An SDN-Based Approach to Enhance the End-to-End Security: SSL/TLS Case Study

Alireza Ranjbar*, Miika Komu*, Patrik Salmela*, Tuomas Aura†
*Ericsson Research, Finland †Aalto University, Finland
{alireza.ranjbar, miika.komu, patrik.salmela}@ericsson.com, tuomas.aura@aalto.fi

***Abstract*—End-to-end encryption is becoming the norm for many applications and services. While this improves privacy of individuals and organizations, the phenomenon also raises new kinds of challenges. For instance, with the increase of devices using encryption, the volumes of outdated, exploitable encryption software also increases. This may create some distrust amongst the users against security unless its quality is enforced in some ways. Unfortunately, deploying new mechanisms at the end-points of the communication is challenging due to the sheer volume of devices, and modifying the existing services may not be feasible either. Hence, we propose a novel method for improving the quality of the secure sessions in a centralized way based on the SDN architecture. Instead of inspecting the encrypted traffic, our approach enhances the quality of secure sessions by analyzing the plaintext handshake messages exchanged between a client and server. We exploit the fact that many of today's security protocols negotiate the security parameters such as the protocol version, encryption algorithms or certificates in plaintext in a protocol handshake before establishing a secure session. By verifying the negotiated information in the handshake, our solution can improve the security level of SSL/TLS sessions. While the approach can be extended to many other protocols, we focus on the SSL/TLS protocol in this paper because of its wide-spread use. We present our implementation for the OpenDaylight controller and evaluate its overhead to SSL/TLS session establishment in terms of latency.**

***Index Terms*—Software-Defined Networking, SSL/TLS, Centralized policy management, Handshake analysis, Flow verification**

I. Introduction

The general awareness of Internet privacy issues has been on the rise not only for individuals but also for companies and other organizations. For instance, the Internet Engineering Task Force (IETF) has proposed encryption in HTTP/2 [1]. At the same time, due to the increasing number of hijacking threats in plaintext protocols, some of the organizations are already employing encryption as default in their web services [2]. Thus, the trend of the near future appears to be that an increasing fraction of the Internet communications is protected by end-to-end encryption. While providing privacy and security, this will create challenges for middleboxes such as firewalls, proxies and Deep Packet Inspection (DPI) devices. Another potential downside is that mere encryption of traffic cannot solve all cyber-security problems in the Internet, for instance, ones related to misconfiguration and human errors. In particular, users may accept weak algorithms or out-dated and invalid credentials for their communication.

SSL/TLS [3] (hereon just referred to as TLS) might be the most well-known example of today's security protocols. The TLS protocol is widely deployed in different network-based software and services, for instance web browsers and servers, email clients and servers, instant messengers and teleconferencing applications in order to facilitate end-to-end encrypted sessions. The security of the TLS protocol has received close scrutiny, and the protocol and implementations have been updated over the years to maintain a high level of security. After detecting security vulnerabilities, the vulnerable versions or parameters have been obsoleted by the standards body. However, not all of today's communications are initiated based on the latest versions of TLS. Indeed, a considerable number of applications in today's Internet employ TLS sessions with weak security algorithms and outdated or self-signed certificates [4], [5], [6]. These security credentials are mostly verified at the end-points of the communication, which increases the chance of human errors and prevents implementing of a uniform security policy over all the initiated TLS flows of a domain.

In this paper, we tackle these administrative challenges surrounding security protocols with particular focus on TLS. Our approach is inspired by the fact that security protocols, such as TLS, negotiate their security parameters in a key exchange or handshake protocol. The handshake phase occurs at the beginning of the communication, and it allows two end-points to agree on the credentials and parameters in order to establish an encrypted session. While the encrypted application payload cannot usually be inspected, the handshake occurs in plaintext, which opens an opportunity for inspection and verification by intermediary devices.

Our solution to verifying the TLS handshake protocol is based on the centralized architecture of Software-Defined Networking (SDN) [7]. In SDN, a logically centralized point (controller) can inspect all traffic flows in the network and enforce uniform policies over all flows. Since the data flows in TLS are encrypted, the controller only checks the clear-text credentials and parameters transmitted during the handshake phase, but this is already enough to detect, e.g., outdated certificates or protocol version numbers in the TLS handshake. Violation of a policy can result in an alarm to the operator or even in blocking of the TLS flow. After a successful handshake, the controller allows the TLS client and server to establish an end-to-end encrypted session.

