MLSNet: A Policy Complying Multilevel Security Framework for Software Defined Networking

Stefan Achleitner, Quinn Burke, *Student Member, IEEE*, Patrick McDaniel, *Fellow, IEEE*, Trent Jaeger, *Member, IEEE*, Thomas La Porta, *Fellow, IEEE*, and Srikanth Krishnamurthy, *Fellow, IEEE*

***Abstract*—Ensuring that information flowing through a network is secure from manipulation and eavesdropping by unauthorized parties is an important task for network administrators. Many cyber attacks rely on a lack of network-level information flow controls to successfully compromise a victim network. Once an adversary exploits an initial entry point, they can eavesdrop and move laterally within the network (e.g., scan and penetrate internal nodes) to further their malicious goals. In this article, we propose a novel multilevel security (MLS) framework to enforce a secure inter-node information flow policy within the network and therein vastly reduce the attack surface available to an adversary who has penetrated it. In contrast to prior work on multilevel security in computer networks which relied on enforcing the policy at network endpoints, we leverage the centralization of software-defined networks (SDNs) by moving the task to the controller and providing this service transparently to all network nodes. Our framework, *MLSNet*, formalizes the generation of a policy compliant network configuration (i.e., set of flow rules on the SDN switches) as network optimization problems, with the objectives of (1) maximizing the number of flows satisfying all security constraints and (2) minimizing the security cost of routing any remaining flows to guarantee availability. We demonstrate that MLSNet can securely and efficiently route flows that satisfy the security constraints and route the remaining flows with a minimal security cost (e.g., route >85% of flows, where the heuristic achieves 89% and 87% of the optimal solutions for the optimization problems).**

***Index Terms*—Software-defined networks, security services, security management.**

Manuscript received May 1, 2020; revised October 23, 2020 and December 10, 2020; accepted December 11, 2020. Date of publication December 21, 2020; date of current version March 11, 2021. This research was sponsored by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). This work was also supported by the National Science Foundation under award CNS-1946022. The associate editor coordinating the review of this article and approving it for publication was J.-M. Kang. *(Corresponding author: Quinn Burke.)*

Stefan Achleitner was with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802 USA. He is now with Security Research Department, Palo Alto Networks, Inc., Santa Clara, CA 95054 USA (e-mail: stefan@stefanachleitner.com).

Quinn Burke, Patrick McDaniel, Trent Jaeger, and Thomas La Porta are with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802 USA (e-mail: qkb5007@psu.edu; mcdaniel@cse.psu.edu; trj1@psu.edu; tfl12@psu.edu).

Srikanth Krishnamurthy is with the Department of Computer Science and Engineering, University of California Riverside, Riverside, CA 92521 USA (e-mail: krish@cs.ucr.edu).

Digital Object Identifier 10.1109/TNSM.2020.3045998

I. Introduction

ENSURING that information flowing through a network is secure from manipulation and eavesdropping by unauthorized parties is an important task for network administrators. Many attacks against modern networks rely on a lack of network-level information flow controls to infiltrate an organizational network. Here, adversaries initially subvert edge defenses to target and compromise an internal node. Once inside the network, the adversary can probe network nodes or eavesdrop on flows to penetrate further into the network [1]. This adversary-enabling freedom of movement and lack of secure routing (to prevent eavesdropping) can be cast as a classical *information flow* problem in security [2].

Even with defenses such as firewalls, information flow control in networks often fails: configuration is error-prone [3], and compromised internal hosts may initiate flows that never have to cross a firewall boundary [4]. Thus, adversaries can exploit firewall rule conflicts to exfiltrate information, and internal adversaries can eavesdrop and move laterally (i.e., scan and penetrate internal nodes) within their network boundary without restriction. Fundamentally, they are enabled by a lack of security policy governing what flows are permitted and what paths they may take in the network.

Multilevel security (MLS) provides the means to enforce such a policy. A multilevel security framework controls information flow among entities of different security classes with security labels (i.e., levels and categories) assigned to those entities. In fact, multilevel security already plays a critical role in controlling access to information for both military personnel and employees of commercial businesses with different levels of clearance [5]. Common use cases include controlling file access in an operating system [6] or table access in a relational database [7]. The notion of multilevel security can also be applied to computer networks, where the MLS policy dictates which nodes are allowed to communicate, what type of traffic they may exchange, and what paths the flows may take in the network. This strategy can prevent eavesdropping and unrestricted lateral movement that plague modern networks.

Lu and Sundareshan [8] envisioned such a model that enforces the information flow policy at network endpoints; however, the scale and dynamic behavior of modern networks make deploying such an enforcement mechanism on every endpoint impractical. Despite this, the inherent centralization

of software-defined networks (SDNs) allows enforcement of a network-level MLS policy in a scalable and efficient manner. Determination of whether or not flows are permitted can be done by the controller, and the access-control policy can be enforced by the data-plane switches in the form of flow rules—which allows the service to be provided transparently to the entire network.

Thus, in this article, we propose an SDN-based MLS framework to enforce an inter-node information flow policy that preserves confidentiality. The challenge here is to fit the organizational needs by allowing entities to exchange permitted flows while also configuring the network (by leveraging flow rules) to be policy compliant. Permitted flows between two endpoints may not always find a secure path due to limited network resources (e.g., link capacity). Then, to guarantee availability, a flow may have to be routed through an insecure path. We refer to such a situation as a *policy conflict*, and each conflict imposes a *security cost* in terms of the risk the flow is being exposed to.

Unlike prior work [8], we approach the challenge of securing information flow in the network by considering two optimization models: one that can provide a secure network configuration (i.e., composition of flows rules) that obeys the security policy, supplemented by a model that can minimally relax the security policy to ensure that every flow can be routed. The key contributions are:

- An optimization model to maximize the number of flows routed according to the given security policy in an SDN.
- An optimization model to minimize the security cost of routing any remaining flows to guarantee availability.
- A method for constructing flow rules which adhere to a given security policy.
- A comprehensive evaluation of MLSNet's ability to generate policy compliant network configurations and resolve policy conflicts in realistic network topologies.

II. Definitions and Background

In this section, we extend prior work's [8] terms and notations (Table I) to an SDN setting and provide background on related network security threats and defenses, and MLS.

A. Term Definitions

Node: A resource connected to a network (e.g., a user, server, router, or SDN switch).

Subject: A node that initiates communication to other nodes in the network.

Object: A node that either provides (*provider*) and/or receives (*receiver*) information to/from subjects

Forwarding Node: A node (SDN switch) that processes incoming flows according to the installed flow rules.

Controller: An application in the SDN control plane that has a global view of the topology and installs flow rules to forwarding nodes based on the security policy.

Security Levels: Hierarchical attributes (e.g., top-secret, public) that indicate relative authorization power.

Security Categories: Non-hierarchical attributes (e.g., TCP, IP) that offer finer-grained authorization (for any layer of the network stack) besides the security level. In MLSNet, security categories are only assigned to objects and subjects but not forwarding nodes.

Security Label: The security level and categories combined, used by the controller to admit or deny flows.

TABLE I
Nomenclature and Notation

Notation	Description
V	Set of vertices in network graph $G = \{V, E\}$
E	Set of edges in network graph $G = \{V, E\}$
F	Set of packet flows to be accommodated
R	Set of matching fields in a flow rule
A	Set of action fields in a flow rule
S	Set of subjects
O	Set of objects
C	Set of security categories
d^f	Size of flow $f \in F$
κ_{ij} $(\widetilde{\kappa}_{ij})$	Residual capacity of link (i, j), $(i, j) \in E$
σ_i	Security level of node i, $i \in V$
λ_i^c	Security category c at node i
L	Set of labels that form the lattice

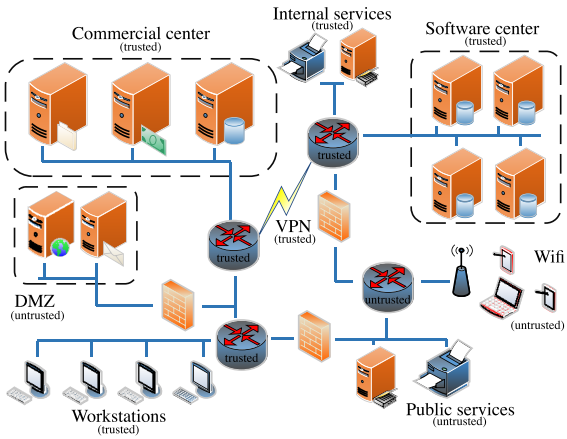


Fig. 1. A corporate network scenario, detailing different regions of a network that may contain nodes (switches and hosts) of different security levels.

B. Network Threats to Confidentiality

Confidentiality ensures that information is only being accessed by authorized parties. In the context of networking, preserving confidentiality means that only explicitly allowed communication can flow between any two nodes in the network to prevent data from falling into the hands of untrusted entities. A lack of a formal communication policy to realize this may allow adversaries who have compromised internal nodes to explore the network or eavesdrop on flows. For example, in Figure 1, a compromised trusted node in the software center may be able to probe other nodes in the software and commercial centers as they are all behind the same firewall boundary; the firewall itself cannot prevent the adversary from probing all nodes on TCP port 22. Indeed, this is possible regardless of the (implicit) security level of the nodes; however, we can reduce attacker capabilities (enforce least-privilege) with multilevel security.

TABLE II
DEFENSES AGAINST DISCUSSED ATTACK TECHNIQUES

	MLSNet	[18]	[4]	[20]
Packet spoofing	×	×	-	-
Lateral movement	×	-	×	-
Man-in-the-middle	×	-	-	×

Recent work has demonstrated the ability of an adversary to freely probe within their network boundary to recover sensitive information about the network [9], [10], [11], [12], [13], including active network hosts and even switch flow table rules [14]. We observe that although discovered attacks on networks in the literature pursue different goals, the strategies of those posing a threat to confidentiality can be reduced to a small set of techniques.

Packet Spoofing: Spoofing packets is the most common technique. By spoofing, adversaries may be able to impersonate other nodes to escalate privilege [15] or leak information to untrustworthy nodes or outside of the network [16].

Lateral Movement: Adversaries can also move laterally by probing many other nodes in the network. This nonessential communication may allow them to extract sensitive information from nodes of higher security levels or compromise nodes and escalate privilege to move deeper into the network [14].

Man-in-the-Middle: Adversaries can also position themselves as a man-in-the-middle (MiTM), silently eavesdropping on communications traversing them or within their broadcast domain [17].

C. Proposed Defenses

Although there have been defenses proposed against some of the discussed attack techniques, they are limited in their ability to preserve confidentiality.

