Strengthening SDN Security: Protocol Dialecting and Downgrade Attacks

1st Michael Sjoholmsierchio
Naval Postgraduate School
michael.sjoholmsierchio@nps.edu

2nd Britta Hale
Naval Postgraduate School
britta.hale@nps.edu

3rd Daniel Lukaszewski
Naval Postgraduate School
dflukasz@nps.edu

4th Geoffrey Xie
Naval Postgraduate School
xie@nps.edu

2021 IEEE 7th International Conference on Network Softwarization (NetSoft) | 978-1-6654-0522-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/NetSoft51509.2021.9492614

***Abstract*—Software-defined networking (SDN) has become a fundamental technology for data centers and 5G networks. Transport Layer Security (TLS) has been proposed as its single security layer for SDN networks; however, use of TLS is optional and TLS connections between the controller and switches are still vulnerable to downgrade attacks. In this paper, we propose a protocol dialecting approach to provide additional and customizable security assurance for network control messages. We consider and evaluate two dialecting approaches for OpenFlow protocol operation, adding per-message authentication to the SDN control channel that is independent of TLS and provides robustness against downgrade attacks targeting TLS. Furthermore, we measure the performance impact of using these dialecting primitives in a Mininet experiment. The results show a modest increase of communication latency of less than 22%.**

***Index Terms*—Network security, Software-Defined Networks, Protocol Dialect, Transport Layer Security, Defense-in-depth**

I. Introduction

The Transport Layer Security (TLS) protocol is the only recommended security solution for protecting the SDN control channel OpenFlow traffic, and it is only considered on an optional basis [1]. Due to TLS being optional, not all commercial SDN switch vendors and controller platforms provide native support of TLS [1]. Furthermore, TLS is vulnerable to relatively well-known downgrade attacks, which are documented in Common Vulnerabilities and Exposures (CVEs)[1] [2], [3]. If such an attack was to occur between an SDN switch and controller, control channel messages would be vulnerable. For example, an attacker could send flow table updates to a switch to alter routing rules, or alter data statistics sent from a switch to the controller.

As SDN devices become geographically spread, the use of TLS will be more important compared to physically secured data centers and consequently the implications of downgrade attacks on TLS for SDNs gain prominence. There are a variety of downgrade attacks in existence [2], and even the mitigations introduced with TLS 1.3 do not prevent a version downgrade (such as to TLS 1.2) [3].

In this paper, following the defense-in-depth principle, we investigate *a complementary solution using* a novel protocol dialecting approach. Protocol dialecting is an emerging network operation technique that aims to specialize the communication syntax and semantics of a standard networking protocol for the purpose of enhancing network security [4], [5]. The addition of a dialect also requires no action or modification of the protocol standard that it is added to. Modifications to existing protocols to achieve new goals form the roots of protocol dialecting. This practice of experimenting with existing or optional fields of a protocol to signal a desired behavior is not new. The standard TCP protocol has incorporated multiple extensions through the use of the TCP Option field, to include window scaling, fast open, timestamps, and Multipath TCP.

Our contribution novelty lies in a defense-in-depth authentication and downgrade protection while enabling original protocol messages to flow as-is. In summary, we formulate high level criteria and a methodology for designing and evaluating a protocol dialect. We design and evaluate two security derivatives (potential dialecting solutions) that can be used to dialect the OpenFlow protocol. We present the security rationale for the derivatives to achieve another layer of per-message authentication that is independent of TLS, and as such, enhance the defense of some TLS versions against certain attacks. We further present experimental results to show that deploying the derivatives will likely incur a modest increase of communication latency. We simulate a TLS downgrade attack and demonstrate how the designed and implemented dialect could be used to detect and prevent such attacks against a Mininet-emulated SDN environment. Finally, we conceptualize a new form of protocol level policy-based networking by extending the protocol dialecting functionality at a SDN device into a policy enforcement proxy (PEP).

Notably, an operator might also leverage the proxies for optionally defining and enforcing security-oriented policy for specific protocols, e.g., by mandating the use of TLS or permitting only specific TLS versions for the SDN control channel. This allows TLS and the dialect to work in tandem to protect command OpenFlow data. A PEP is used as the implementation method for our experiment; however, protocol dialects are not limited to use of proxies.

II. Related Work

We review related work on three fronts. First, we discuss two efforts that explore a similar approach of system specialization. Then, we describe some related research that aims to improve SDN security using alternative approaches to

[1]CVE-2016-0800, CVE-2018-12404, CVE-2018-19608, CVE-2018-16868, CVE-2018-16889, and CVE-2018-16870.

978-1-6654-0522-5/21/$31.00 ©2021 IEEE

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI. Downloaded on February 12,2022 at 11:47:15 UTC from IEEE Xplore. Restrictions apply.

ours. Finally, we consider other proposals for mitigating TLS cipher suite downgrade attacks.

System Specialization. Broadly speaking, protocol dialecting, like code debloating [6]–[8], can be classified as a system specialization technique. Notably, two recent papers, one on application program interface (API) specialization (Shredder [9]) and the other on side-channel authentication [10], illustrate the utility of system specialization in providing an additional layer of security protection and bolstering a defense-in-depth model.

The Shredder system [9] reduces the attack surface vector for an attacker to perform code reuse attacks [9] by limiting the available function calls of an API (e.g., the binary of a C++ library) to those actually used by applications of an organization. The input parameter vector and value of each parameter for permitted function calls can be further restricted to those combinations used by the same applications. This and similar code debloating efforts [6]–[8] effectively restrict code execution paths and values of variables, based on results from static analysis or dynamic profiling of the code with respect to usage by target applications. In comparison, our work is focused on protocol level semantics, including authenticity of control messages and selection of protocol parameters, and thus, does not require code analysis or profiling.