We have implemented a prototype of our solution using

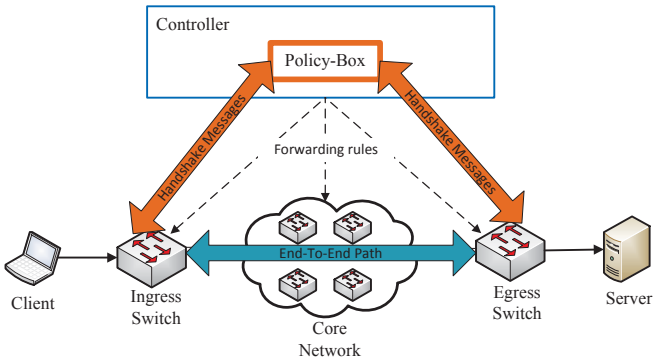

Figure 1: System architecture

the OpenDaylight controller to verify TLS handshakes in real time. The evaluation of our prototype shows that our approach does not have a major impact on the latency of TLS session establishment. While we implemented the prototype based on the TLS protocol, our approach can also be deployed on other protocols with an initial plain-text handshake such as DTLS [8] or SSH [9]. Our solution can even be useful with the recent version of TLS 1.3 [10].

The rest of the paper is organized as follows. Section II explains the design of our solution. Section III describes the details of flow verification of the TLS protocol and how our approach can improve its security. Section IV explains implementation details and Section V presents a performance evaluation of the prototype. The results are discussed and future work is presented in Section VI. Related work is described in Section VII, and Section VIII concludes the paper.

II. Design

This section explains the system architecture of our approach. We first discuss the different system components and then describe how the components interact with each other.

A. Architectural Components

Figure 1 illustrates the different components of our system architecture. The centralized controller implements network services, provides northbound APIs for external applications and configures and controls the network elements at the data plane. The controller is able to program and control the switches by installing, modifying and deleting forwarding rules with southbound protocols.

Additionally, the controller includes a *policy-box* service which is responsible for checking the handshake messages of each protocol based on the policies defined by the network administrator. For instance, this service can be used to check the encryption algorithms and validity of certificates in the TLS handshake process. The policy-box is, however, not bound to a single protocol and can be extended to enforce policies for many different protocols. Moreover, the policy-box might be deployed as a third party service or it could utilize external services for the flow verification process. For instance, it could be connected to an external authenticator or a Certification Authority (CA) to check the validity of the session and the used credentials.

We have chosen OpenFlow as the southbound protocol in our architecture due to its popularity. It allows the centralized controller to have fine-grained control over network flows. Furthermore, OpenFlow enables the switches to transmit the initial packets of each flow to the controller (packet-in messages), and it allows the controller to forward the packets to the switch (packet-out messages), or to install forwarding rules for the new received flow requests (flow-mod messages).

As depicted in Figure 1, a set of OpenFlow-enabled switches at the edge and core network forward the flows based on prioritized forwarding rules installed in their flow tables. The edge switches transmit the handshake messages to the controller, which then instructs the switches at the edge and core network on how to handle these flows. Furthermore, the network includes a client, which initiates a request, and a server that responds to the request from the client. Henceforth, we simply refer to the TLS connection initiator end-point as "client" and responding side as "server".

B. Interaction of Components

The interaction of the components in the system architecture is depicted in Figure 1. As shown in the figure, the switches intercept handshake messages of each flow and inject them to the controller. The SDN controller consults the policy-box to match each packet with the local policies of the network. When the policy-box accepts a handshake message, the controller directly delivers the temporarily postponed message to the destination, bypassing the switches in the network core. The policy-box repeats the process for all handshake messages between the client and server until the handshake is successfully verified. The controller then installs forwarding rules on all of the switches along the path between the endpoints to accept the flow. From this point on, the client and server communicate directly without interventions from the controller.

Thus, the policy-box is able to verify and track the TLS handshake messages from a centralized control point. While this approach can be used for any protocol that starts with an unencrypted initial handshake, we have targeted TLS specifically because of its wide use in various networking applications and services. In the next section, we explain in detail how our solution can be used to increase the security of TLS sessions.

III. TLS Flow Analysis

The TLS [3] protocol for secure communication consists of two main parts: the *Handshake* and *Record* protocols. In the handshake phase, the end-points negotiate a set of cryptographic methods, primitives and keys. In the record phase, the end-points protect their communication using the primitives and keys negotiated during the handshake process. Since the handshake defines the security parameters for the record layer, i.e. for the actual data, it is important to ensure that this negotiation does not use, e.g., weak encryption algorithms

or outdated certificates. Therefore, the recent RFC on TLS [11] provides a list of recommendations, which are mostly concerned with the structure and attributes of the handshake to attain strong security. In the following, we discuss the important attributes in the handshake messages and how our solution can be used for verifying them.

Protocol Versions: The TLS protocol has been improved and updated throughout the years in order to eliminate discovered vulnerabilities. While the recommendations [11] suggest to use the latest TLS version, it may not be supported yet by all of today's services. As a consequence, the servers usually are backwards compatible and allow the clients to negotiate an older versions of TLS. Unfortunately, an attacker may be able to exploit this backwards compatibility and the vulnerabilities of old protocol versions. Proposed countermeasures, which are typically implemented in the TLS servers and clients, include preventing downgrades to known vulnerable versions, and verification of the parameters and algorithms to check that the client is using the latest version of TLS. Our solution moves this verification of the TLS handshake parameters to a centralized controller. From there, the controller can verify that the sessions only use TLS versions and algorithms allowed by the network's security policy. Exceptions to the policy can also be added for legacy clients on per-host basis.