Source Validation: To address the issue of packet spoofing, source validation seeks to only permit packets into a network which's source IP is valid on the given network interface. This is typically implemented via ingress filtering [18] in wide-area networks; however, it is difficult to implement, especially in data-center networks [19], and does not prevent adversaries from spoofing nodes within their own subnetwork.

Firewalling: The primary purpose of a firewall is to prevent unauthorized packets from entering a network or subnetwork. However, firewalling is limited with respect to preventing lateral movement as configuration is error-prone [3], and compromised internal hosts can still probe within their network boundary [4] to compromise internal nodes.

Encryption: Active man-in-the-middle attacks (i.e., those staged by spoofing) may be mitigated with source validation; however, preventing passive MiTM (eavesdroppers) is difficult. Even with services such as encryption, adversaries can still perform traffic analysis to extract sensitive information [20].

D. Preserving Confidentiality With Multilevel Security

Broadly speaking, existing defenses solve distinct problems and only partially address the issue of confidentiality. Adversaries are enabled by a lack of policy preventing them from probing network nodes and eavesdropping on communications. What is needed are means to specify what flows are permitted and what paths they may take in the network.

MLS: A multi-level security policy provides the means to prevent these problems with a secure flow model between entities that are assigned specific security labels (i.e., a level and categories). The security labels form a lattice structure, which reflects a hierarchical ordering of their relative authorization power. We consider a node's label to be higher than another node's if the former's level is greater than or equal to, and the categories form a superset of, the latter's. With respect to confidentiality, information should only flow to nodes with the same or higher security label to prevent the potential leakage of sensitive data to nodes of lower security labels. This is typically summarized as *"no read up, no write down"*.

Network MLS: Multilevel security already plays a critical role in controlling access to files and databases in military and commercial business contexts [5], [6], [7]. This notion can also be applied to computer networks to prevent the eavesdropping and unrestricted lateral movement that plague modern networks. For example, nodes with lower security levels should not be able to probe or communicate with nodes of higher levels on specific TCP ports, and sensitive (e.g., top-secret) flows should not traverse a node of lower security level. In this context, for communication to be permitted and routed between two nodes, both nodes and any intermediate nodes must adhere to the *"no read up, no write down"* policy.

Lu and Sundareshan [8] envisioned such an MLS model that enforces the information flow policy at network endpoints. The problem with this approach is that the scale and dynamic behavior of modern networks make deploying such an enforcement mechanism on every endpoint impractical. However, the inherent centralization of software-defined networks (SDNs) allows enforcement of a network-level MLS policy in a scalable and efficient manner.[1] Determination of whether or not flows are permitted can be done by the controller, and the policy can be enforced by the data-plane switches in the form of flow rules. This offers the significant advantage over previous work of allowing the service to be provided transparently to the entire network, because network devices do not have to run specialized software. The controller's global view of the network also offers greater flexibility as changes to labels and policy can be reflected by simple changes to flow rules, as opposed to manually re-configuring individual devices.

Ultimately, multilevel security can ensure that the network achieves (to the degree possible) least-privilege isolation of entities of different access classes (per their security labels). Source nodes of certain levels are not permitted to, for example, send data to any nodes of lower security level. Similarly, switches of certain security levels are not permitted to forward (specifically, receive then forward), or potentially eavesdrop, flows of higher security levels. Thus, we denote the flow paths in this paradigm as *secure paths*.

[1]Note that, in general, SDNs suffer the threat of single-point-of-failure. In the event of failure, MLSNet is just one of all controller applications affected, and this problem is out of scope of this work.

III. MLSNet Overview

In this section, we present our threat model, lattice of security labels, and policy constraints for MLSNet.

A. Threat Model and Assumptions

For the assignment of security labels, we assume a *Network Security Officer (NSO)*, as defined by Lu and Sundareshan [8], who assigns appropriate security labels (i.e., levels and categories) to the network entities (e.g., endpoint devices and forwarding nodes). [2] The assignment can be done by leveraging the controller as it has a global view of the network, and it must be based on a security assessment of the entities in the network. For example, endpoints with unpatched operating systems should be assigned a lower security level, as they are more likely to contain vulnerabilities than others with the latest software updates. IoT devices or forwarding nodes connected to third-party networks can also be considered less secure, and therefore should be assigned a lower security level and a restricted set of categories. In contrast, endpoints containing more sensitive (e.g., top-secret) data should have a higher security level assigned since information flow to nodes with lower levels should be prevented.

MLSNet aims to protect confidentiality by preventing leakage to unauthorized entities. We assume that nodes connected to a network may become compromised and have malicious intentions. In this scenario, we aim to limit an adversary's ability to further compromise the network.

Additionally, we assume the controller has an accurate view of the topology (i.e., nodes have not been spoofed). MLS cannot detect all forms of packet spoofing, and we rely on other SDN-based defenses to detect packet spoofing against the topology discovery service [9].

B. Multilevel Security Lattices for Computer Networks

To compute an SDN-based network configuration (set of flow rules installed to SDN switches) that satisfies the security policy, we must first consider security levels and categories. As drawn from *Denning* [2], we order the security levels used in our model according to the following: *TopSecret (4)* $>$ *Secret (3)* $>$ *Confidential (2)* $>$ *Public (1)*. For an SDN, we define the security categories as the packet types supported by the OpenFlow [21] protocol for matching incoming packets to flow rules: TCP, ICMP, etc. Although, any number of levels and categories can be defined to separate classes of flows; we just use the above descriptions as one example for evaluation.

The combination of a level and one or more categories then forms the label at a node. These labels form a lattice, a partially ordered set that reflects the secrecy and privilege requirements of communication in the network. We consider a node's label to be higher than another node's if the former's level is greater than or equal to, and the categories form a superset of, the latter's. This can be seen in the sample lattice shown in Figure 2. We note that this construction may lead to incomparable labels, where neither label is a subset/superset of the other, in which case communication would be denied by default. This will preserve confidentiality but with the caveat that not every flow may be accommodated.

[2]Note that if business continuity required any nodes to communicate, then they should be labeled appropriately. Our assumption is that MLSNet is given as input a set of labels deemed appropriate per business needs, while the problem of changing labels to fit business needs is orthogonal to our work.

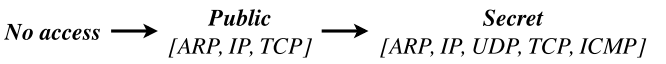


Fig. 2. An OpenFlow specific security lattice for the networks used in our evaluations. Here, information may only flow to nodes with the same or higher security label (i.e., from left to right in the lattice).

Given the labels, the controller will install flow rules to the SDN switches to allow communication only if the security constraints are satisfied.

C. Security Policy Constraints

In this section, we discuss the *access control* and *flow control* constraints, which form the basis of our security policy.

Access Control: As the first step to compute a security policy compliant network configuration, we determine if a subject (e.g., user or process) initiating communication with an object (e.g., file or resource) is allowed to exchange information with the object based on the security levels and categories. To define the constraints for access control, we have to consider if the subject is communicating with a receiver object (i.e., object receives from subject) or a provider object (i.e., subject sends to object). If the subject communicates with a provider object, then information flows from object o to subject s; inversely, if object o is a receiver object, then information flows from s to o. In case the object is both a provider and receiver object at the same time, information flow between s and o is bidirectional.

Considering these three cases, given security level σ and categories C of subject s and object o, the authorized information flows are defined by a conventional MLS confidentiality model [22]. For a subject s (e.g., workstation user) communicating with a provider object o, (e.g., mail server in the DMZ in Figure 1), the following constraint must be satisfied:

$$\sigma_o \leq \sigma_s \text{ and } C_o \subseteq C_s, \ \forall o \in O, s \in S \tag{1}$$

Secondly, for a subject s (e.g., Wi-Fi client) communicating with a receiver object o (e.g., network printer), the following constraint must be satisfied:

$$\sigma_o \geq \sigma_s \text{ and } C_o \supseteq C_s, \ \forall o \in O, s \in S \tag{2}$$

And for a subject s (e.g., workstation user) communicating with an object o that is both a provider and receiver (e.g., git repository), the following constraint must be satisfied:

$$\sigma_o = \sigma_s \text{ and } C_o = C_s, \ \forall o \in O, s \in S \tag{3}$$

Upon the initial arrival of a flow at the SDN controller from a subject, it can be determined if the subject s is allowed to exchange information with object o by considering the access control constraints. While we define our framework in a general way, security levels and categories are defined specific to SDNs, as discussed in Section III-B.

Flow Control: If the access control constraints are satisfied, information is allowed to flow between object o and subject s.

The next step before rule installation is for the controller to determine whether there exists a path between *o* and *s* such that the *security level* of any forwarding node on the path between *o* and *s* is not lower than that of the flow. In Figure 1, the security level of the switch connecting the publicly accessible Wi-Fi to the network is lower compared to the switches connected with the secure VPN, which are behind firewalls and only for internal users. Traversing lower classified nodes puts a flow at risk of being leaked to untrustworthy entities, being modified, or otherwise disrupted. Thus, protection of confidentiality is constrained by secure path selection, and MLSNet will choose an optimal path (if one exists) that satisfies this constraint for any candidate flow.

We can formulate such a constraint by stating that the security level σ of a node *j* on the path between *o* and *s* cannot be lower than the security level of the originating node of the flow. As with the access control constraint, we have to take into account whether a subject is communicating with a provider object, a receiver object, or an object that is both. If the subject is communicating with a provider object, then the following constraint must be satisfied:

$$\sigma_o \leq \sigma_j,\ \forall j \in V\ \text{ on path }(o,s)\ \text{ for flow }\ f \in F \quad (4)$$

Secondly, if the subject is communicating with a receiver object, then traffic is flowing from the subject toward the object, and the following constraint must be satisfied:

$$\sigma_s \leq \sigma_j,\ \forall j \in V\ \text{ on path }(o,s)\ \text{ for flow }\ f \in F \quad (5)$$

Lastly, the access control constraint for communicating with an object that is a receiver and provider at the same time defines that *s* and *o* are required to have the same security level as stated in (3). Therefore, the flow control for such a case requires a forwarder node to have a security level that is higher or equal compared to the level of *s* and *o*:

$$(\sigma_s, \sigma_o) \leq \sigma_j,\ \forall j \in V\ \text{ on path }(o,s)\ \text{ for flow }\ f \in F \quad (6)$$

In addition to the security labels, we also must consider the capacity κ_{ij} of a link (i, j) on a path between *s* and *o* for a flow with a size of d^f: $\kappa_{ij} \geq d^f \forall (i,j)$ on path (*o*, *s*). As a trade-off for providing flow control, policy compliant paths may be longer than a shortest available path which does not consider a security policy. Additionally, in case two nodes satisfy the access control constraint, there is no guarantee that a path between the nodes can be found which fulfills the flow control constraint. If such, there may be a path traversing nodes which *do not* have a high enough security label. We refer to such cases as *policy conflicts*. In Section IV-C, we present a model to minimize policy conflicts on flow paths. In short, it will find the best fitting configuration and report the exact locations on paths where policy conflicts exist. By deploying additional security mechanisms, such as *declassification* via encrypted communication channels, such conflicts can be resolved, as we further discuss in Section IV-F.

