 2.4x to 6.3x due to the extra overhead of mcTLS. When all connections are ephemeral and the object size is small (1KB), monTLS outperforms splitTLS by 63.5x as splitTLS must run TLS handshake per each split connection while monTLS does not participate in the handshake – it simply relays the packets for TLS handshake for endpoints.

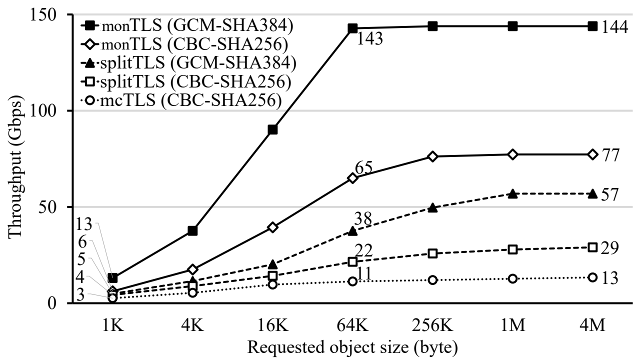

Figure 3: Monitoring throughputs over varying content sizes

4 Conclusion

TLS and TLS traffic monitoring are fundamentally in conflict as the latter operates by breaking at least one property of TLS – confidentiality. However, TLS middleboxes are unlikely to go away unless the needs for security monitoring disappears. In this abstract, we argue that existing security enhancement works have neglected the performance aspect and that a generalized architecture would be often unnecessary for client-side monitoring. We show that monTLS brings a great potential for improving the performance without sacrificing the security.

Acknowledgments

We thank Samsung SDS Technology Research for lively discussion on the system architecture and for providing us with the information on real-world TLS middlebox operation. This work is in part supported by Samsung SDS, Inc. as well as the the ICT Research and Development Program of MSIP/IITP, Korea, under [2018-0-00693, Development of an ultra low-latency user-level transfer protocol].

References

[1] RFC 8446 (TLS 1.3). `https://www.rfc-editor.org/rfc/rfc8446`.

[2] X.deCarn´e de Carnavalet et al. Killed by proxy: Analyzing client-end tls interception software. In *Proceedings of NDSS*, 2016.

[1]Unfortunately, mbTLS [4] is closed source and maTLS [5] does not support large file transfer.

[2]mcTLS [3] currently supports only the CBC mode in TLSv1.2.

[3] David Naylor et al. Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In *Proceedings of ACM SIGCOMM*, 2015.

[4] David Naylor et al. And Then There Were More: Secure Communication for More Than Two Parties. In *Proceedings of CoNEXT*, 2017.

[5] Hyunwoo Lee et al. maTLS: How to Make TLS middlebox-aware? In *Proceedings of NDSS*, 2019.

[6] Muhammad Jamshed et al. mOS: A Reusable Networking Stack for Flow Monitoring Middleboxes. In *Proceedings of USENIX NSDI*, 2017.

[7] F5 Labs. The 2021 TLS telemetry report. https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report.