Electromagnetic side-channel authentication [10] is a similar concept to protocol dialecting because it also aims to add a new layer of authentication to the original communication protocol. This security method relies upon physical layer modifications to add identity features at the source and verification at other end of the communication. Particularly worth noting is its ability to add a unique end-point fingerprint that is operating on a totally separate plane from that of the original protocol. In comparison, the design explored in this paper aims to modify protocol behaviors within the same plane.

SDN Security. Much prior research on SDN security has been focused on verifying the consistency and accuracy of new flow rules at the controller before deploying them to switches (e.g., [11], [12]). Our work is complementary to this line of research by providing additional control channel protection so that (i) rules that have been properly verified will be less likely to be modified without detection in transit before they reach their destination switches, and (ii) network state updates received by the controller are more likely to be authentic and thus, the rule verification performed at the controller is more likely based on accurate network state information.

Interestingly, the ConGuard work [13] shows that an adversary can exploit race conditions among different software threads of a deployed SDN controller to disrupt network operation and even crash the controller, by simply generating a specific sequence of network events from a compromised host, without having to breach any networking device or the control channel. While such attacks and the type of race conditions in controller software are outside the scope of this work, we note that our protocol dialecting proxies can be extended to infer the associated network events from received OpenFlow messages (even in their encrypted form) and consequently, act like stateful firewalls by rejecting messages deemed suspicious.

Mitigation of Cipher Downgrade Attacks. A countermeasure for TLS cipher suite downgrade attacks is discussed in RFC 7507 [14]. RFC 7505 relies on server detection of a `TLS_FALLBACK_SCSV` flag value provided by the client for all fallback ciphersuites. However, this assumption in turn relies on an attacker's inability to tamper with the client-to-server message (the server acts to reject the connection in RFC 7505, vice the client). A MitM attacker forcing weaker authentication can simply remove such flags so the server does not detect them. This makes RFC 7505 inadequate to the threat model.

III. Preliminaries

The concept of protocol dialecting, as well as its use as a network security solution, has been applied to a relatively small number of networking protocols. In this section, we provide a brief overview of the technical area and a formulation for systematically designing and evaluating a protocol dialect.

A. Protocol Dialecting

Protocol dialecting allows an operator to specialize operation of a networking protocol beyond the standard configuration options, on a per-network basis. In today's data network operation, it is common to use open-source implementations of standard protocols such as DNS, HTTPS, and OpenFlow. Therefore, such networks share the same software vulnerabilities and cannot escape from the dreadful reality of "break one [network], break them all". By adding network-specific protocol behaviors, the premise of protocol dialecting is to force potential attackers to spend significantly more resources to decipher specific details of the protocol specialization and to meet customized security needs of an organization. The goal of the particular dialecting approach presented here is to cause an attacker to spend more time and resources compared to networks with only TLS, as well as to enable per-message authentication secrets.

A protocol dialect employs multiple underlying sub-solutions for customizing the operation of a protocol. We define each of the sub-solutions as a dialecting derivative in this paper. Derivatives can take many forms. In reported use cases, some researchers have proposed to change a protocol's messaging semantics (e.g., re-purposing of fields and private options) and timing (e.g., artificial delays, duplicates, or omissions of messages) [4], [5]. Direct modifications of protocol semantics and timing have the additional benefit of amplifying the presence of attack traffic, in the same way that a visitor who doesn't speak the local dialect would stand out when speaking to a roomful of locals. In one of our derivatives, we follow this precedence and require no message modification; in the second derivative we consider modifications to the format and semantics of OpenFlow messages. The first derivative adds security into existing data packets using a normally random packet field, while the second derivative provides security as a wrapper around the entire unmodified packet.

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI. Downloaded on February 12,2022 at 11:47:15 UTC from IEEE Xplore. Restrictions apply.

B. Toward Systematic Development of Protocol Dialects

We observe that two factors are critical to controlling the complexity of dialect implementation. First, most network operators do not have the programming or security background required for safely modifying protocol software, even if the source code is available. Automated tools for creating protocol dialects from original binaries is highly desirable. Alternatively, plug-and-play proxy software with minimum configuration requirements can be deployed to alter protocol behaviors without the need to modify protocol binaries. The use of proxy software reduces the requirement of tailoring to the underlying protocol being modified and can be protocol agnostic. Second, protocol dialects should not adversely affect the effectiveness of current network monitoring tools. Modifications made to the OpenFlow packets or wrappers do not modify the OpenFlow protocol and can be removed prior to analysis by a monitoring tool. The resulting traffic flows should not cause significant manual re-configurations of intrusion detection systems.

From the above observations, we propose these two criteria for properly evaluating the effectiveness of a protocol dialect:

1) **Security benefits**. The protocol dialect must provide net security benefits in that it enables significantly improved security protection without introducing new vulnerabilities that could weaken the security properties of the original protocol implementation.
2) **Deployment costs**. The expertise, time, and resources required of the network manager to deploy the protocol dialect must be at a similar level to that in a typical deployment of the original protocol implementation. Similarly, deployment of the protocol dialect must not significantly increase network communication overhead.

Next, we present a methodology for systematically designing, testing, and deploying a protocol dialect to ensure meeting both criteria. The methodology consists of six distinct stages as illustrated in Fig. 1.

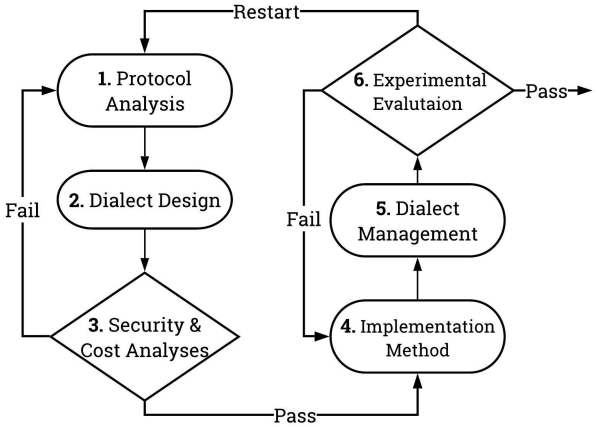

Fig. 1: 6-step Methodology for Designing and Evaluating a Protocol Dialect.