Cipher Suites: A cipher suite gives the combination of the key exchange algorithm, encryption algorithm, message authentication code (MAC) algorithm and pseudo-random function (PRF) algorithm to be used. A secure session will be established between the client and server based on the negotiated algorithms in the chosen cipher suite. The client offers a set of supported cipher suites to the server and server usually selects the strongest one or the cipher suite for which it has hardware acceleration. This means that the selection of cipher suites depends on the algorithms supported by both of the end-points. However, an attacker could tamper with the handshake message and fool the other endpoint of the communication into selecting a weak algorithm. This creates a vulnerable point for the attacker to compromise the session and gain access to the secure channel. To verify the selected algorithms at the controller, our solution analyzes the handshake messages of each flow and only allows the client and server to create secure sessions using specific cipher suites defined in the policy.

Certificates: During the handshake phase, the client and server can exchange certificates in order to identify and authenticate each other. In web applications, it is common for only the server to have a certificate, but in other applications, both sides may have one. These certificates are typically issued by third-party Certificate Authorities (CAs). While the end-points are ultimately responsible of checking the validity of certificates of a session, they can make mistakes. First, users regularly override security controls in order to accept invalid or expired certificates or ones that cannot be verified against a revocation list [12]. In fact, a significant portion of today's servers use expired or self-signed certificates [5], [13]. Moreover, certificate chain validation is a complex process and endpoint implementations have had subtle errors [14]. In addition, even if today's client applications may consider a long list of trusted CAs, all CAs are not equally trustworthy [15]. Therefore, organizations may consider a policy to accept certificates from specific CAs, at least for internal communication. Our proposed solution can enforce centralized verification of specific fields in each certificate, for instance the dates or issuer of the certificate. Also, invalid certificate chains can be rejected without leaving an option for the user or endpoint software to override the controls.

Compression Methods: TLS compression methods can be used during the handshake phase in order to decrease the amount of payload data transferred in a TLS session. This results in a reduction in the required bandwidth and latency for transferring large amounts of data in TLS sessions [3]. However, based on the recommendations [11], this feature introduces vulnerabilities and should be disabled. Since our solution inspects the parameters exchanged in the handshake protocol, it can be configured to block TLS sessions that try to enable compression features.

Handshake Renegotiation: TLS renegotiation gives the client the option to establish a new TLS session through an existing secure channel. Based on the recent updates [16], the secure renegotiation is enabled by including a specific cipher suite or extension to the initial handshake protocol. Again, this introduces the opportunity to detect and filter the sessions that employ the renegotiation extension. With the central control, the network administrator can prevent renegotiation in the network completely or force the end-points to only make secure renegotiation using the specified cipher suite and extension.

IV. Implementation

We have implemented a prototype of the proposed approach using the OpenDaylight controller (Hydrogen release). The prototype consists of roughly 1000 lines of Java code. In this section, we discuss the functionality of the implementation and how it is deployed in order to verify TLS flows. Our prototype enforces some of the TLS recommendations referred to in the previous section, but is possible to extend it to check also other attributes in the handshake.

The policy-box in our implementation is an OSGi [17] bundle defined on the OpenDaylight controller. This module is able to load the policies defined in a simple text file or it can be extended to be configured using the REST APIs. For each received TLS flow, the policy-box matches the defined policy with the flow and subsequently, it accepts or rejects the flow requests. Table I illustrates a sample policy definition for a TLS flow in the policy-box.

The handshake protocol in TLS consists of several messages which are exchanged between a client and server before establishing a secure session. Figure 2 depicts the message flow between the two end-points that passes through the centralized controller. We have simplified the example in Figure 2 by assuming that only two switches exist between