IV. Policy Compliant Flows

Given the policy constraints and security labels, we introduce optimization models to compute a flow-rule-based network configuration under consideration of policy compliance and resource availability. We first introduce an integer linear programming (ILP) model to maximize the number of flows strictly satisfying all security constraints. If no path meeting the required security constraints can be found for a flow *f*, the model will suggest to drop *f*. However, if all flows must be routed, we propose a second ILP-based optimization model that minimizes the sum of policy conflict values along paths for the remaining flows.

A. Policy Compliant Flow Maximization Problem

In this section, we introduce a *binary integer programming* model, a special case of integer linear programming (ILP), to maximize the number of flows that can be accommodated by a network under consideration of capacity and security constraints. We refer to this problem as the *policy compliant flow maximization problem*, and formulate the constraints in (7). The optimization model shown determines if a network configuration fulfilling the defined security policy can be found to route the flows *F* between the subjects *S* and objects *O*. To compute a path, we first introduce a binary decision variable x_{ij}^f to indicate if link $(i, j) \in E$ is used on the path for flow f (i.e., $x_{ij}^f = 1$) or not (i.e., $x_{ij}^f = 0$). To decide if a flow f can be accommodated, we also introduce the binary decision variable α^f.

$$\max \sum_{f \in F} \alpha^f \quad (7a)$$

$$\text{s.t.} \sum_{i:(i,s)\in E} x_{is}^f + \alpha^f = \sum_{j:(s,j)\in E} x_{sj}^f,\ \forall f \in F \quad (7b)$$

$$\sum_{j:(o,j)\in E} x_{oj}^f - \alpha^f = 0,\ \forall f \in F \quad (7c)$$

$$\sum_{i,j\in E} x_{ij}^f = \sum_{j,k\in E} x_{jk}^f,\ \forall f \in F \quad (7d)$$

$$\sum_{i:(i,j)\in E} x_{ij}^f \leq 1,\ \forall f \in F, j \in V \quad (7e)$$

$$\sum_{f \in F} x_{ij}^f \cdot d^f \leq \kappa_{ij},\ \forall (i,j) \in E \quad (7f)$$

$$\alpha^f \cdot lev(\sigma_o, \sigma_s) = \alpha^f,\ \forall o(f) \in O, s(f) \in S, f \in F \quad (7g)$$

$$\alpha^f \cdot cat(\lambda_o^c, \lambda_s^c) = \alpha^f,\ \forall o(f) \in O, s(f) \in S, c \in C, f \in F \quad (7h)$$

$$x_{ij}^f \cdot orig(\sigma_o, \sigma_s) \leq x_{ij}^f \cdot \sigma_j,\ \forall (i,j) \in E, o(f) \in O, s(f) \in S, f \in F$$

$$x_{ij}^f \in \{0, 1\},\ \forall (i,j) \in E, f \in F$$

$$\alpha^f \in \{0, 1\},\ \forall f \in F \quad (7i)$$

In (7b), we add α^f to the link indication variable x_{is}^f to trigger a flow f at a subject s. To compute a path between the subject node s and object node o, a flow f is consumed at a node o, as stated in constraint (7c), by subtracting α^f from the link indication variable. In (7d), we state the flow preservation

constraint to ensure that the sum of incoming flows into a node equals the sum of outgoing flows of a node.

We add constraint (7e) to limit the number of visits of a node to one for each flow. Constraint (7f) ensures that the given capacity κ_{ij} of a link $(i, j) \in E$ is not exceeded for forwarding flows over a link i, j with a size of d^f per flow. Typically, in bidirectional communication in computer networks the size of the request flow is different than the size of the reply flow. Since we assume symmetric routes, the flow size variable d^f should be chosen to account for the flow size in both directions. Additionally, since new flow demands typically arrive at different times in a network, we can replace the above link capacity κ_{ij} with the residual capacity $\widetilde{\kappa}_{ij}$ which states the remaining capacity on a link $(i, j) \in E$ considering the existing flows in a network traversing link (i, j).

In constraints (7g) and (7h), we define the access control properties. Constraint (7g) ensures that a flow f between a subject s and an object o is only permitted if the function $lev(\sigma_o, \sigma_s)$, shown in (8), returns 1, indicating that the security levels of s and o allow communication:

$$lev(\sigma_o, \sigma_s) = \begin{cases} 1, & \textbf{if } \text{o is provider object } \textbf{and } \sigma_o \leq \sigma_s \\ 1, & \textbf{if } \text{o is receiver object } \textbf{and } \sigma_o \geq \sigma_s \\ 1, & \textbf{if } \text{o is both } \textbf{and } \sigma_o = \sigma_s \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

As defined in Section III-C for access control, we further have to ensure that the subject s and object o have the appropriate security categories before calculating a path. In function $cat(\lambda_o^c, \lambda_s^c)$ shown in (9), we model the requirement of security categories to allow a flow between s and o:

$$cat(\lambda_o^c, \lambda_s^c) = \begin{cases} 1 - (\lambda_o^c - \lambda_o^c \cdot \lambda_s^c), & \textbf{if } \text{o is provider} \\ 1 - (\lambda_s^c - \lambda_s^c \cdot \lambda_o^c), & \textbf{if } \text{o is receiver} \\ 1 - (\lambda_s^c - \lambda_o^c) \cdot (\lambda_s^c - \lambda_o^c), & \textbf{if } \text{o is both} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

To mathematically define this, we introduce variable λ_i^c which indicates if a node i has a security category c, i.e., $\lambda_i^c = 1$, or not, i.e., $\lambda_i^c = 0$. As an example for the operation of function *cat*(), suppose a subject s wants to communicate with a provider object o. In order to permit the flow, the constraint that $C_o \subseteq C_s$ must be satisfied. To evaluate if the security categories C_o of an object are a subset of the categories in C_s, we introduce the formulation $1 - (\lambda_o^c - \lambda_o^c \cdot \lambda_s^c)$ as shown in (9). This will evaluate to 0 if subject s does not have a security category c, but object o does, i.e., $(1 - (1 - 1 \cdot 0)) = 0$. Such a case does not fulfill the access control constraint, and therefore the flow cannot be admitted, i.e., $\alpha_f = 0$.

Function *cat*() works in a similar way if o is a receiver object. In case o is both a provider and receiver, function *cat*() evaluates to 1 if $C_o = C_s$. As stated in constraint (7h), the function *cat*() has to return 1 for all categories $c \in C$ for a flow f between a subject $s(f) \in S$ and an object node $o(f) \in O$. Additionally, in constraint (7i), we define the secure flow property to prevent information flow to lower classified nodes. Thus, for each next node j on a link (i, j) of a flow f, indicated by the decision variable x_{ij}^f, the security class of the originating node of flow f (i.e., the subject if the object is a receiver, and the object otherwise) has to be less or equal to the security class at the next node j on the path. We define function $orig(\sigma_o, \sigma_s)$ as shown in (10), where *orig*() returns the security level depending on the type of object node o, according to the defined flow control constraint in Section III-C:

$$orig(\sigma_o, \sigma_s) = \begin{cases} \sigma_s, & \textbf{if } \text{o is receiver} \\ \sigma_o, & \text{otherwise} \end{cases} \tag{10}$$

This last constraint ensures that on a path between a subject s and an object node o, no forwarding nodes with a lower security level compared to the security level of the originating node of the flow are visited. We then use the specified constraints (7b)-(7i) as the basis for our heuristic-based maximization algorithm discussed in the next section.

Algorithm 1 PolicyCompliantPath(G,s,o,d^f)

```
V = nodes in G
for all v ∈ V do
    dist[v] = infinity, prev[v] = null
end for
dist[s] = 0
N = nodes in G
if lev(σ_o, σ_s) = 1 and cat(λ_o^c, λ_s^c) = 1, ∀λ_o^c ∈ C_o, λ_s^c ∈ C_s then
    while N not empty do
        i = node in N with smallest dist[]
        remove i from N
        for all adjacent node j of i do
            if orig(σ_o, σ_s) ≤ σ_j and d^f ≤ κ̃_ij then
                dist_new = dist[i] + 1
                if dist_new ≤ dist[j] then
                    dist[j] = dist_new
                    prev[j] = i
                end if
            end if
        end for
    end while
end if
return prev
```

B. Policy Compliant Flow Maximization Algorithm

The linear programming model introduced in Section IV-A is a binary integer programming model, a special case of integer linear programming (ILP) since all variables are binary. Integer linear programming models are NP-hard problems in general, and the special case of binary integer programming is one of Karp's 21 NP-complete problems [23]. Although solvers such as *Gurobi* [24] are efficient in computing a solution for such problems, binary integer programming models can be impractical to solve for certain inputs.

To address this issue, we also formulate a heuristic algorithm to compute a security compliant path between subjects and objects based on a modification of *Dijkstra's* shortest path algorithm. In Algorithm 1, we extend Dijkstra's

algorithm by enforcing the access control constraints, flow control constraints, and the residual link capacities (similar to the constraints presented in (7). Specifically, we formulate the access control constraint in line 7 based on the introduced functions *lev*() as defined in (8) and $cat()$ as defined in (9). To compute a secure path between *s* and *o* we define the constraints in line 12 to only consider an adjacent node *j* of a link if the security level of node *j* is greater or equal the security level of the originating node of flow *f* and the link connecting node *i* and *j* has enough residual capacity to accommodate flow *f*. In effect, the introduced model and algorithm will compute paths that accommodate the maximum number of flows $f \in F$ between a subject node *s* and an object node *o*, with consideration for security and link capacity. Moreover, a natural result of this strategy is that the algorithm will tend to find similar paths for similar level flows, thus further keeping flows isolated from dissimilar-labeled switches.

C. Policy Conflict Minimization Model

Finding a path fulfilling all security conditions might not always be possible considering the nature of real-world networks. In contrast to the previous model, here we assume that *all* flows fulfilling the access control and link capacity constraints must be accommodated in the network, which may lead to policy conflicts. Policy conflicts are conditions where a flow is visiting a node on a path that has a lower security level than the transferred information (i.e., than the sender node), and we quantify a policy conflict as the numerical difference between those security levels. In this scenario, the network administrator is assuming the risk of using insecure paths in order to achieve 100% coverage of flows. Thus, while the network is still subject to traffic analysis attacks even with encrypted traffic, we suggest here using encryption as a minimal security measure over the data traversing the unsafe links, if not already using encryption.