Central to our methodology are two iterative processes specifically designed to mandate that both aforementioned evaluation criteria are considered. The process in the left half of Fig. 1 evaluates the security benefits and deployment cost primarily through formal modeling and analysis while refining assumptions about the targeted deployment scenarios. Ideally, the dialect provides a general solution with sufficient flexibility to work with networks of varying size, topology, and services provided. Assumptions must explicitly, and precisely, characterize resource and time constraints, threat model, and strengths and weaknesses of available security solution building blocks. These assumptions will likely go through several rounds of validation and refinement before they actually match operational reality as demonstrated in Fig. 1.

The iterative process shown in the right half of Fig. 1 requires that the network operator prototype all major dialect deployment and management steps and experimentally evaluate how these steps may impact the performance assurance results derived from the security and cost analyses. In particular, potential impacts on the performance of authorized traffic flows and on the effectiveness of network monitoring tools should be measured and analyzed to guide selection and refinement of dialect implementation and management. Some negative impacts can be overarching and not correctable by implementation or management changes alone; such issues necessitate re-designing some of the dialect's basic elements, i.e., returning to Step 1 of the methodology. An example of an implementation limitation, which requires a step back to design, would be timing requirements of devices that are no longer satisfied due to the addition of deep packet inspection and security checks. In this case, the design required fast functioning security algorithms and packet inspection to not add excessive time delay.

Inspiration for the the dialect design process begins with the selection of a security concern by an organization, leading to the selection of a security mechanism and its associated options. In this stage, one or more derivatives are designed and created for testing. Examples of desirable security objectives include confidentiality and/or authenticity. For this research, we selected authenticity as the primary security objective.

IV. Dialecting OpenFlow

In this section we apply our proposed six-step methodology to design, implement, and evaluate two different OpenFlow dialecting solutions. We refer to these solutions as OpenFlow derivatives for the remainder of the paper. Each derivative provides a different method for including additional layers of security. The methods we select include (i) a wrapper modification and (ii) a direct packet modification within the constraints of the protocol.

In addition to the two high-level criteria introduced in the beginning of Section III-B, we also seek to minimize changes to the normal communication process of the SDN control channel devices in order to minimize additional network management complexity required for deploying the proposed OpenFlow derivatives. To this end, we consider a class of derivatives, which we term inline solutions, that re-purposes existing OpenFlow messages for new security checks. These

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI. Downloaded on February 12,2022 at 11:47:15 UTC from IEEE Xplore. Restrictions apply.

checks are then performed by the inline proxies with the potential for switch or controller code modification as well.

Furthermore, we make two simplifying design assumptions as follows: (i) clocks of SDN devices are always synchronized within 100 milliseconds, and (ii) pre-shared keys are securely deployed *a priori* between the controller and each switch. The clock synchronization can be achieved by deploying the Network Time Protocol (NTP) in the network or attaching GPS hardware to each SDN device. The requirement of pre-shared keys clearly introduces additional network management complexity. However, we observe that it occurs only once for each control channel device pair. Moreover, while pre-shared keys are assumed for simplicity in testing purposes, a suitable key exchange and management could be instituted via another dialecting derivative.

A. Protocol Analysis

Protocol analysis consists of an examination of the standard operation of the base communication protocol (i.e. OpenFlow) for the purpose of creating protocol derivatives. We follow an information security approach in this analysis, identifying baseline adversarial capabilities and protocol goals.

Threat Model. We consider an attacker who can monitor all communications, delay, intercept, replay, and replace arbitrary messages between the controller and switches. We assume that the attacker's purpose is not to instigate denial-of-service attacks, but to breach the SDN control channel. We further assume that the attacker has no means to directly compromise software, storage, and data structures running on the SDN devices without first compromising the control channel.

While this threat model under-estimates the scope and severity of some existing and future attacks against the SDN control channel, it supports a proof-of-concept for protocol dialecting as a promising new direction for protecting SDN networks, while not being over-aiming for an all-encompassing solution. Thus, this work serves as an initial step toward a comprehensive solution based on protocol dialecting.

OpenFlow Message Features and Protocol Patterns. In the next step, while developing in-line solutions, we focus on identifying (i) the prevailing protocol patterns between the controller and a switch and the critical messages used in each pattern, and (ii) header features of these messages with potentially reusable fields. We pay particular attention to the latter; for example, a field that is normally random can allow for modification without requiring changes to the protocol specification. Such fields may be modified, still within the bounds of the OpenFlow standard, to provide packet space for inclusion of authentication tags.

We make two observations regarding the OpenFlow messaging patterns. First, a switch-specific exchange of Hello messages is required for the controller to establish communication with each switch, and another switch-specific periodic exchange of Echo messages is used for maintaining the connection. Given their importance and pervasiveness, these messages are good candidates for dialecting to provide additional security features. Second, when TLS is used, content of

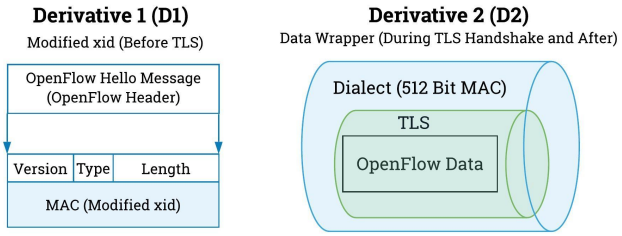

Fig. 2: Illustration of Dialecting of existing OpenFlow Messages to Support the New Derivatives. D2 outer (blue) shading indicates the derivative while the inner (green) shading indicates the TLS wrapper.