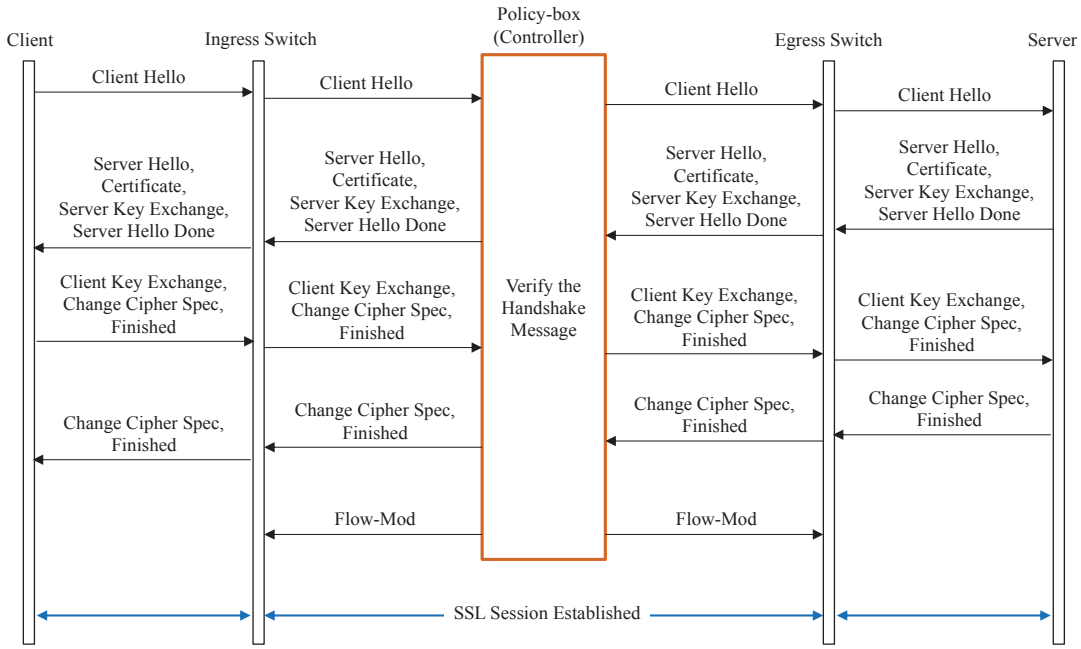

Figure 2: Flow diagram of SSL session establishment

the client and server. As it is depicted in the figure, the edge switches forward all handshake messages to the controller for checking credentials in these messages. When the handshake is verified, the policy-box establishes a path between the client and server by installing the forwarding rules (flow-mod messages in the OpenFlow protocol) to the switches. After this, the session continues directly between the switches without further intervention from the controller.

Algorithm 1 describes the program logic of our prototype for verifying TLS flows. The policy-box creates a new session and tracks the session until the client and server have exchanged all the relevant handshake messages. The prototype checks only certain values in the handshake messages in order to optimize the performance of the prototype. It searches for the location of certain TLS parameters and matches their values with the policy of the network.

To start establishing a secure session, the client sends a *Client Hello* message, and the server responds with a *Server Hello* message. The Client and Server Hello messages determine the supported TLS version, the list of supported cipher suites, extensions and compression algorithms. As explained in the previous section, negotiating vulnerable values for any of these parameters may allow attackers to exploit the vulnerabilities in the TLS protocol and compromise the session.

Certificates are another important piece of information transmitted between the end-points. The server sends its certificate chain to the client so the client is able to verify the identity of the server. Optionally, the server might ask the client to send its certificate chain. As already explained, the certificates used in the handshake can be expired or otherwise untrustworthy. Our prototype verifies the issuer of the certificates and validates the certificates based on the date of issue and expiry.

The end-points exchange other handshake messages such as *Server Hello Done*, *Server Key Exchange* and *Client Key Exchange* without major inspection from the policy-box. After receiving the *Change Cipher Spec* and *Finished* message from the server, the policy-box installs forwarding rules on the switches in order to establish a direct path between the client and server.

For each established session, the controller is able to decide an idle time-out which is embedded in the forwarding rules on the switches. If the session is inactive for this period of time, then the flow will be removed from the switches. The value of the idle timeout for the flow tables can be set according to the TLS session idle timeout at the server side or it can be configured based on the network policy.

V. Evaluation

One of the main concerns in our solution is the increased latency in establishing TLS sessions. Since the SDN controller verifies the handshake in our approach, it is important to measure the overhead in establishing TLS sessions. Thus, we focus here on evaluating our approach in terms of latency. More precisely, we generate HTTPS traffic between a client and server and measure the impact of our solution on the time it takes to establish a TLS session.

The test network for the evaluation includes client and server hosts which are connected to each other using Open-

Algorithm 1: TLS flow verification logic

```
session_ssl ← sessionDB.get(pkt_ssl);
if session_ssl is null then
    create_session(pkt_ssl);
if pkt_ssl is verified session then
    install_forwarding_rules(pkt_ssl);
    return;
if pkt_ssl includes Client Hello then
    verify_version(pkt_ssl.version);
    veriy_cipher_suites(pkt_ssl.ciphers);
    verify_renegotiation_info(pkt_ssl.reneg);
    verify_compression_algorithms(pkt_ssl.comp);
    update_session(session_ssl, client-hello);
else if pkt_ssl includes Server Hello then
    verify_selected_cipher_suite(pkt_ssl.cipher);
    update_session(session_ssl, server-hello);
else if pkt_ssl includes Certificate then
    validitate_certificates(pkt_ssl.certificates);
    update_session(session_ssl, certificates);
else if pkt_ssl includes Server Hello Done or Server
Key Exchange or Client Key Exchange then
    update_session(session_ssl, server-hello-done /
    server-key-exch / client-key-exch);
else if pkt_ssl includes Change Cipher Spec and
Finished then
    update_session(session_ssl, change-cipher-spec,
    finished);
    install_forwarding_rules(pkt_ssl);
```