Considering the lattice in Section III-B, we assume that nodes classified as *Confidential (2)* have a higher risk of being compromised than nodes classified as *Secret (3)*. The goal here is to minimize policy conflicts; therefore, if information classified as *Top Secret (4)* is transferred on a path with policy conflicts, it is preferable to select nodes with the smallest numerical difference (i.e., *Secret (3)* nodes are preferred over *Confidential (2)* nodes).

Resolving policy conflicts requires additional security measures (e.g., *declassification*). The larger a policy conflict (i.e., higher numerical difference in security levels), the more an additional security measure will cost, in terms of transmission time or computation overhead. By minimizing the numerical distance of policy conflicts, we aim to minimize the *cost* required to apply additional security measures to meet a defined security policy.

To achieve this, we compute a network configuration in a two-step process. We first select a subset of the flows $F_l \subseteq F$ that fulfill the access control constraints, and second, compute

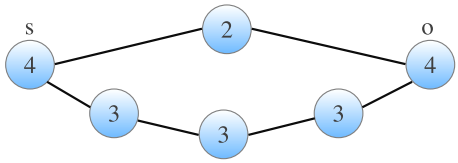


Fig. 3. An example network where path selection is done selectively based on the security levels of nodes. Here, multiple potential paths exists, where the best one is chosen for the flow based on the imposed policy conflict value.

paths between subjects and objects with the objective to minimize security policy conflicts. We define the access control constraints as follows:

$$F_l = \{f \in F : lev(\sigma_o, \sigma_s) = 1 \text{ and } cat(\lambda_o^c, \lambda_s^c) = 1, \forall \lambda_o^c \in C_o,\ \forall \lambda_s^c \in C_s,\ o \in O, s \in S\} \quad (11)$$

Next, for the set of legitimate flows F_l, we also define an objective function, *conf*, to find a network configuration that accommodates all flows in F_l while minimizing the policy conflicts on a path of a flow $f \in F_l$ between a subject *s* and an object *o*. The function returns the difference between the security level of the flow's originating node, given by $orig(\sigma_o, \sigma_s)$, and the security level σ_j of a node *j* on the path between *s* and *o* if $\sigma_j < orig(\sigma_o, \sigma_s)$. More formally:

$$conf\left(\sigma_o, \sigma_s, \sigma_j\right) = \begin{cases} orig(\sigma_o, \sigma_s) - \sigma_j, & \textbf{if } \sigma_j < orig(\sigma_o, \sigma_s) \\ 0, & \text{otherwise} \end{cases} \quad (12)$$

We aim to minimize the policy conflicts caused by visited nodes with lower security levels. Assuming a flow originates from a node *o*, we define the severity of the policy conflict by the numerical distance from level σ_o of node *o* to a node *j* with level σ_j, if $\sigma_j < \sigma_o$. Then, choosing a node *j* over a node *h*, where $(\sigma_o - \sigma_j) < (\sigma_o - \sigma_h)$, is preferable. And for selecting a secure path, we want to give preference to these nodes with a smaller difference in security level with the originating node, even if such a path is significantly longer than the shortest path. To model this, we introduce a factor γ and define our objective function as follows:

$$\min \sum_{f \in F_l} \sum_{i,j \in E} x_{ij}^f \cdot \gamma^{conf(\sigma_o(f), \sigma_s(f), \sigma_j)} \quad (13)$$

In (13), x_{ij}^f denotes the decision variable if link (*i*, *j*) is selected as part of the path between *s* and *o* for a flow *f*. In the objective function as shown in (13), $\sigma_o(f)$ denotes the security level of object node *o* of a flow *f*, $\sigma_s(f)$ denotes the security level of subject node *s* of a flow *f*. The security level of a node *j* on the path between *s* and *o* is defined by σ_j. The factor γ controls the length of a path that should be chosen as a trade-off for visiting nodes with a smaller distance in terms of security levels. We visualize this in an example shown in Figure 3.

Considering this small network, two possible paths exist between *s* and *o*. The numbers in the nodes indicate their security level. If we select a factor $\gamma = 4$, the top path results in a value of $4^{4-2} = 16$, while the bottom path has a *smaller*

policy conflict value of $3 \cdot 4^{4-3} = 12$, and thus would be selected. In contrast, if we select a smaller value for γ (e.g., $\gamma = 2$) then considering the top path, a transition from the node with security level 4 to the node with security level 2 has to be made, resulting in a policy conflict value of $2^{4-2} = 4$. Computing the policy conflict value for the bottom path would result in $3 \cdot 2^{4-3} = 6$, since we have to visit three nodes with a difference in the security level of 1. Based on this, the top path would be selected, although from a security perspective, the bottom path may be more preferable since nodes with a smaller security level difference are visited. This example shows that the factor γ controls the selection of longer paths visiting nodes with a smaller security level difference. To always select paths with the smallest security level difference, γ can be set to the network diameter $+1$, in terms of hop count, which we prove as follows.

Lemma 1: To select a longer path with lower policy conflicts, we must set γ to the maximum path length $+1$.

Proof: Assuming a flow originating from an object o, we have to show that for a path of flow f defined by a set of links $(i, j) \in E$ indicated by the decision variable x_{ij}^f, the value of policy conflicts specified as $\sum_{i,j \in E} x_{ij}^f \cdot \gamma^{\sigma_o - \sigma_j}$ and assuming that $\forall j, \sigma_j < \sigma_o$, is larger for a path with higher policy conflicts than a potentially longer path with a lower conflict value if γ is chosen appropriately. Given a candidate node for the path of flow f with a policy conflict of a, we want to choose γ so that a potentially longer path y over a set of nodes with a lower policy conflict of b is selected, i.e., $\gamma^a > y \cdot \gamma^b$. Since $a > b$, we can express b as $a - q$, where q is the numerical difference of the security levels of a and b, i.e., $q = a - b$. By replacing b with $a - q$, we can write the inequality above as $\gamma^a \cdot \gamma^q > y \cdot \gamma^a$. Assuming the smallest absolute difference of two unequal security classes, i.e., $q = 1$, the inequality above can be written as $\gamma > y$. Therefore, we can say that in order to select a path that is y hops longer, over a set of nodes with a lower policy conflict value, we have to select a value for γ that is at least $y + 1$. This also holds for larger security differences, since $\gamma^q < \gamma^{q+1}$ holds true for positive values of q. ∎

D. Policy Conflict Minimization Problem

To minimize the policy conflicts on a path, we formulate the optimization problem as an integer linear programming (ILP) model. We refer to this problem as the *security policy conflict minimization problem*, as shown in (14).

To trigger a flow at a node s, (14b) adds 1 to the decision variable x_{is}^f. In our formulation to compute a path from s to o, the flow is consumed at node o as stated in constraint (14c). In (14d), we state the flow preservation constraint to ensure that the sum of incoming flows to a node equals the sum of outgoing flows. Constraint (14e) ensures that the given capacity κ_{ij} of a link $(i, j) \in E$ is not exceeded by forwarding flows $f \in F_l$ with a size of d^f per flow. In (14e), we assume that the flow size d^f is chosen to include traffic between s and o in both directions since we assume symmetric routes. Since new flow demands typically arrive at different times in a network, we can replace the above link capacity κ_{ij} with the residual capacity $\widetilde{\kappa}_{ij}$ which states the remaining capacity on a link $(i, j) \in E$ considering the existing flows traversing link (i, j). Accordingly, with the specified constraints (14b)-(14e), the introduced model will compute a path for every flow $f \in F_l$ between a subject node $s(f)$ and an object node $o(f)$ with the objective function as defined in (14a).

$$\min \sum_{f \in F_l} \sum_{i,j \in E} x_{ij}^f \cdot \gamma^{conf(\sigma_o(f), \sigma_s(f), \sigma_j)} \tag{14a}$$

$$\text{s.t.} \sum_{i:(i,s) \in E} x_{is}^f + 1 = \sum_{j:(s,j) \in E} x_{sj}^f, \ \forall f \in F_l \tag{14b}$$

$$\sum_{j:(o,j) \in E} x_{oj}^f - 1 = 0, \ \forall f \in F_l \tag{14c}$$

$$\sum_{i,j \in E} x_{ij}^f = \sum_{j,k \in E} x_{jk}^f, \ \forall f \in F_l \tag{14d}$$

$$\sum_{f \in F_l} x_{ij}^f \cdot d^f \leq \kappa_{ij}, \ \forall (i,j) \in E$$

$$x_{ij}^f \in \{0, 1\}, \ \forall (i,j) \in E, f \in F_l. \tag{14e}$$

E. Policy Conflict Minimization Algorithm

As discussed in Section IV-B, ILP models with binary integer variables, such as (14), are typically NP-hard and can be impractical to solve for certain input sequences. To address this, we also propose a heuristic algorithm to approximate an optimal solution and replace the objective to find the shortest path with the objective to compute a path with the smallest sum of policy conflict values (Algorithm 2, lines 12-13).

Since Algorithms 1 and 2 are based on *Dijkstra's shortest path algorithm*, we can express their time complexity as $O(|F| \cdot (|E| + |V| log |V|))$ for a number of $|F|$ flows.

F. Resolving Policy Conflicts

As we've shown in the previous section, paths with sufficient security levels and capacity may not always exist for two nodes permitted to communicate. In such a case, additional security mechanisms must be applied on the flow in order to be policy compliant. These mechanisms typically involve a cost to implement (e.g., increased transmission delay, processing time, or capacity), which the latter model aims to minimize.

An important mechanism for resolving policy conflicts is *declassification*, which is the process of lowering the security level of the information. Sabelfeld and Sands [25] discuss a general framework for declassification by defining the dimensions of information release, including: *what* information is released, *who* releases the information, *where* information is released, and *when* it is released. By analyzing these dimensions, a network operator is then able to evaluate the risks and benefits of declassification to resolve certain security policy conflicts. In our framework, declassification can involve lowering a flow's security level so it can traverse a path with lower classified nodes than the originating node. Methods to achieve this include traffic camouflaging techniques [26] or VPNs to defend against traffic analysis.