OpenFlow messages are meaningful at the receiving end only after the outer TLS encapsulation is properly removed. This implies that if dialecting of OpenFlow messages is carried out before TLS functionality at the sending end, i.e., based on plain-text fields of OpenFlow messages, TLS and/or SDN parts of switch software must be dialected. This offsets the complexity of dialecting kernel-level switch software that may incur significant management complexity in a network deploying switches of different models or vendors. This motivates a wrapper variant of dialecting to operate for every message sent between devices, which does not depend on the OpenFlow message format.

B. Dialect Design

The dialect design stage affects all future stages. For this reason, it is important to consider how the choices made will impact the security and implementation stages, and revisit design choices during those stages. Derivative design choices important to this stage include security objectives, cryptographic algorithms, and key generation and management.

From the protocol analysis stage, we identify two possible derivatives to add message-level authentication that is independent of TLS security features.

Derivative 1 Design. The Derivative 1 (D1) Hello message format is shown in the left part of Fig. 2. D1 modifies data within the bounds of the protocol, but re-purposes the normally random 32-bit Transaction ID (xid) field of the initial OpenFlow Hello exchange as a hash-based message authentication code (HMAC). The HMAC verification is performed by the proxies at both the controller and switch to prevent malicious commands and data from reaching the SDN devices. The use of D1 allows for immediate authentication of packets before TLS is established.

The available 32-bit transaction ID field is well below recommended MAC lengths [15]. This issue is unsurprising when dialecting with re-purposed existing header fields; however, it raises a design question of ensuring security even under a limited bit field that can easily be brute-forced. To accomplish this, we effectively limit the HMAC lifespan by "killing off" keys from memory after a pre-determined interval so that the MAC verification cannot take place after the security window is exceeded.

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI. Downloaded on February 12,2022 at 11:47:15 UTC from IEEE Xplore. Restrictions apply.

For constructing D1, we utilize a Hash-based Key Derivation Function (HKDF) [16] to derive two uni-directional keys from the switch-controller pair-specific pre-shared key and the BLAKE2b [17] hash algorithm for computing HMACs. New keys are then derived sequentially in a ratcheted fashion (the current key is used to derive the next key), with each key having a lifespan of 1 second.

D1 cannot be used indefinitely without potentially interfering with the OpenFlow standard. When the controller or switch exchange multiple messages with the corresponding `more` flag set, follow-up messages are required to utilize the same xid field [1]. Under D1, this field is overwritten for each message and, since the message contents are different, the xid field will contain a different MAC for the subsequent messages. This is not an issue during setup of the initial connection, however, as each message does not use the `more` flag.

Derivative 2 Design. While D1 is native to the OpenFlow protocol, it only adds protection to the initial communication between the controller and switch for the reasons stated above. Conceivably, we could expand the protection while using a similar strategy to dialect Echo messages. However, with Derivative 2 (D2), we chose to develop a more comprehensive solution covering all OpenFlow messages. There is no limitation on using both D1 and D2 to reduce initial communication forgeries and per-packet protection following the initial OpenFlow Hello. Furthermore, D2 is designed to allow inter-operation with TLS.

To avoid management complexity associated with dialecting kernel level switch software, we define D2 as message wrapper as follows. A 512-bit HMAC tag is appended to encrypted data packet before sending, as illustrated in the right part of Fig. 2. The communicating ends accept an OpenFlow message as valid only if it passes the HMAC check. D2 is protocol agnostic and can be added to a variety of protocols (e.g. various versions of TLS or when TLS is not enabled) because it does not manipulate any internal fields or data in the packets.

C. Security and Cost Analysis

The security and cost analysis stage is an intermediate analysis step to ensure that these objectives have been met. An organization's requirements determine if the added security is sufficient, as well as the impact on overhead and system delay. If a given organization's security requirements and cost constraints are not met during this intermediate analysis, then the dialect designer must return to stage 1. Otherwise, the designer may continue in the process to determine the implementation method.

We defer the analysis of our design choices to its own section (Section V).

D. Implementation Method

We observe that there are important considerations in selecting an implementation method for our OpenFlow derivatives. One option we considered was to borrow from code debloating work [6]–[8] and extend some of the existing automated tools that manipulate binaries directly to support adding new functionality to existing OpenFlow software. This approach looks attractive at first because it would enable rapid deployment of derivatives without requiring much code development effort by the operator. However, upon closer examination, we found major limitations with it. The tools operate at granularity of loops and function calls, thus lacking the ability to rapidly prototype, enforce high-level security policies, and be compatible with a variety of protocols.

Deployment via Policy Enforcement Proxies. Based on these observations, and to enforce high-level security policies such as select versions of TLS or a certain ciphersuite, and to speed up evaluation of the derivatives, we choose to create a new software component to function as a policy enforcement proxy (PEP) in front of the controller and each switch. This method is compatible with Mininet's approach of using virtualized devices for testing and measurement and allows for all testing to occur with one physical machine.

The PEP implementation with D1 and D2 functionality is shown in Fig. 3. A pair of proxies sit between each switch and controller, and consequently do not have access to the TLS keys used by the switch or controller. For each OpenFlow message, the proxy assigned to the sending device intercepts and dialects messages, without decrypting them, as shown in Fig. 2. The proxy for the receiving device verifies the derivative and performs other checks as required by the security policy. We see other possibilities for security policy rules, e.g., rate limiting of flow requests and link state updates from a switch as a countermeasure to denial-of-service attacks. We also expect that the PEPs can be extended to enforce and regulate other protocols and features besides OpenFlow and TLS, as discussed in Section VII.

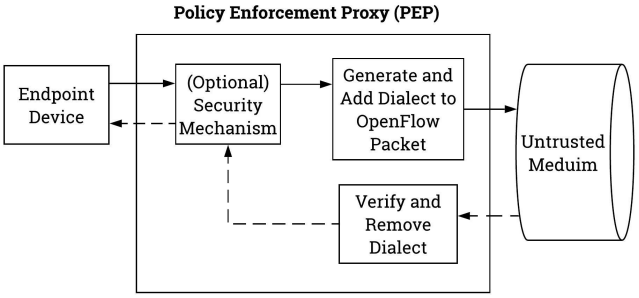

Fig. 3: PEP Design. This figure demonstrates the logical setup of the proxy and how inbound (dashed) and outbound (solid) data is managed between functions. An optional security mechanism may also be implemented at the PEP (e.g. connection management, ciphersuite selection enforcement, etc.).