Table I: Sample policy definition in policy-box

```
{
  "TLS Flow Attributes": {
     "version": "TLS 1.2",
     "cipher suites": [
        {TLS_DHE_RSA_WITH_AES_128_GCM_SHA256},
        {TLS_DHE_RSA_WITH_AES_256_GCM_SHA384}
     ],
     "certificate issuers": [
        {DiGiCert}, {VeriSign}, {Symantec}
     ],
     "compression": "disabled",
     "secure renegotiation": "enabled"
     }
}
```

vSwitch v2.0.2. The network is emulated using Mininet v2.1.0 [18] on a machine running Ubuntu 14.04 and equipped with Core i7 Quad Core 2.80 GHz CPU and 8G RAM. We run the prototype on the OpenDaylight controller (Hydrogen Release) using a separate machine installed with Ubuntu 12.04 and equipped with Intel Core i5 Quad Core 2.67 GHz CPU and 4G RAM. The two hosts are connected directly to each other using gigabit Ethernet ports.

Moreover, we have defined a custom policy in our prototype

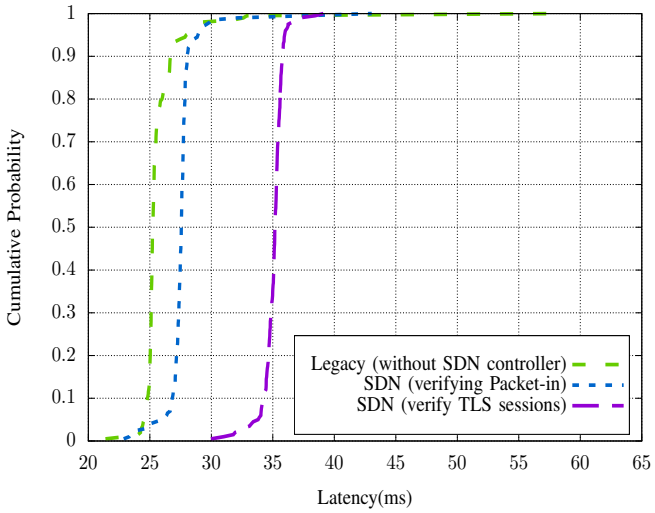

Figure 3: CDF graph for httping latency

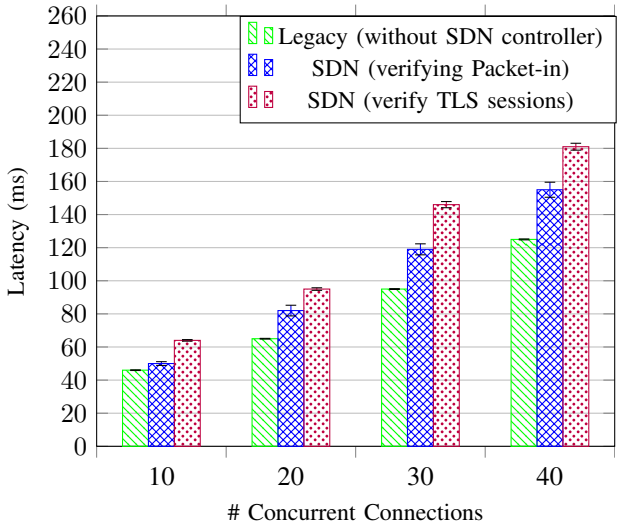

Figure 4: Measured latency for concurrent connections

to accept TLS 1.2 with five acceptable cipher suites and ten recognized issuers of certificates.

We consider three different use cases and measure and compare the latency in each case. In the first case, we measure the time it takes to establish a TLS session between a client and server without any SDN controller. In this case, all flows are installed pro-actively and the controller does not check any information in the TLS session. The second case is the legacy, reactive SDN approach in which the switch sends the first packet of a flow (packet-in message in the OpenFlow protocol) to the controller, which then installs forwarding rules between the client and server. In this case, the controller establishes the forwarding path by transmitting only the packet-in messages. The third case is our proposed solution, in which the switch forwards handshake messages to the controller for inspection in the policy-box.

For testing the latency of establishing TLS sessions, we benchmark our prototype by initiating HTTPS requests from the client to the server. Our benchmarking tool is httping v1.5.8 [19], which establishes a TLS session with the server and measures the time needed for initiating the session. We repeat our measurement 200 times in order to increase the accuracy of the results. The results of the measurements are depicted in a CDF graph in Figure 3. As expected, the first legacy approach is the fastest and our approach is the slowest. However, the difference in the mean value between the lowest and highest latency is less than 10 ms. Comparing our approach with the second approach (which is based on transmitting the packet-in messages), we observe that the difference in latency is less than 8 ms.