Resolving policy conflicts can also be achieved by the NSO relabeling certain nodes in the network (e.g., increasing the

Algorithm 2 MinConflictPath(G,s,o,d^f)

```
V = nodes in G
for all v ∈ V do
    conf[v] = infinity, prev[v] = null
end for
conf[s] = 0, N = nodes in G
if lev(σ_o, σ_s) = 1 and cat(λ_o^c, λ_s^c) = 1, ∀λ_o^c ∈ C_o, λ_s^c ∈ C_s then
    while N not empty do
        i = node in N with smallest conf[]
        remove i from N
        for all adjacent node j of i do
            if d^f ≤ κ̃_ij then
                conf_new = conf[j] + γ^conf(σ_o,σ_s,σ_j)
                if conf_new ≤ conf[j] then
                    conf[j] = conf_new
                    prev[j] = i
                end if
            end if
        end for
    end while
end if
return prev
```

Algorithm 3 GenerateFlowRule(R, A, P, *next*)

```
rule.append("match:")
for all r ∈ R do
    if r ∈ P then
        rule.append(r = P(r))
    end if
end for
rule.append("action:")
if next! = drop then
    for all a ∈ A do
        rule.append(a)
    end for
    rule.append("next")
else
    rule.append("drop")
end if
return rule
```

security level of forwarding nodes) after upgrading the security measures on a switch and re-evaluating its security level. Our proposed optimization model to minimize policy conflicts will point out exactly which components of the network topology are causing conflicts; therefore, relabeling of nodes can be a permanent solution to policy conflicts which may reoccur.

V. Secure Flow Rule Construction

To realize a policy compliant network configuration, in the following we define a set of principles for the construction of secure flow rules which preserve confidentiality.

A. Isolating Flows

Attacks that exploit the composition of flow rules are effective because the matching criteria often only identifies packets by a limited set of header fields, as discussed by Achleitner *et al.* [14]. If the flow rules are only matching packets against header fields of a certain network layer (e.g., IP addresses), then the information in other layers will be seen as *"wild cards"* and thus will be ignored. This problem motivates the construction of SDN flow rules with consideration of information spanning all network layers.

The OpenFlow protocol [21] defines a set of matching fields supporting different network layers. Multiple endpoints may share lower layer fields such as physical ingress port; thus, to differentiate them and identify their security levels, we must include fields from higher network layers (e.g., IP or Ethernet addresses). But security leaks caused by the exchange of certain packet types in SDN-enabled networks [9], [11], [14] motivate the use of categories in a security lattice to offer finer granularity of information exchange in SDN flow rules. Therefore, we derive these categories from additional packet header fields which may span all layers of the network stack (e.g., *ARP*, *IP*, *TCP*, *UDP* and *ICMP*), and use them in enforcing the security policy.

Additionally, with this general framework, a security category can be defined with even finer granularity. For example, by specifying field subtypes: *ICMP type 8 code 0*, to allow ping packets. Thus, the various header fields allow greater flexibility when defining the security policy, and unlike traditional networks, the policy can be efficiently managed by sending *flow_mod* messages to the forwarding nodes to update their routing tables.

B. Constructing Secure Flow Rules

In SDN-enabled networks, we must consider the assigned security labels during the construction of flow rules at the controller. As described previously, security leaks can arise with imprecise matching criteria. Considering this, we must construct precise flow rules which ensure that only packets fulfilling the defined security level and category constraints can be transmitted. We extend the general method for generating rules to accomplish this. More formally, given a security label of a flow, for secure rule construction, we represent the set of supported fields in the security label for network layer $N_i \in N$, where N is the set of all network layers, as R_{Ni}. Then, the superset of fields to be matched against some packet P during secure rule construction can be realized by taking the union of sets, which we denote as R:

$$R = \bigcup_{N_i \in N : \forall N_i \leq N_P} R_{N_i} \tag{15}$$

This formulation ensures that—after satisfying access control constraints—a flow is *isolated* and handled correctly according to its security categories (i.e., packet fields).

Besides matching criteria, OpenFlow also defines action sets, specifying what actions to apply on matched packets. These include a required action part (e.g., forwarding or dropping) and optional actions (e.g., rewriting packet header fields). We denote the required action part as *next*, which specifies

TABLE III
FLOW MAXIMIZATION BENCHMARK

	AS network			k=8			k=12			k=16		
# Lattice Levels	2 lev.	3 lev.	4 lev.	2 lev.	3 lev.	4 lev.	2 lev.	3 lev.	4 lev.	2 lev.	3 lev.	4 lev.
LP (no congestion)	85%	75%	65.7%	79.5%	64.6%	58.3%	79.5%	66.6%	59%	81%	68.4%	64%
HA (no congestion)	79.6%	71%	63.1%	63%	52%	50.6%	66.5%	57%	51.1%	68.5%	61.1%	53%
LP (cong. network)	53.4%	45.2%	37.4%	45.5%	39.3%	34.3%	56%	46.5%	36.5%	56%	50.6%	36.5%
HA (cong. network)	51.9%	41.8%	35.8%	44.5%	33.5%	31.9%	50%	38%	32.3%	52.3%	45.8%	34.5%

TABLE IV
POLICY CONFLICT MINIMIZATION BENCHMARK

		AS network			k=8			k=12			k=16		
# Lattice Levels		2 lev.	3 lev.	4 lev.	2 lev.	3 lev.	4 lev.	2 lev.	3 lev.	4 lev.	2 lev.	3 lev.	4 lev.
LP	no conflict	85%	75%	65.7%	79.5%	64.6%	58.3%	79.5%	66.6%	59%	81%	68.4%	64%
	1 lev. diff.	15%	21.5%	29.3%	20.5%	24.6%	24%	20.5%	26%	28.5%	19%	22%	20%
	2 lev. diff.	-	3.5%	4.3%	-	10.8%	12.7%	-	7.4%	10%	-	9.6%	11%
	3 lev. diff.	-	-	0.7%	-	-	5%	-	-	2.5%	-	-	5%
HA	no conflict	79.6%	71%	63.1%	63%	52%	50.6%	66.5%	57%	51.1%	68.5%	61.1%	53%
	1 lev. diff.	20.4%	21.5%	24.6%	37%	25.7%	22.1%	33.5%	29%	27.5%	31.5%	23.2%	23%
	2 lev. diff.	-	7.5%	6.6%	-	22.3%	22.6%	-	14%	14.7%	-	15.7%	14%
	3 lev. diff.	-	-	5.7%	-	-	4.7%	-	-	6.7%	-	-	10%

to either send a packet to a specific output port or to drop it. Similarly, we specify the set of optional actions, such as rewriting addresses, as the action set A.

Based on sets R and A for a packet P, we formulate Algorithm 3. We begin rule construction by defining the *matching* part of a flow rule in line 1. We continue to iterate through the set of matching fields in R, as defined in Equation (15), and check in line 3 if a specified field r can be applied to a value in packet P. If this evaluates to true, we add the matching field r and its associated value $P(r)$ to the flow rule in line 4. In line 7, we add the action part of a flow rule and check in line 8 if the action is to drop the packet. In case we specify a rule to drop packets with specific protocol types, we must ensure the priority of that rule is higher than other rules for the flow that allow forwarding for other protocol types (i.e., allowing most IP traffic, but disallowing any UDP over IP).

To resolve flow rule conflicts between rule actions, we refer to existing frameworks such as Porras *et al.* [12]. However, we note that MLSNet inherently generates a conflict-free set of flow rules (per the guarantees offered by MLS), and our contributions are not focused on methods for conflict resolution. Nonetheless, if the packet is forwarded, then the set of action fields and the output port are added to the rule, as shown in lines 10 and 12. With this construction, we can properly generate a secure flow rule configuration and isolate the flows to ensure that confidentiality of information flow is preserved in the network.

VI. EVALUATION

With an MLS policy, adversarial capabilities (in terms of probing, eavesdropping, and lateral movement) are by definition restricted to only that allowed by policy. Here, we still want to be able to route all legitimate flows. In the following, we demonstrate that (in comparison to not enforcing a security policy) a network administrator can still provide strong coverage of network flows. The goal of our approach is to achieve this, while also reducing the security cost associated with guaranteeing all flows be routed.

We perform a simulation-based analysis for both fat-tree and extended-star topologies, which are representative of datacenter and enterprise networks.[3] In Table III, we report on the performance of our framework to find policy compliant paths for flows in various topologies and with lattices of different sizes. In Table IV, we report MLSNet's performance when minimizing the policy conflicts, where accommodating the remaining flows may require routing along paths containing nodes with a lower security-level than required. We further show MLSNet's ability to mitigate common attacks (see Section II-B), such as those executed by the recently proposed reconnaissance tool SDNMap [14], [27].

A. Flow Maximization Benchmark

To evaluate the ability of our framework to maximize the number of policy compliant flows and minimize policy conflicts on paths, we test the introduced linear programming (LP) models and heuristic algorithms (HA) on different network topologies. We first consider a realistic autonomous system (AS) network, which is an extended-star topology. Then, to model common data-center and cloud topologies, we consider different k-ary fat-tree networks [28], where k is the port density of each switch in the network (e.g., 8, 12, and

[3]Note that simply moving sources and sinks to achieve better coverage of flows requires manual relocation of devices, which is impractical for large networks. With MLSNet, we have flexibility in that a network administrator can achieve the same effect through logical security labels and flow rules; if the topology or node levels change, no manual relocation is needed as the changes can be reflected transparently through new flow rules.

16 ports). We consider lattices of 2-4 security levels which are evenly distributed and randomly assigned to the nodes in a network. To generate flows, we randomly pick source and destination node pairs which fulfill the access control constraint and compute paths with our linear program models and heuristic algorithms. Further, we consider networks with different link capacities to simulate congestion. Our results for flow maximization and policy conflict minimization (for the remaining flows) are averaged over several runs and shown in Tables III and IV.

We explore the number of flows able to be routed in the AS3257 Rocketfuel [29] topology (161 nodes, 656 links), as well as *8*-ary (208 nodes, 384 links), *12*-ary (612 nodes, 1296 links), and *16*-ary (1344 nodes, 3072 links) fat-tree networks. For flow maximization, shown in Table III, our framework shows that a majority of flows was always routed securely. For the ASN topology, the number of flows routed by the LP reached a peak of 85% coverage, while the heuristic algorithm reached a peak of 79.6% coverage (i.e., 94% of the optimal). For the remaining three topologies, the heuristic was slightly less performant. We also observed, for any of the topologies, that the number of flows routed securely decreases as the number of security levels increases (from left to right in any row). However, even at 4 security levels (common in military networks), a majority of flows was routed securely in the noncongested network. Certainly, congestion dynamics, node labels, and different traffic types vary with different networks and will affect the number of flows able to be routed, although this situation can be remedied with conflict minimization. Nonetheless, the results demonstrate that the framework is feasible in several network topologies of different sizes.