E. Dialect Management

This stage involves the support systems and methods that will allow for PEPs to operate long-term and in day-to-day operations. Attributes important to dialect management include ease of management, installation difficulty, and scalability.

To maximize security, design of OpenFlow derivatives should require an independent key management system. In this

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI. Downloaded on February 12,2022 at 11:47:15 UTC from IEEE Xplore. Restrictions apply.

work, we assume that TLS runs between the switch and controller, while D2 is proxy-to-proxy. This separation means that compromise of an end device does not necessarily imply loss of derivative keys (on the proxy), or vice-versa. Conceivably, to reduce deployment costs, the operator may choose to reuse the same physical infrastructure as for storing TLS keys. In such a scenario, we recommend to minimally use virtualization techniques to establish a logical partition between the different key computing, storage, and communication resources used for the derivative and TLS. Even with such precautions, it should be noted that security of reuse of such a physical infrastructure for both the derivative and TLS is dependent on the absence of compromise of the end devices.

F. Implementation Testing

Implementation is the last stage of our protocol dialect process. Implementation testing consists of testing security properties and the availability of the system or protocol with the included dialect. Before this testing can occur, control tests of normal operation are performed and later used as a testing baseline following dialect integration. The communication latency and other types of overhead added due to the addition of a dialect are then measured and calculated to ensure that availability of the system meets the organization's requirements. For clarity of presentation, we defer the details of our testing efforts to Section VI.

V. Security Basis

In this section we take a closer look at the security of D1 and D2, based on the design choices discussed in Section IV. We did not conduct a formal security proof of the composition of D1 and D2, leaving that to future work, but provide justification arguments for the design security through comparison to prior work.

A. Formalizing Design Choices

In the following discussion, we assume both D1 and D2 are performed in a sequential format. Prior to implementation of D1 and D2, a pre-shared symmetric key, generated at random, is established and delivered out-of-band (OOB).[2] It is assumed that keys used between a controller and switch are not reused for any other controller/switch pair. Following the initial key distribution, each side generates uni-directional keys (UDK) for D1 and D2 according to Fig. 4 and Fig. 5, with the following terms:

- $\mathtt{K_{ab}}$: symmetric key used to protect data from a to b
- `KDF`: Key Derivation Function
- $\mathtt{TS_a}$: a timestamp at party `a`
- `sid`: a session identifier which we calculate as $\mathtt{sid} = \mathtt{D1\_MAC\_Switch||D1\_MAC\_Controller}$
- $\mathtt{SN_a}$: a sequence number for party `a`

As seen in Fig. 5, the proxies will initially exchange D1 messages potentially non-interactively (the OpenFlow protocol does not mandate an order to the Hello messages) [1]. The freshness value incorporated in our design is based on implicit timestamps (rounded to seconds). This has implied assumptions that the controller and switch have the same clock time, which is reasonable with embedded hardware clocks. Although such synchronization constitutes a potential point of failure, the timestamp could be computed based on the completion of the TCP handshake to limit clock de-synchronization.

[2]This may occur as an OOB key distribution. However, there is the potential to embed a full key exchange in a prior run of D2 as discussed in Section VII.

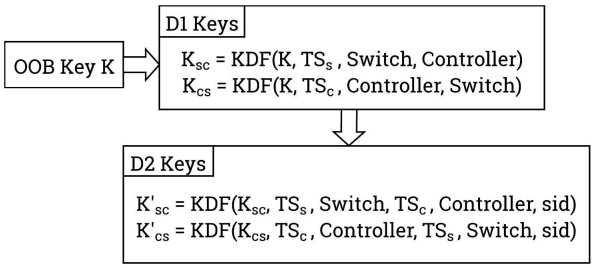

Fig. 4: D1, D2 Key Derivation.

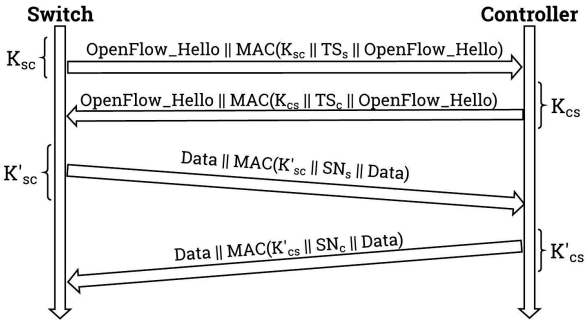

Fig. 5: D1, D2 Key Usage. Horizontal flows indicate (potentially) non-interactive messages such that ordering is not enforced. Angled flows indicate interactive messages with enforced ordering.

Packets arriving during the same second should pass D1 MAC verification, while those arriving outside of the time window will fail verification. Note that we do not employ timestamp values simply to inhibit replay attacks; as noted earlier, the MAC length is limited in D1 by the 32-bit header field and as a result is liable to forgery attacks. Consequently, it is necessary to limit the viable lifespan of the MAC, which is performed via the key lifespan window. Any packet received outside of this window will have a different key used in the D1 MAC calculation step. Therefore, if an attacker attempts to replay any messages, they must be replayed within the same time window in order to pass the MAC verification step. Our calculations to justify selection of a 1 second key lifespan are based on average collision resistance of $2^{32\ \mathrm{bits}/2}$ and an average Python test time to create a 32-bit MAC of 10^{-4} seconds. This yields an average of 6.5 seconds necessary to find a collision.