Since httping shows the time for a single TLS connection, it cannot show the latency of establishing concurrent TLS sessions. Many of today's web servers may receive several concurrent TLS connections. Therefore, we extend our evaluation to measure the latency of establishing concurrent TLS sessions. We use AB Apache [20] to send connection requests in four concurrency levels (10, 20, 30 and 40 simultaneous clients), between a client and web server. For each concurrency level, AB sends 100 requests to increase accuracy of our measurements (i.e for 40 concurrent connections we send 4000 requests). Figure 4 shows the mean values and standard error for the different numbers of concurrent TLS connections. As the concurrency increases, the mean time for establishing TLS sessions increases, as does the variation. The highest latency and standard error occurs in the case 40 concurrent connections. In this case, the mean value of latency for the first legacy approach is about 125 ms. The second approach increases the latency with about 24% when compared to the first case. Our approach further increases the latency with around 16% when compared to the second case.

VI. Discussion and Future Work

Our solution can be seen as a centralized method for enhancing the security of communication flows. In this paper, we have focused on TLS as an example protocol. Based on the SDN paradigm and the OpenFlow protocol, we have developed a module to verify all initiated TLS flows from a centralized control point. Unlike today's solutions such as SSL proxy [21], our approach does not break the TLS sessions. In fact, by offloading only the handshake messages to the centralized controller, we have increased the security of establishing sessions while retaining the efficiency of end-to-end encryption between a client and server.

Our approach increases the latency compared to the native reactive SDN approach in which only the packet-in messages are transmitted through the controller. While this level of latency is acceptable for initiating TLS sessions, most of the delay consists of transmitting several messages through the centralized controller and processing the messages based on the defined policy. The delay can be somewhat mitigated by the TLS resumption feature [3], [22], which reduces the number of handshake messages. TLS resumption provides the possibility to resume previously initiated sessions with just a short handshake. This means, the clients need to initiate the complete TLS handshake for the first time and for subsequent requests, they can rely on the TLS resumption feature to decrease the latency of re-establishing TLS sessions. This feature is useful especially in cases where the client resumes a session after a server timeout or when it connects with the same data flow from a different location and IP address. Furthermore, the policy-box can utilize additional special purpose hardware for performing heavy crypto functions in order to minimize the processing delay.

The centralized control point increases the manageability of the network and makes it possible to define uniform policies for all TLS flows in the network. However, a large number of initiated flows might overflow a centralized controller. Considering that the controller is responsible for all of the functions in the SDN architecture, the controller may hit its performance limits sooner than expected. Inspired by Casado et al [23], we propose to separate the control plane to edge and fabric controllers in order to support different services and functionalities: the edge controller analyzes the handshake messages and verifies the sessions, whereas the fabric controller is responsible for other tasks such as routing and load balancing at the core network. Another solution to decreasing the load on the controller is the clustering approach [24]. Clustering makes it possible to spread the control traffic to several logically-centralized controllers. The OpenDaylight controller supports this feature, and our solution can be deployed in the clustering mode.

Another thing to consider is that today's OpenFlow-enabled hardware switches are limited with regards to storing large numbers of concurrent forwarding rules [25], which can affect our solution since our design is based on filtering the traffic at the edge switches. This issue can be mitigated by considering software switches at the edge of the network. Today's software-based switches are able to store a large number of forwarding rules [26].

The proposed solution is effective in terms of preventing many of today's network attacks. POODLE [27], BEAST [27] and FREAK [27] are examples of well-known attacks which are mainly mounted using the vulnerabilities of old and weak cipher suites in the TLS protocol. In these types of attacks, the attacker is placed in the middle of the communication channel in order to break the encrypted session by exploiting vulnerabilities in weak encryption algorithms. Our approach can prevent these attacks by forcing the client and server to choose the latest TLS versions and by only allowing strong cipher suites. Other examples of today's network attacks are CRIME [27], TIME [27] and BREACH [27], in which the attacker exploits the compression feature in TLS in order to hijack the TLS sessions by recovering the session information in the web cookies. However, we can prevent these attacks disallowing the client and server to establish TLS sessions with compression methods.

One of the security objectives of the currently worked on TLS v1.3 [10] is to extend encryption to the handshake

protocol itself and also the exchange of session credentials. However, our solution is still useful for long into the future because many applications will continue to communicate using older versions of TLS, and our approach gives a possibility to work with different versions of TLS. Moreover, as already argued, despite the fact that the current implementation only supports the TLS protocol, the principle is not limited to the TLS and can be extended to support other protocols. In fact, this approach can be used with any protocol that is established with an initial handshake phase with some parameters exchanged in plaintext. One example of a security protocol that can be verified with our architecture is DTLS [8]. Similar to TLS, the DTLS protocol is established with a plaintext handshake protocol. SSH [9] is another example which starts with a negotiation of the encryption algorithms and cipher suites.