In the case of congested networks, we observed for the ASN topology that the heuristic is able to achieve 97.2% of the optimal coverage, with quantitatively similar results for the fat-tree topologies. We note that despite the low flow coverage ($\sim$50%) because new flows could not be supported by the links at some specific time, rules may still have a scheduled install at a delayed time (i.e., when the links can support the new flows), so permitted flows do not necessarily have to be discarded.

The key insight here is that the heuristics are effective, achieving on average 89% of the optimal coverage across all experiments. Moreover, in real data-center or cloud networks, edge switches may carry similar traffic [30] and thus have similar security levels and only be limited by the available capacity (i.e., not the security levels). These results show that even in the worst case of random level assignment—where for example higher-level nodes may be surrounded by lower-level ones and thus cannot communicate without a policy conflict—paths (even if longer) can be found for a majority of flows.

B. Flow Minimization Benchmark

We then evaluate our model's ability to route the remaining flows (in the noncongested case) to guarantee availability. Table IV shows that all flows can be routed with minimal policy conflict along the allowed path. We define policy conflicts as the scenario where a node is visited on a flow path that has a lower security level than the transferred information (i.e., the originating node), and quantify it as the difference of the security levels. Minimizing the conflicts also minimizes the additional security measures needed to protect the flows traversing unsafe links (e.g., via stronger encryption).

For the AS network, paths with no conflict can be found for the majority of flows, while most of the remaining flows only impose a conflict of one security level difference. Less than 5% of flows must be routed through even less secure paths in order to guarantee availability. The case is similar for the fat-tree networks; the majority of flow paths have no conflict, approximately 20-30% of flows can be routed with minimal policy conflict of one level, while feasible paths for the remaining flows can also be found, fitting as many flows along two-level difference paths, and so forth.

The key insight here is that most of the remaining flows were able to be routed with a conflict of one security level difference, and the heuristic algorithms are effective, approximating the optimal solution by 87% on average across all experiments. This leaves many questions for future work, where it may be possible to identify whether or not this single-level conflict occurs at *hot* (or commonly used) nodes, and whether that information can be used to relabel nodes (and perhaps repurpose them) or physically reconfigure the network to reduce possible conflicts to a minimum, for any set of flows.

C. Running Time

We observed that the execution time of computing secure paths is strongly correlated with the number of switches in the network, scaling with a power law. As shown in Fig. 4, we observed that the LP solver Gurobi [24] generally requires $>$3 minutes to compute secure paths and $>$10 minutes to compute minimal conflict paths when there are 500 flows in the network, which is impractical in real networks. On the other hand, the greedy heuristic algorithms (in a Python-based implementation) require on average 8.7 seconds and 8.1 seconds, respectively, even when $k = 10$ (with 250 hosts and 125 switches), while smaller networks (e.g., $k = 8$) only require on average 2.7 seconds and 1.5 seconds to compute paths and thus scales more effectively.

However, large datacenters may have thousands of nodes [28] and/or flows, and even mid-sized datacenters may contain several hundred nodes. In these scenarios, solving the optimization with Gurobi can take on the order of several minutes or hours, while the greedy algorithms still may require several minutes. Therefore, the greedy algorithms can compute paths for individual flows in smaller networks. However, in larger networks, the speed of MLSNet can be further improved by buffering precomputed paths, implementation on hardware, or clustering flows before computing paths.[4]

[4]Note that by clustering flows and generating rules by, e.g., subnets, MLSNet can create secure paths between subnets (i.e., for flows that enter or leave similar gateways), instead of generating rules for individual flows. This may be useful when flow tables have limited size (e.g., in modern hardware switches [31]). We leave development of such a system to future work.

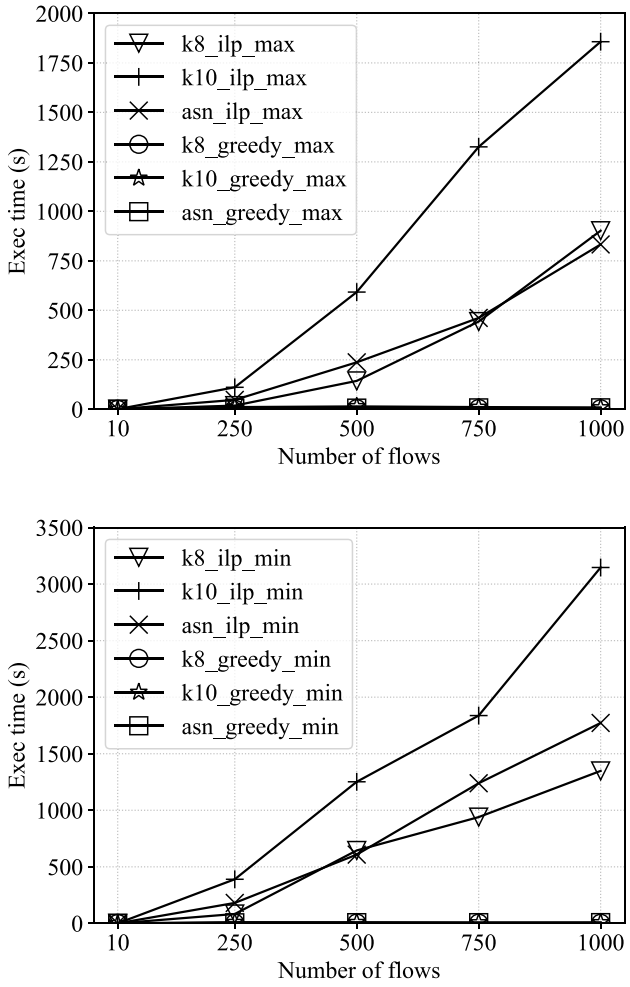


Fig. 4. Execution time for flow maximization (top) and conflict minimization (bottom) benchmarks. The execution time for the ILP solver Gurobi has a power law relationship with the number of flows, while the heuristic (greedy) algorithms scale more efficiently.

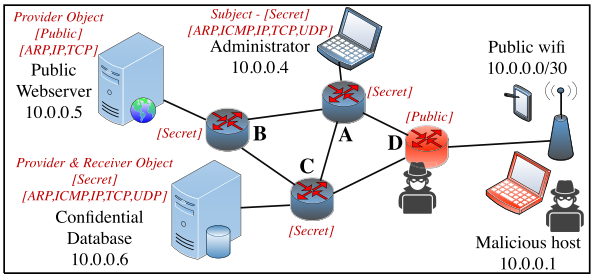


Fig. 5. Experimental network setup with assigned security labels. Using the SDNMap scanner, the (Public) adversary was only able to identify the other Public node in the network, where MLSNet prevented any higher-level traffic from ever reaching them, thus limiting their ability to reconstruct flow rules.

D. Defending Against Attacks

We then implement the network scenario shown in Figure 5 with the SDN simulator Mininet [32], assigning security labels to the nodes and configuring the network to use our framework *MLSNet*. Here, we use SDNMap [27] to demonstrate that our framework can mitigate the attack techniques discussed in Section II-B. SDNMap operates by iteratively probing network nodes with crafted packets and *eavesdropping* on reply messages from all endpoints in the network to reconstruct flow rules (by identifying active hosts and supported protocols). The gathered information is then used to exploit flow rules and bypass security measures such as access-control lists.

Here, we use the security lattice from Figure 2 to configure the network. Running the MLSNet system at the SDN controller, we assign the security classification of *Public - [ARP,IP,TCP]* to the adversary node at 10.0.0.1 (who is using SDNMap). We then let the adversary begin sending probes into the network. The switch default action for an unknown flow is to send it to the controller for inspection and flow rule generation. On receipt of a new flow, the controller will verify the access control constraint of the communicating parties and the flow control constraint of nodes along a potential flow path. A secure flow rule that obeys the security policy will then be generated.

Mitigating Lateral Movement: MLSNet ensures that a node with this classification cannot receive packets from nodes with a higher classification (e.g., from the node at 10.0.0.4 labeled *Secret - [ARP,ICMP,IP,TCP,UDP]*). Therefore, all of the probes destined from the attacker toward a node of higher level should be blocked at the controller, and any induced responses (e.g., from ARP) should also be blocked from flowing back toward the attacker, with no flow rules being generated. We observed exactly this behavior after scanning the network's IP space. As shown in the SDNMap output in the Appendix, the attacker sent out a series of probes enumerating the packet fields (e.g., IP addresses and protocols) to identify active hosts and supported protocols. However, all of the probes sent toward nodes of higher classification were blocked by the controller for not satisfying the access control constraint. Here, SDNMap reported (highlighted in the boxed text) that only the node with IP address 10.0.0.5 replied (the other public host). Although present in the network, the remaining hosts, 10.0.0.4 and 10.0.0.6, were not discovered.

Further, any induced responses over different protocols such as UDP or ICMP were blocked at the controller for not satisfying the flow control constraint as well. Therefore, the attacker discovered that there was a host at 10.0.0.5 that was reachable via TCP (highlighted in boxed text), but none of the other hosts nor protocols supported by them, as the adversary was limited to scanning public nodes over ARP, IP, or TCP. As a result, the adversary was able to reconstruct flow rules for the host at 10.0.0.5. Indeed, future work may investigate optimal labeling and relabeling strategies that can respond to the current network traffic profile in order to dynamically reinforce least-privilege across label categories and further reduce the threat surface.

Mitigating Packet Spoofing: For the same reasons the adversary could not receive response messages to concretely identify nodes of different security labels, MLSNet mitigates the threat of packet spoofing with respect to data exfiltration. Adversaries impersonating nodes of higher security labels and attempting to exfiltrate data to lower-security (i.e., Public, or untrusted nodes) must inevitably traverse a lower-security switch or be destined for a lower-security endpoint. Thus, even if an adversary found a way to identify those active nodes, unless the adversary compromises every lower-security node along a flow path, at least one of them will block the flow per

the MLSNet policy, effectively preventing the exfiltration. Moreover, MLSNet simply rejects the deployment of a flow rule for an unrecognized (i.e., spoofed) source node.

Mitigating Eavesdropping: MLSNet also prevents eavesdropping on communication between nodes of higher-classification by careful construction of flow rules. Since the controller rejects flow rules that direct higher-level traffic toward Public nodes from being generated, the attacker was unable to eavesdrop any traffic passing between the Secret and Confidential nodes at 10.0.0.4 and 10.0.0.6, including broadcast traffic from either node. As a result, *MLSNet* was able to minimize the ability of the attacker to reconstruct flow rules, which may have been the first phase of a potentially much larger attack.