We use the HKDF and BLAKE2b algorithms for key derivation and MAC computation for both D1 and D2, with two differences. First, in D2, the HMAC length is increased to 512 bits. The length can be further increased based on network

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI. Downloaded on February 12,2022 at 11:47:15 UTC from IEEE Xplore. Restrictions apply.

security requirements. Second, under the extended MAC tag length we no longer bound the key lifetime, but still require per-message keys as well as unique keys for each sender.

We use a unidirectional sequence number (i.e. one sequence number per sender, incremented per message) as part of the hash input to combat replay attacks. For instance, once the TLS handshake has completed, the switch and controller initialize independent (derivative) sequence numbers. These numbers are included in the HKDF function and incremented after each message. In the event an attacker attempts to replay a previous message, the receiving side will attempt verification using the correct sequence number and detect an error, resulting in a dropped packet. In our experiment we set the receiver to drop the packet and connection completely if an error associated with a validation code was detected. Since re-establishing the switch-controller connection requires relatively little time, we select to recreate the setup of the switch versus simply dropping such error packets and keeping the connection.

Note that we use the transcript D1 MAC values of the initial flows to provide continuity to the session over D1 and D2, essentially employing these as a session identifier. This links protection of the initial D1 security layer to later layers.

B. D1 Security

The security goal of D1 is to establish that the proxies involved possess the correct keys and perform MAC verification of the OpenFlow messages. Thus D1 acts, in part, as a mu authentication protocol with associated data. We reference | for in-depth analysis methods which may be applied to s composed protocols with associated data. An attacker's goa breaking entity authentication during D1 is largely depenc on breaking the security of the MAC, necessitating the us a short MAC lifespan through validity windows, due to MAC output length used during D1.

As a justification of the authentication properties of D1, we draw to comparison with a previously analyzed protocol, the repaired ISO/IEC 9798-4:1999 Mechanism 5.2.1 two-pass mutual authentication protocol (referred to as ISO 9798-4-3 by Cremers et al.) [19], [20]. The comparison is shown in Fig. 6. We specifically note and justify the following.

a) Uni-Directional Keys (UDK): Cremers et al. [19] incorporate identities into the check function of ISO/IEC 9798-4:1999 Mechanism 5.2.1 to show direction of flows, hence avoiding reflection attacks arising from use of a single symmetric key. D1 embeds this in the derivation process of the UDKs. Each key derivation contains identifiers for the sender and the receiver. This prevents messages sent by the switch to be accepted by the switch, e.g. a reflection attack. Furthermore, the control plane of an SDN limits communication from a switch only to the controller (switches will not communicate with other switches). When UDKs are used, such as described above, the identities are no longer a necessary input to the cryptographic check function [20].

b) Text Fields: $Text_x$ values in Fig. 6 correspond to the OpenFlow Hello messages sent by D1. Unlike in the original ISO/IEC 9798-4:1999 Mechanism 5.2.1, we do not allow for variable selection of data for these fields: it is pre-determined. Consequently, we also avoid the associated potential protocol flow syntax errors [19].

c) Timestamps: The repaired ISO/IEC 9798-4:1999 Mechanism 5.2.1 explicitly sends timestamps (TS_x) which it uses as inputs to the MAC. D1 uses a form of implicit timestamps, which are used in key derivation for the MAC vs. input into the protocol data fields. By using implicit timestamps, D1 conforms as closely as possible to the OpenFlow standard. Moreover, implicit use is enabled as the switch and controller have shared knowledge of the timestamps.

d) Flow Ordering: The repaired ISO/IEC 9798-4:1999 Mechanism 5.2.1 protocol [19] uses a flow tagging value to ensure messages are correctly interpreted according to their order in the protocol, e.g. "`Flow1`" and "`Flow2`". We require that D1 proxies enforce ordering, where the protocol flow generated by the switch will always be interpreted as the first flow, while the protocol flow from the controller will be held to be the second.

With consideration of the above points, we map D1 to the repaired ISO/IEC 9798-4:1999 Mechanism 5.2.1, which achieves mutual authentication of party A and B [19]. Thus, D1 acts as a pre-shared key mutual entity authentication protocol for the subsequent D2 derivative in addition to providing authentication of the additional data fields, namely the OpenFlow Hello messages.

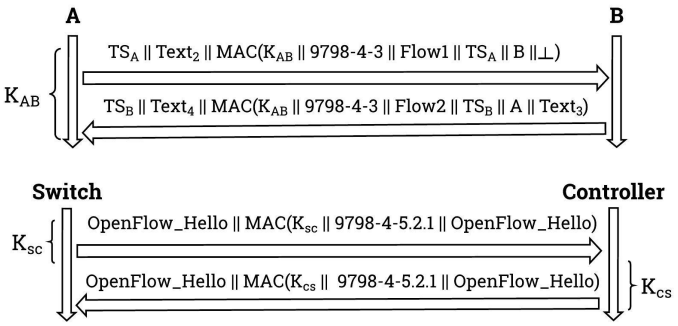

Fig. 6: Alignment of the repaired ISO/IEC 9798-4:1999 Mechanism 5.2.1 protocol (top figure) using a MAC check function to D1 (bottom figure). $\mathtt{TS_S}$ (resp. $\mathtt{TS_C}$) is incorporated into the derivation of $\mathtt{K_{sc}}$ (resp. $\mathtt{K_{cs}}$). $\mathtt{K_{sc}}$ and $\mathtt{K_{cs}}$ also address the directional separation.

C. D2 Security

The PEP logic requires that D1 is completed before starting D2, and therefore mutual authentication has already taken place. On the completion of D1, the switch and controller proxies create UDKs used for D2. The security goal of D2 is to provide an authentication layer for the remainder of the controller/switch session. In this case, the authenticated "data" is in fact the actual OpenFlow packet. Keys for this derivative are derived as presented earlier and shown in Fig. 4. In particular, D2 keys are derived from those of D1 and are computed over the transcript of D1 messages sent, thereby

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI. Downloaded on February 12,2022 at 11:47:15 UTC from IEEE Xplore. Restrictions apply.

binding D2 to the earlier entity authentication and ensuring both parties commit to the transcript of D1.