Another case where the approach of this paper could be applied is the Host Identity Protocol (HIP) [28]. HIP can be used as a Virtual Private Network (VPN) solution that operates end to end between two hosts. Unlike many other VPNs, the control protocol of HIP is just integrity-protected with asymmetric keys, which means that middleboxes can inspect most of the control traffic. The control protocol authenticates the two communicating end-hosts using their public keys and optionally with certificates, and negotiates cipher suites and keys for IPsec, which means that our approach could be used to analyze various parameters similarly to the TLS case. Furthermore, in HIP, both the client and the server always communicate their public keys, and these could be used to authenticate both of the end-hosts in order to implement a HIP-based (distributed) firewall functionality at existing middleboxes, rather than building and deploying new devices implementing HIP firewall functionality [29].

Additionally, our approach can also be integrated with application layer protocols such as the Session Initiation Protocol (SIP) [30]. With SIP, two end-points exchange signaling messages in order to establish a media session. These messages include the credentials of the end-points and the media formats for the new session. The policy-box in our architecture can verify the end-point credentials and the media formats for establishing the media session.

Finally, the central view and control over session parameters means that the system administrators can be notified about potential weaknesses in the observed end-host behavior even if the sessions are not outright blocked. Our solution introduces a higher level of flexibility by enabling the administrators to enforce polices over specific flow requests (e.g. TLS flows) while allowing them to implement other network functions on the rest of the traffic.

VII. Related Work

To the best of our knowledge, the solution proposed in this paper is unique in a sense that it enhances the security of data flows by analyzing the handshake messages in a centralized SDN controller. However, there are other efforts which have used an SDN controller for verifying various aspects of data flows using the packet-in messages in the OpenFlow protocol. Also, some non-SDN solutions discuss enhancing the TLS flows by verifying the flows in communication networks. In the following, we discuss each group of these works.

SDN firewall solutions: Typically, in these solutions, the first packet of each flow is forwarded to the controller and the controller is responsible of enforcing the pre-defined ACLs on the flows and forwarding the packets through the network. Following this approach, a firewall application has been implemented using the Floodlight controller [31].

Wang et al. [32] have proposed a firewall application to solve the so-called bypass threats in networks by tracking the flows. Their solution detects and resolves conflicting firewall rules based on header-space analysis algorithms. FLOWGUARD [33] is a recent solution that both verifies flows using dynamic policies and detects conflicting rules and security violations. FLOWGUARD verifies the flows centrally at the controller based on the defined policies and also tracks the verified flows in the network, so that it can dynamically react to policy and network-state updates. For isolation and tracking of flows, FLOWGUARD checks the source and destination address of each packet in a flow. Based on the addresses, it creates a *flow path space* for routing a flow in the network.

TLS flow enhancements: Enrique et al. [34] proposed a distributed solution to mitigate man-in-the-middle attacks by verifying TLS certificates using *Bayesian networks*. The SDN controller is used in this solution to reroute and forward suspicious TLS flow requests to a quarantined network. However, this solution does not verify other important parameters of the handshake protocol.

Another well-known solution which can be used for TLS flow verification is Snort [35]. Snort is able to track the flows and set the rules in order to allow certain TLS versions and TLS flow states in the network. Our solution is comparable to the Snort with the advantage of enforcing the network-wide policies from a centralized and consistent control point. Authors in [36], [37], [38] discussed other approaches to verifying the TLS protocol implementation at the end-points in order to enhance the security of TLS flows. However, these solutions lack a centralized network-view to be able to manage and verify all initiated flows in the network.

VIII. Conclusion

In this paper, we proposed a centralized solution based on the SDN architecture to enhance the security of communication sessions. We focused especially on the widely adopted TLS protocol, in which we observed the plaintext parameters in the handshake protocol in order to monitor and enforce security of the session establishment. In our solution, a centralized controller analyzes the parameters of a TLS handshake, such as protocol version, cipher suites and certificates, and allows the client and server to establish an end-to-end encrypted session with each other but only if the network's security policy is followed. That is, we enhanced the security of establishing sessions by only allowing

the communication end-points to initiate a secure session with known strong encryption algorithms and valid security credentials. This approach enables us to enforce network-wide security policies and to manage all secure connections from a centralized control point.

We have developed a prototype of the proposed solution based on the OpenDaylight SDN controller. Our experiments showed that the TLS handshake analysis introduces roughly 16% overhead in verifying 40 concurrent HTTPS connections when compared to a more traditional SDN-based approach. We argue that this overhead is justified by the benefits of the central control over policy compliance. Our approach can improve the overall security by preventing several well-known attacks on the TLS protocol, and it can be extended to support other protocols and applications.

ACKNOWLEDGEMENTS

We are grateful to the anonymous reviewers of this paper for their insightful reviews. Also, special thanks to John Mattsson, Daniel Migault, Bengt Sahlin and Eva Fogelstrom for reviewing the paper and for their valuable feedback. We would also like to thank Suvi Koskinen for verifying the statistics of our initial evaluation. This project was supported by the CyberTrust program of DIGILE (the Finnish Strategic Center for Science, Technology and Innovation in the field of ICT and digital business).