VII. Related Work

A. Confidentiality in Networks

Historically, networks have enforced security policies (i.e., information flow) through firewall and routing configuration. However, these mechanisms are often very coarse and prone to ambiguity, errors, and require coordination across many devices [3], [33], [34]. Indeed, failures due to errors have enabled a variety of attacks to be launched against real-world networks, including device impersonation, man-in-the-middle, performance degradation and denial-of-service [35], [36], [37], [38], among others. Typically these attacks manifest from a small set of techniques: packet spoofing, lateral movement, and eavesdropping, which have been well-known problems since the 90s [39] and have become increasingly important as more information is being put online [40]. In fact, recent work has already demonstrated the ability of an adversary to freely probe within the network to recover sensitive information [9], [10], [11], [12], [13], including active network hosts and even switch flow table rules in software-defined networks [14].

Over time, there have been many defense methods proposed against these techniques, including: source validation to prevent or mitigate packet spoofing [18], [19], firewalling to enforce access policies at network boundaries [3], encryption to prevent unauthorized parties from intelligibly interpreting sniffed data, among others. While each useful in a variety of scenarios, they target specific attack techniques and only partially address the problem of confidentiality—ensuring that only authorized entities have access to some data. Further, while we can accomplish many of the same security goals with a rule-based approach (i.e., per-node, on-demand whitelisting or blacklisting) the inherent problem extends beyond the endpoints to entire route throughout the network (i.e., the switches). Then, network administrators must configure and manage what will quickly become a complex and conflicting set of rules, as demonstrated already to be a problem with firewalls [3]. This motivates our work for developing a solution that exploits multilevel security to provide provable guarantees about who in the network may access what data.

B. Multilevel Security in Networks

Multilevel security allows a network administrator to specify an hierarchical access control policy of a set of subjects on a set of objects. With labels (i.e., a level and categories) given to each subject and object, the policy is enforced through access and flow-control constraints. In fact, multilevel security already plays a critical role in controlling access to information for both military personnel and employees of commercial businesses with different levels of clearance [5]. Common use cases include controlling file access in an operating system [6], object access in generic storage systems [41], table access in a relational database [7], as well as a primitive for securing information flow between variables in programming languages [42].

The notion of multilevel security can also be applied to computer networks, where the MLS policy dictates which nodes are allowed to communicate, what type of traffic they may exchange, and what paths the flows may take in the network. This property precisely address the concerns about confidentiality. We draw inspiration for our framework from the seminal work by Lu and Sundareshan [8] and apply it to SDN-enabled networks. In their work, they introduce a model for multilevel security (MLS) in computer networks by defining a Trusted Network Base (TNB) that is similar to a Trusted Computing Base (TCB) in single-computer systems. The proposed model defines a set of entities (e.g., terminals or printers) and users of the network and relies on the implementation of a security policy by the network endpoints. This approach becomes impractical when having to deploy it on every node in the network, and we exploit the centralization of software-defined networking (SDN) [43] to provide this service transparently to the entire network.

C. Deploying and Verifying Network Policies

There is also a related body of work in leveraging SDNs to deploy network policies that offer other desirable properties, including reachability, loop-free forwarding, and isolation guarantees, among others. Kazemian *et al.* [44] introduced NetPlumber, a verification tool that builds on header-space analysis [45] to incrementally verify updates to network state. Here, header-space analysis provides a formal model of packet processing in the network (i.e., how packets are forwarded, rewritten, encapsulated, etc.), allowing the verification engine to check the *current flow-rule configuration* against target invariants. This work differs from MLSNet in that it focuses on validating a given set of invariants in response to events that trigger network updates (i.e., new flow rule being installed by the controller). MLSNet focuses on constructing a set of rules that inherently adhere to the security policy (i.e., satisfy the security invariants), and furthermore, focuses specifically on access-control invariants. Although, the invariants (security constraints) and flow rules of MLSNet can be plugged into NetPlumber to verify network updates against both the MLS policies and other types of network policies.

Other works introduce similar graph-based models of the network and forwarding functionality to identify conflicting

actions among network policies generated by SDN applications [46], [47], [48] (e.g., different flow rules enforcing conflicting actions against a host). They introduce abstractions for specifying policies over network endpoints, providing a natural way to merge the policies into a conflict-free set. As above, MLSNet differs in that it uses the established guarantees of multilevel security to construct an inherently conflict-free set of flow-rules for the switches at runtime. Moreover, MLSNet extends the security invariants to switches in addition to network endpoints. Although, the security invariants of MLSNet can again be verified by these systems alongside other types of network policies.

D. Trustworthiness of Nodes in Networked Environments

An important requirement to configure a network based on a defined multilevel security policy is the determination of trusted nodes in a network, and formulating a security lattice based on different levels of trust. Recently published papers [49], [50], [51] point out methods to determine trustworthiness of nodes in networked environments.

Jiang *et al.* [49] discuss the data collection process with unmanned aerial vehicles (UAV) in a large-scale Internet of Things deployment. A major problem in such systems are compromised nodes which results in a declined network lifetime and delivery of unreliable and corrupted data. To address this problem, the authors propose a mechanism to evaluate the quality and trustworthiness of the collected data by UAVs from a wireless sensor network (WSN).

Security issues of data collection in Smart Internet of Things (SIoTs) networks are further discussed by Li *et al.* [50]. In their paper, vehicles are considered as a data collection mechanism in SIoTs to deliver information from sensors to a data center (DC). In the discussed scenario, malicious vehicles can lead to data loss which has a negative impact on the security of the data collection process. The aim of their proposed work is to improve the security of the data collection process by selecting trusted vehicles for the data collection process.

Similarly, Ren *et al.* [51] discuss the security challenges of efficient data collection in a Peer-to-Peer (P2P) network. In such scenarios, collecting data from nodes faces the challenge of unknown trustworthiness of a data collector before a certain cost is paid to collect and verify the data. The proposed solution considers a machine learning system that predicts the trust value of a data reporter based on collected historical information.

VIII. Conclusion

In this article, we propose *MLSNet*, a framework which can efficiently enforce an MLS policy by generating secure flow-rule configurations. Built upon *access control* and *flow control* constraints, we develop models and heuristic algorithms to compute policy compliant configurations according to two goals: satisfying a strict flow policy and a soft policy. For the deployment of a policy compliant network configuration, we define principles for secure flow rule construction. We then demonstrate that our framework can deploy network configurations able to withstand recently identified attacks on SDNs. We hope this framework will serve as a base for further investigation into defenses which protect the network with a broader scope than specific attacks and efficient mechanisms for resolving policy conflicts in real-time (perhaps, using advances such as P4 [52] to implement rich SDN features on the switches).

Appendix

```
  SDNMap output:
1: SDNMap/python main.py 10.0.0.0/29 TCP
h1-eth0 []
2: Sending ARP request to 10.0.0.0
3: ···
4: Sending ARP request to 10.0.0.7
5: 10.0.0.1 / 00:00:00:00:00:01 received
response from the following hosts:
 6 : 10.0.0.5 / 00:00:00:00:00:05
7: --------------------
8: Use 10.0.0.5 / 00:00:00:00:00:05 for
probing
9: -- Determine enforced protocols --
10: ----- Check with TCP -----
11: Check if host at 10.0.0.5 -
00:00:00:00:00:05 is reachable with src
addresses 10.0.0.1 - 00:00:00:00:00:01 with
TCP on src port 64836 and dst port 36748
 12 : Host is reachable via TCP!
13: --------------------
14: ----- Check with ICMP -----
15: Check if host at 10.0.0.5 -
00:00:00:00:00:05 is reachable with src
addresses 10.0.0.1 - 00:00:00:00:00:01 with
ICMP
16: --------------------
17: ----- Check with UDP -----
18: Check if host at 10.0.0.5 -
00:00:00:00:00:05 is reachable with src
addresses 10.0.0.1 - 00:00:00:00:00:01 with
UDP on src port 64836 and dst port 36748
19: --------------------
 20 : Accepted protocols:
21: TCP
22: -- Determine which L2/L3 fields are
enforced using TCP --
23: ----- Check if layer 3 routing is used
-----
24: Check if host at 10.0.0.5 -
00:00:00:00:00:05 is reachable with src
addresses 10.0.0.205 - 00:00:00:00:00:01 from
port 64836 to port 36748
25: Spoof ARP cache at 10.0.0.5 from
10.0.0.205 to 00:00:00:00:00:01
26: Check if host at 10.0.0.5 -
00:00:00:00:00:05 is reachable from 10.0.0.205
- 00:00:00:00:00:01 from port 64836 to port
36748
```

```
27: --------------------
28: ----- Check if layer 2 routing is used
-----
29: Check if host at 10.0.0.5 -
00:00:00:00:00:05 is reachable with src
addresses 10.0.0.1 - 00:00:00:25:12:b2 from
port 64836 to port 36748
30: Spoof ARP cache at 10.0.0.5 from 10.0.0.1
to 00:00:00:25:12:b2
31: Check if host at 10.0.0.5 -
00:00:00:00:00:05 is reachable from 10.0.0.1
- 00:00:00:00:00:01 from port 64836 to port
36748
32: Spoof ARP cache at 10.0.0.5 from 10.0.0.1
to 00:00:00:00:00:01
33: --------------------
34: ···
35 : ----- Reconstructed rules ------
36: match=type:tcp,dl_src:00:00:00:00:00:01,
dl_dst:00:00:00:00:00:05,
tp_src:64836,tp_dst:36748, nw_src:10.0.0.1,
nw_dst:10.0.0.5
actions=output:#OUT_PORT
37: match=type:tcp,dl_src:00:00:00:00:00:05,
dl_dst:00:00:00:00:00:01, tp_src:36748,
tp_dst:64836, nw_src:10.0.0.5,nw_dst:10.0.0.1
actions=output:#OUT_PORT
38: --------------------
```

ACKNOWLEDGMENT

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] J. Jang-Jaccard and S. Nepal, "A survey of emerging threats in cybersecurity," *J. Comput. Syst. Sci.*, vol. 80, no. 5, pp. 973–993, 2014.

[2] D. E. Denning, "A lattice model of secure information flow," *Commun. ACM*, vol. 19, no. 5, pp. 236–245, 1976.

[3] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, "FIREMAN: A toolkit for firewall modeling and analysis," in *Proc. IEEE Symp. Security Privacy (SP)*, Oakland, CA, USA, 2006, pp. 199–213.

[4] L. Spitzner, "Honeypots: Catching the insider threat," in *Proc. 19th Annu. Comput. Security Appl. Conf.* , Las Vegas, NV, USA, 2003, pp. 170–179.

[5] O. S. Saydjari, "Multilevel security: Reprise," *IEEE Security Privacy*, vol. 2, no. 5, pp. 64–67, Sep./Oct. 2004.

[6] P. Loscocco, *Security-Enhanced Linux*, Linux 2.5 Kernel Summit, San Jose, CA, USA, 2001.