D2 makes use of a 512-bit MAC that allows for a longer MAC lifetime than D1. Additionally, D2 incorporates sequence numbers into the MAC calculation step as seen in Fig. 5, which increment for each new packet sent under a given key. Sequence numbers are thus uni-directional to each sender. Notably, this limits the potential for replay of messages by an attacker, which is of particular concern in an SDN due to the controller sending configuration commands to the switch. If an attacker was able to replay an old configuration command, then the switch could be induced to start routing traffic improperly or the update may result in erratic network behavior.

VI. Experimental Evaluation

In this section we present results from our experimentation on the D1 and D2 derivatives in a testbed built with the Mininet SDN emulation software. The testbed topology with the PEPs is shown in Fig. 7.

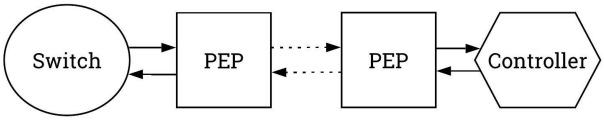

Fig. 7: Mininet Testbed Typology. Solid lines indicate unaltered messages. Dotted lines indicate dialected messages.

We maintain a minimalist topology for testing purposes. Since TLS is point-to-point and the dialecting operation behaves in parallel for each TLS connection, the measured communication overhead relative to TLS can be considered estimates per switch-controller connection for larger, more complex networks that deploy multiple controllers. Controller platforms such as ONOS also have built-in support for (near) real-time state synchronization among multiple controller instances. Such support can help ensure the PEP state is properly migrated when a switch migrates from one controller instance to another, if the dialect security requirements allow state sharing among controllers.

A. Mitigation of Downgrade Attacks

First, we perform a simulation of an attack scenario, validation that the operation of the derivative against a TLS version downgrade attack. We assume that an attacker can modify packets at will between the controller and the switch, but only between the PEPs. Therefore, it is the responsibility of the receiving proxy to enforce that the modified packet by the attacker does not make it to the end point device.

In this experiment we modify the data in the first message of the TLS handshake (i.e., the switch's ClientHello) from TLS version 1.2 to 1.0 before the message is delivered to the PEP for the controller. Two versions of the attack were carried out. In the first, the TLS version number was modified without changing to the MAC tag. In the second, the TLS version number was modified while also changing the MAC tag (i.e. to simulate the attacker "guessing" the MAC). In both cases, the receiving PEP rejected the attack packet by flagging the wrong D2 MAC value. The PEP then sent an alert to the controller so that the controller could terminate the TLS handshake.

It is straightforward to add logic to the PEPs to ensure that the ciphersuite advertised by each party conforms to IETF guidelines [14], [21]. For brevity, we omit such experiments.

B. Estimation of Communication Overhead

After verifying that the derivatives can be used to stop a TLS downgrade attack, we conduct a series of experiments to quantify the communication overhead from using the D1 and D2 derivatives. Baseline and TLS experiments with and without utilizing the PEPs were performed. Specifically, we conducted a total of 120 tests (30 each) for the four different experiment configurations.

Delays with and without TLS were measured from the first OpenFlow Hello to the first OpenFlow Echo Request message. This measurement was taken because the OpenFlow Echo Request message signifies that all setup actions have completed and the SDN devices are switching to a liveliness check state [1]. To perform control testing with TLS required a generic setup and the establishment of certificates as specified by the Mininet project [22]. Additionally, enabling TLS with Wireshark also required the use of the preinstalled ovs-testcontroller as well as an OVSSwitch.

A summary of the average connection time measurements and the average percentages of latency increase due to TLS and the derivatives are shown in Table I. The overhead added by the derivatives without using TLS was minimal. Using TLS alone significantly increased the connection latency (by 87%) since the TLS connection introduces additional flows for the handshake. Then, deploying the derivatives on top of TLS further increased the latency (by 22%). This larger overhead increase (compared to the the case without using TLS) can likely be attributed to the larger packet sizes of TLS and corresponding MAC processing. Still, we consider the connection latency increase over TLS to be moderate.

TABLE I: Summary of Average Connection Latency

TLS Enabled	**Using D1 & D2**	**Average Latency**	**Overhead**
No	No	5.00 sec.	Baseline
No	Yes	5.01 sec.	0.0%
Yes	No	9.33 sec.	New Baseline (87% beyond no security)
Yes	Yes	11.37 sec.	22% beyond TLS (127% beyond no security)

C. Reproducing the Experiments

The virtual machine (password: "default"), scripts, and packages used to perform all experiments can be found open-source at https://tinyurl.com/yxdrp8uh.

VII. Discussion

Limitations One could query on the security-cost trade-off of the D1 and D2 derivatives. After all, if TLS requires a key exchange for each new connection between a given switch and

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI. Downloaded on February 12,2022 at 11:47:15 UTC from IEEE Xplore. Restrictions apply.

controller, are we not doubling the set-up efficiency cost. Is it not simply possible to upgrade to a newer version of TLS? In answer to this, we highlight the point-to-point and defense-in-depth approach in our solution.

a) Initial Set-Up Cost: Once an initial pre-shared key is obtained, a new D1 key is not required for subsequent sessions, but rather can be derived from the last D2 key in use. Thus the derivative keys are consistently ratcheted forward over time. The first D1 key in use could potentially be derived for a TLS pre-shared key. In this case, the first controller-switch communication would be protected only by TLS, while in subsequent sessions D1 would protect the TLS handshake. Alternative methods for obtaining a pre-shared D1 key are also possible, allowing for a great deal of flexibility.

b) Reliance on TLS: Updating all SDN devices to, say, TLS 1.3, may not be feasible dependent on network agility. Once a protocol is implemented, especially over a wide, distributed network, small modifications to data packets (such as in D1) may be significantly more viable than a syndicated upgrade. Moreover, depending on organizational need, TLS may not even be the ideal underlying protocol.