REFERENCES

[1] "Pcworld: Next-gen http 2.0 protocol will require https encryption," Available: http://www.pcworld.com/article/2061189/next-gen-http-2-0-protocol-will-require-https-encryption-most-of-the-time-.html .

[2] "Symantec: Facebook implements always on ssl," Available: http://www.symantec.com/connect/blogs/facebook-implements-always-ssl .

[3] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), IETF, Aug. 2008.

[4] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in ssl/tls implementations," in *Security and Privacy (SP), 2014 IEEE Symposium on*, May 2014, pp. 114–129.

[5] L. Zhang *et al.*, "Analysis of ssl certificate reissues and revocations in the wake of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14. ACM, 2014, pp. 489–502.

[6] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your ps and qs: Detection of widespread weak keys in network devices," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA: USENIX, 2012, pp. 205–220.

[7] "Sdn architecture," Issue 1.0, TR-502, June 2014, Open Networking Foundation, Available at https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf .

[8] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347 (Proposed Standard), IETF, Jan. 2012.

[9] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol," RFC 4253 (Proposed Standard), IETF, Jan. 2006.

[10] E. Rescorlar, "The Transport Layer Security (TLS) Protocol Version 1.3," (Standards Track), Internet Engineering Task Force, Jul. 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-07

[11] Y. Sheffer, R. Holz, and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," RFC 7525 (Best Current Practice), IETF, May 2015.

[12] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying wolf: An empirical study of ssl warning effectiveness," in *Proceedings of the 18th Conference on USENIX Security Symposium*, ser. SSYM'09. USENIX Association, 2009, pp. 399–416.

[13] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The ssl landscape: A thorough analysis of the x.509 pki using active and passive measurements," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. ACM, 2011, pp. 427–444.

[14] A. Bates *et al.*, "Securing ssl certificate verification through dynamic linking," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. ACM, 2014, pp. 394–405.

[15] A. Arnbak, H. Asghari, M. Van Eeten, and N. Van Eijk, "Security collapse in the https market," *Queue*, vol. 12, no. 8, pp. 30:30–30:43, Aug. 2014.

[16] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension," RFC 5746 (Proposed Standard), IETF, Feb. 2010.

[17] "Osgi," Available: http://www.osgi.org.

[18] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. ACM, 2010, pp. 19:1–19:6.

[19] "Httping," Available: http://linux.die.net/man/1/httping.

[20] "Ab apache," Available: http://httpd.apache.org/docs/2.2/programs/ab.html.

[21] J. Jarmoc, "Ssl/tls interception proxies and transitive trust," Presentation at Black Hat Europe, March 2012.

[22] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State," RFC 4507 (Proposed Standard), IETF, May 2006.

[23] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving sdn," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. ACM, 2012, pp. 85–90.

[24] Z. Caor, "Analysis of SDN Controller Cluster in Large-scale Production Networks," Informational, Internet Engineering Task Force, Jul. 2013. [Online]. Available: http://tools.ietf.org/html/draft-zcao-sdnrg-controller-00.html

[25] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow macs: Scalable label-switching for commodity ethernet," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. ACM, 2014, pp. 157–162.

[26] B. Pfaff *et al.*, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130.

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

[28] T. Heer and S. Varjonen, "Host Identity Protocol Certificates," RFC 6253 (Experimental), IETF, May 2011.

[29] T. Henderson and A. Gurtov, "The Host Identity Protocol (HIP) Experiment Report," RFC 6538 (Informational), IETF, Mar. 2012.

[30] J. Rosenberg *et al.*, "SIP: Session Initiation Protocol," RFC 3261 (Proposed Standard), IETF, Jun. 2002.

[31] "Floodlight project," Available: http://www.projectfloodlight.org.

[32] J. Wang, Y. Wang, H. Hu, Q. Sun, H. Shi, and L. Zeng, "Towards a security-enhanced firewall application for openflow networks," in *Cyberspace Safety and Security - 5th International Symposium*, November 2013, pp. 92–103.

[33] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: Building robust firewalls for software-defined networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. ACM, 2014, pp. 97–102.

[34] E. de la Hoz *et al.*, "Detecting and defeating advanced man-in-the-middle attacks against tls," in *Cyber Conflict (CyCon 2014), 2014 6th International Conference On*, June 2014, pp. 209–221.

[35] "Snort manual," Available: http://manual.snort.org.

[36] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P. Strub, "Implementing tls with verified cryptographic security," in *Security and Privacy (SP), 2013 IEEE Symposium on*, May 2013, pp. 445–459.

[37] S. Chaki and A. Datta, "Aspier: An automated framework for verifying security protocol implementations," in *Computer Security Foundations Symposium, 2009. CSF '09. 22nd IEEE*, July 2009, pp. 172–185.

[38] K. Bhargavan, C. Fournet, R. Corin, and E. Zălinescu, "Verified cryptographic implementations for tls," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, pp. 3:1–3:32, Mar. 2012.