[7] X. Qian and T. F. Lunt, "A semantic framework of the multilevel secure relational model," *IEEE Trans. Knowl. Data Eng.*, vol. 9, no. 2, pp. 292–301, Mar./Apr. 1997.

[8] W.-P. Lu and M. K. Sundareshan, "A model for multilevel security in computer networks," *IEEE Trans. Softw. Eng.*, vol. 16, no. 6, pp. 647–659, Jun. 1990.

[9] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. 22nd Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, pp. 8–11, 2015.

[10] C. Yoon *et al.*, "Flow wars: Systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3514–3530, Dec. 2017.

[11] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks," in *Proc. Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, pp. 8–11, 2015.

[12] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2012, pp. 121–126.

[13] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "FLOWGUARD: Building robust firewalls for software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014, pp. 97–102.

[14] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, "Adversarial network forensics in software defined networking," in *Proc. ACM Symp. SDN Res.*, 2017, pp. 8–20.

[15] W. Dawoud, I. Takouna, and C. Meinel, "Infrastructure as a service security: Challenges and solutions," in *Proc. 7th Int. Conf. Inf. Syst. (INFOS)*, 2010, pp. 1–8.

[16] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Netw. Security*, vol. 2011, no. 8, pp. 16–19, 2011.

[17] M. Antikainen, T. Aura, and M. Särelä, "Spook in your network: Attacking an SDN with a compromised openflow switch," in *Proc. Nordic Conf. Secure IT Syst.*, 2014, pp. 229–244.

[18] D. Senie and P. Ferguson, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," IETF, RFC 2827, 1998.

[19] L. Savu, "Cloud computing: Deployment models, delivery models, risks and research challenges," in *Proc. Int. Conf. Comput. Manag. (CAMAN)*, Wuhan, China, 2011, pp. 1–4.

[20] S. Feghhi and D. J. Leith, "A Web traffic analysis attack using only timing information," *IEEE Trans. Inf. Forensics Security*, vol. 11, pp. 1747–1759, 2016.

[21] *OpenFlow Protocol*. Accessed: Nov. 10, 2015. [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow

[22] D. E. Bell and L. J. LaPadula, "Secure computer system: Unified exposition and Multics interpretation," NTIS, Springfield, VA, USA, Rep. ESD-TR-75-306, 1976.

[23] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Boston, MA, USA: Springer, pp. 85–103, 1972.

[24] *Gurobi*. Accessed: Jun. 1, 2020. [Online]. Available: http://gurobi.com

[25] A. Sabelfeld and D. Sands, "Declassification: Dimensions and principles," *J. Comput. Security*, vol. 17, no. 5, pp. 517–548, 2009.

[26] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *Proc. 13th Workshop Privacy Electron. Soc.*, 2014, pp. 121–130.

[27] *SDNmap Open Source Tool*. Accessed: Jun. 1, 2020. [Online]. Available: https://github.com/SDNMap

[28] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[29] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2002, pp. 133–145.

[30] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 267–280.

[31] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation," in *Proc. Int. Conf. Passive Active Netw. Meas.*, 2012, pp. 85–95.

[32] *Mininet—Realistic Virtual SDN Network Emulator*. Accessed: Nov. 6, 2017. [Online]. Available: http://mininet.org/

[33] L. Bauer, S. Garriss, and M. K. Reiter, "Detecting and resolving policy misconfigurations in access-control systems," *ACM Trans. Inf. Syst. Security*, vol. 14, no. 1, p. 2, Jun. 2011. [Online]. Available: https://doi.org/10.1145/1952982.1952984

[34] A. Wool, "A quantitative study of firewall configuration errors," *Computer*, vol. 37, no. 6, pp. 62–67, Jun. 2004.

[35] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2027–2051, 3rd Quart., 2016.

[36] O. Osanaiye, K.-K. R. Choo, and M. Dlodlo, "Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework," *J. Netw. Comput. Appl.*, vol. 67, pp. 147–165, May 2016.

[37] M. Yu, T. He, P. McDaniel, and Q. K. Burke, "Flow table security in SDN: Adversarial reconnaissance and intelligent attacks," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Toronto, ON, Canada, 2020, pp. 1519–1528.
[38] Q. Burke, P. McDaniel, T. L. Porta, M. Yu, and T. He, "Misreporting attacks in software-defined networking," in *Proc. 16th EAI Int. Conf. Security Privacy Commun. Netw. (EAI SecureComm)*, 2020, pp. 1519–1528.
[39] B. Harris and R. Hunt, "TCP/IP security threats and attack methods," *Comput. Commun.*, vol. 22, no. 10, pp. 885–897, 1999.
[40] F. B. Shaikh and S. Haider, "Security threats in cloud computing," in *Proc. Int. Conf. Internet Technol. Secured Trans.*, Abu Dhabi, UAE, 2011, pp. 214–219.
[41] V. Varadharajan and S. Black, "A multilevel security model for a distributed object-oriented system," in *Proc. 6th Annu. Comput. Security Appl. Conf.*, 1990, pp. 68–78.
[42] D. Volpano and G. Smith, "A type-based approach to program security," in *Colloquium on Trees in Algebra and Programming*. Heidelberg, Germany: Springer, 1997, pp. 607–621.
[43] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (SDN): A reference architecture and open APIS," in *Proc. Int. Conf. ICT Converg. (ICTC)*, Jeju Island, South Korea, 2012, pp. 360–361.
[44] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," *Presented as part of the 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 99–112.
[45] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," *Presented as part of the 9th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2012, pp. 113–126.
[46] C. Prakash *et al.*, "PGA: Using graphs to express and automatically reconcile network policies," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 29–42, 2015.
[47] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang, "Brew: A security policy analysis framework for distributed SDN-based cloud environments," *IEEE Trans. Depend. Secure Comput.*, vol. 16, no. 6, pp. 1011–1025, Nov./Dec. 2019.
[48] A. Abhashkumar, J.-M. Kang, S. Banerjee, A. Akella, Y. Zhang, and W. Wu, "Supporting diverse dynamic intent-based policies using janus," in *Proc. 13th Int. Conf. Emerg. Netw. EXp. Technol.*, 2017, pp. 296–309.
[49] B. Jiang, G. Huang, T. Wang, J. Gui, and X. Zhu, "Trust based energy efficient data collection with unmanned aerial vehicle in edge network," *Trans. Emerg. Telecommun. Technol.*, to be published.
[50] T. Li, W. Liu, T. Wang, Z. Ming, X. Li, and M. Ma, "Trust data collections via vehicles joint with unmanned aerial vehicles in the smart Internet of Things," *Trans. Emerg. Telecommun. Technol.*, to be published.
[51] Y. Ren, Z. Zeng, T. Wang, S. Zhang, and G. Zhi, "A trust-based minimum cost and quality aware data collection scheme in P2P network," *Peer-to-Peer Netw. Appl.*, vol. 13, pp. 2300–2323, Mar. 2020.
[52] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

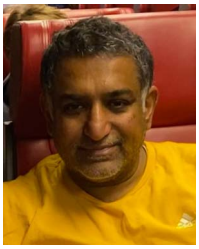
Stefan Achleitner received the B.S. and M.S. degrees in computer science from Vienna University of Technology, Austria, and the Ph.D. degree in computer science and engineering from Pennsylvania State University, University Park, PA, USA. He is currently a Security Researcher with the Threat and Application Research Group, Palo Alto Networks. His research interests include, software defined networking, network security, virtual machines, Internet of Things, and machine learning.

Quinn Burke (Student Member, IEEE) received the B.S. and M.S. degrees in computer science from Pennsylvania State University with a focus on computer security, where he is currently pursuing the Ph.D. degree in computer science. His research interests include network and systems security, software-defined networking, and virtualization technologies.

Patrick McDaniel (Fellow, IEEE) is the William L. Weiss Professor of Information and Communications Technology and the Director of the Institute for Networking and Security Research with the School of Electrical Engineering and Computer Science, Pennsylvania State University. Prior to joining Penn State in 2004, he was a Senior Research Staff Member with AT&T Labs-Research. His research focuses on a wide range of topics in computer and network security and technical public policy. He also served as the Program Manager and the Lead Scientist for the Army Research Laboratory's Cyber-Security Collaborative Research Alliance from 2013 to 2018. He is also a Fellow of ACM and AAAS and the Director of NSF Frontier Center for Trustworthy Machine Learning.

Trent Jaeger (Member, IEEE) is a Professor with the Computer Science and Engineering Department, Pennsylvania State University. He has authored the book *"Operating Systems Security,"* which examines the principles behind secure operating system designs. He has made a variety of contributions to the open-source security community, particularly to the Linux operating system. His research interests include systems and software security, on which he has published over 150 journal and conference papers. He serves on the Executive Committee of the ACM Special Interest Group on Security, Audit, and Control, the Steering Committee Chair for the Network and Distributed Systems Security Symposium, and an Editorial Board Member for the Communications of the ACM and IEEE Security and Privacy.

Thomas La Porta (Fellow, IEEE) received the B.S.E.E. and M.S.E.E. degrees from the Cooper Union, New York, NY, USA, and the Ph.D. degree in electrical engineering from Columbia University, New York. He is the Director of the School of Electrical Engineering and Computer Science, Penn State University, where he is an Evan Pugh Professor and the William E. Leonhard Chair Professor with the Computer Science and Engineering Department and the Electrical Engineering Department. He joined Penn State in 2002. He was the Founding Director of the Institute of Networking and Security Research, Penn State. Prior to joining Penn State, he was with Bell Laboratories for 17 years. He was the Director of the Mobile Networking Research Department, Bell Laboratories, Lucent Technologies, where he led various projects in wireless and mobile networking. He received the Bell Labs Distinguished Technical Staff Award, and an Eta Kappa Nu Outstanding Young Electrical Engineer Award. He also won two Thomas Alva Edison Patent Awards. He was the founding Editor-in-Chief of the IEEE TRANSACTIONS ON MOBILE COMPUTING. He served as the Editor-in-Chief of *IEEE Personal Communications Magazine*. He was the Director of Magazines for the IEEE Communications Society and was on its Board of Governors for three years. He is an Bell Labs Fellow.



Srikanth Krishnamurthy (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of California at San Diego in 1997. From 1998 to 2000, he was a Research Staff Scientist with Information Sciences Laboratory, HRL Laboratories, LLC, Malibu, CA, USA. He is currently a Professor of Computer Science with the University of California Riverside. His research interests are in network, computer system, and ML security, and computer and wireless networks. He is the recipient of the NSF CAREER Award from ANI in 2003. He was the Editor-in Chief for ACM MC2R from 2007 to 2009 and is currently the Associate Editor-in-Chief for the IEEE TRANSACTIONS ON MOBILE COMPUTING.