Extensions We note some natural extensions of this work:

1) A General Security Framework Based on PEPs: We observe that protocol dialecting is particularly suited for identifying attackers in a zero trust model network that assumes all devices may be malicious, even hosts inside the enterprise firewall [23], [24]. Compared to an intrusion detection system that relies on offline deep packet inspection, PEPs can enforce fine grain protocol specific security policy on live traffic, similar to an intrusion prevention system. We argue that PEPs can be more flexible than the current device-centric solutions; they allow for separation of the security policy definition or enforcement from other detailed device configuration tasks, and also support rapid deployment of new security policies.

2) New Derivative for Key Management: Our design of the two dialecting derivatives (D1 and D2) assumes that each switch side PEP possesses a pre-shared secret with the controller side PEP. To allow dynamic replacement of the shared secret, we envision an option of a third derivative (D3) that introduces a new type of OpenFlow Experimenter message [1] to carry out key management functions between two communicating PEPs.

VIII. Conclusion

In this research, we have shown it is feasible to leverage protocol dialecting to add another layer of per-message authentication into the OpenFlow protocol that is independent of TLS. We view protocol dialecting as a general and effective solution for deploying and enforcing enterprise security policy on a per-protocol basis, and one that also supports the industry-led movement toward zero trust access control models.

Acknowledgements

We would like to thank anonymous reviewers for their helpful comments. This work is partially funded by the Office of Naval Research (ONR) under the TPCP program.

References

[1] Open Networking Foundation, "OpenFlow Switch Specification Version 1.5.1 (Protocol Version 0x06)," march 26, 2015.

[2] E. S. Alashwali and K. Rasmussen, "What's in a downgrade? a taxonomy of downgrade attacks in the tls protocol and application protocols using tls," in Security and Privacy in Communication Networks, R. Beyah, B. Chang, Y. Li, and S. Zhu, Eds. Cham: Springer International Publishing, 2018, pp. 468–487.

[3] NCC Group Research, "Downgrade Attack on TLS 1.3 and Vulnerabilities in Major TLS Libraries," https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2019/february/downgrade-attack-on-tls-1.3-and-vulnerabilities-in-major-tls-libraries/. Last accessed: Nov-2019.

[4] D. Xu, "Towards internets of dialect-speaking things," Project Update, TPCP PI Meeting, Jun. 2019.

[5] M. Mao, "Security assurance via protocol customization: Novel program analysis and ML-based automation," Project Update, TPCP PI Meeting, Jun. 2019.

[6] H. Sharif et. al, "TRIMMER: Application specialization for code debloating," in Proc. 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018.

[7] A. Quach, A. Prakash, and L. Yan, "Debloating software through piece-wise compilation," in Proc. 27th USENIX Security Symposium, 2018.

[8] H. Koo, S. Ghavamnia, and M. Polychronakis, "Configuration-driven software debloating," in Proc. 12th European Workshop on Systems Security, 2019.

[9] S. Mishra and M. Polychronakis, "Shredder: Breaking Exploits through API specialization," in Proc. 34th Annual Computer Security Applications Conference, 2018.

[10] J. B. Perazzone et. al, "Cryptographic side-channel signaling and authentication via fingerprint embedding," IEEE IEEE Trans. on Information Forensics and Security, vol. 13, no. 9, pp. 2216–2225, 2018.

[11] A. Khurshid et. al, "VeriFlow: Verifying Network-Wide Invariants in Real Time," in Proc. 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13), 2013.

[12] M. Reitblatt et. al, "Abstractions for network update," in Proc. ACM SIGCOMM Conference, 2012.

[13] L. Xu et. al, "Attacking the brain: Races in the SDN control plane," in Proc. 26th USENIX Security Symposium, 2017.

[14] B. Moeller et al., "TLS Fallback Signaling Cipher Suite Value (SCSV) for preventing protocol downgrade attacks," Internet Requests for Comments, RFC 7507, Apr. 2015.

[15] Q. Dang, R. M. Blank, C. F. E. Barker et al., "Nist special publication 800-107 revision 1 recommendation for applications using approved hash algorithms," 2012.

[16] H. Krawczyk et al., "Hmac-based extract-and-expand key derivation function (HKDF)," Internet Requests for Comments, RFC 5869, May 2010.

[17] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, smaller, fast as MD5," in International Conference on Applied Cryptography and Network Security. Springer, 2013, pp. 119–135.

[18] P. Rogaway and T. Stegers, "Authentication without elision: Partially specified protocols, associated data, and cryptographic models described by code," in 2009 22nd IEEE Computer Security Foundations Symposium. IEEE, 2009, pp. 26–39.

[19] D. Basin, C. Cremers, and S. Meier, "Provably repairing the iso/iec 9798 standard for entity authentication 1," Journal of Computer Security, vol. 21, no. 6, pp. 817–846, 2013.

[20] "Information Technology - Security Techniques - Entity Authentication - Part 4: Mechanisms using a cryptographic check function," International Organization for Standardization, Geneva, Switzerland, Standard, Jul. 2012.

[21] Y. Sheffer et al., "Recommendations for secure use of transport layer security (TLS) and datagram transport layer security (DTLS)," Internet Requests for Comments, RFC 7525, May 2015.

[22] "SSL on Open vSwitch and ovs controller," https://github.com/mininet/mininet/wiki/SSL-on-Open-vSwitch-and-ovs-controller. Last accessed: June 2019.

[23] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," National Institute of Standards, NIST Special Publication SP-800, 2020.

[24] I. Z. T. P. Team, "Zero trust cybersecurity, current trends," American Council for Technology, Tech. Rep., April 2019. [Online]. Available:

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI. Downloaded on February 12,2022 at 11:47:15 UTC from IEEE Xplore. Restrictions apply.