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 paper, we propose a novel multilevel security (MLS) framework to enforce a secure inter-node information flow policy within the network and thereby 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 nodes in the network. 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 route flows that satisfy the security constraints (e.g., > 80% of flows in a performed benchmark) and route the remaining flows with a minimal security cost.

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

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 et al. [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 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 paper, we propose an SDN-based MLS frame-
work 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 \textit{policy conflict}, and each conflict imposes a \textit{security cost} in terms of the risk the flow is being exposed to.

Unlike prior work \cite{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.

\begin{table}[ht]
\centering
\caption{Nomenclature and notation.}
\begin{tabular}{|c|l|}
\hline
\textbf{Notation} & \textbf{Description} \\
\hline
$V$ & 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 vertices in network graph \\
$O$ & Set of subjects \\
$C$ & Set of security categories \\
$\lambda_i^c$ & Security category $c$ at node $i$ \\
$L$ & Set of labels that form the lattice \\
\hline
\end{tabular}
\end{table}

\section{Definitions and Background}

In this section, we extend prior work’s \cite{8} terms and notations (Table 1) to an SDN setting and provide background on related network security threats and defenses, and MLS.

\subsection{Term Definitions}

\textbf{Node.} A resource connected to a network (e.g., a user, server, router, or SDN switch).

\textbf{Subject.} A node that initiates communication to other nodes in the network.

\textbf{Object.} A node that either provides (\textit{provider}) and/or receives (\textit{receiver}) information to/from subjects

\textbf{Forwarding Node.} A node (SDN switch) that processes incoming flows according to the installed flow rules.

\textbf{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.

\textbf{Security Levels.} Hierarchical attributes (e.g., top-secret, public) that indicate relative authorization power.

\textbf{Security Categories.} Non-hierarchical attributes (e.g., TCP, IP) that offer finer-grained authorization besides the security level. In MLSNet, security categories are only assigned to objects and subjects but not forwarding nodes.

\textbf{Security Label.} The security level and categories combined, used by the controller to admit or deny flows.

\subsection{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.

Recent work has demonstrated the ability of an adversary to freely probe within their network boundary to recover sensitive information about the network \cite{9,10,11,12,13}, including active network hosts and even switch flow table rules \cite{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, 18].

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 [19] in wide-area networks; however, it is difficult to implement, especially in data-center networks [20], 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 [21].

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 et al. [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. 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, where only explicitly allowed communication can flow within the network and must flow through secure paths. Hence, it provides for maximal isolation from unauthorized parties and therefore the smallest possible threat surface. Here, we can mitigate lateral movement (by enforcing least-privilege), eavesdropping (with secure routing paths), and packet spoofing (blocking unknown sources) by unauthorized entities—a significant improvement over prior work (see Table [1]). We note that compromised nodes may still be able to probe or eavesdrop nodes for which they have sufficient security level and categories; however, their capabilities are significantly restricted to only that allowed by policy, and they can quickly be quarantined (via flow rules) upon detection.

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 et al. [8], who assigns appropriate security labels (i.e., levels and categories) to the network entities (e.g., endpoint devices and forwarding nodes). 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 [22] 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.

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 [23]. 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:

\[ \sigma_o \leq \sigma_s \text{ and } C_o \subseteq C_s, \forall o \in O, s \in S \]  

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:

\[ \sigma_o \geq \sigma_s \text{ and } C_o \supseteq C_s, \forall o \in O, s \in S \]  

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:

\[ \sigma_o = \sigma_s \text{ and } C_o = C_s, \forall o \in O, s \in S \]  

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 \( \sigma \) 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:

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

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:

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

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 \( o \):

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

In addition to the security labels, we also must consider the capacity \( \kappa_{ij} \) 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-E].

### 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 \( f \). Further, we propose a second ILP-based optimization model which minimizes the sum of policy conflict values, given a topology and security labels, under the assumption that all flows \( f \in F \) permitted by the access control constraint must be accommodated.

#### A. Policy Compliant Flow Maximization Problem

In this section, we introduce an 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 \( x^f_{ij} \) to indicate if link \( (i,j) \in E \) is used on the path for flow \( f \) (i.e., \( x^f_{ij} = 1 \)) or not (i.e., \( x^f_{ij} = 0 \)). To decide if a flow \( f \) can be accommodated, we also introduce the binary decision variable \( \alpha^f \).

In \( 7b \), we add \( \alpha^f \) to the link indication variable \( x^f_{ij} \) 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 \( \alpha^f \) 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.

\[
\max \sum_{f \in F} \alpha^f \tag{7a}
\]

\[
s.t.
\sum_{i,(i,s) \in E} x^f_{is} + \alpha^f = \sum_{j,(s,j) \in E} x^f_{sj}, \forall f \in F \tag{7b}
\]

\[
\sum_{i,(i,j) \in E} x^f_{ij} - \alpha^f = 0, \forall f \in F \tag{7c}
\]

\[
\sum_{i,j \in E} x^f_{ij} = \sum_{j,k \in E} x^f_{jk}. \forall f \in F \tag{7d}
\]

\[
\sum_{i,(i,j) \in E} x^f_{ij} \leq 1, \forall f \in F, j \in V \tag{7e}
\]

\[
\sum_{f \in P} x^f_{ij} \cdot d^f \leq \kappa_{ij}, \forall (i,j) \in E \tag{7f}
\]

\[
\alpha^f \cdot \text{lev}(\sigma_o, \sigma_s) = \alpha^f, \forall o(f) \in O, s(f) \in S, f \in F \tag{7g}
\]

\[
\alpha^f \cdot \text{cat}(\lambda_o^c, \lambda_s^c) = \alpha^f, \forall o(f) \in O, s(f) \in S, c \in C, f \in F \tag{7h}
\]

\[
x^f_{ij} \cdot \text{orig}(\sigma_o, \sigma_s) \leq x^f_{ij} \cdot \sigma_j, \forall (i,j) \in E, o(f) \in O, s(f) \in S, f \in F \tag{7i}
\]

\[
x^f_{ij} \in \{0, 1\}, \forall (i,j) \in E, f \in F \tag{7j}
\]

\[
\alpha^f \in \{0, 1\}, \forall f \in F \tag{7k}
\]

We add constraint \( 7c \) to limit the number of visits of a node to one for each flow. Constraint \( 7i \) ensures that the given capacity \( \kappa_{ij} \) 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 \(\kappa_{ij}\) with the residual capacity \(\bar{\kappa}_{ij}\) 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 \(\text{lev}(\sigma_o, \sigma_s)\), shown in (3), returns 1, indicating that the security levels of \(s\) and \(o\) allow communication:

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

(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 \(\text{cat}(\lambda^c_o, \lambda^c_s)\) shown in (9), we model the requirement of security categories to allow a flow between \(s\) and \(o\):

\[
\text{cat}(\lambda^c_o, \lambda^c_s) = \begin{cases} 
1 - (\lambda^c_o - \lambda^c_s), & \text{if } o \text{ is provider} \\
1 - (\lambda^c_o - \lambda^c_s), & \text{if } o \text{ is receiver} \\
1 - (\lambda^c_s - \lambda^c_o), & \text{if } o \text{ is both} \\
0, & \text{otherwise} 
\end{cases}
\]  

(9)

To mathematically define this, we introduce variable \(\lambda^c_i\) which indicates if a node \(i\) has a security category \(c\), i.e., \(\lambda^c_i = 1\), or not, i.e., \(\lambda^c_i = 0\). As an example for the operation of function \(\text{cat}()\), 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 \(1 - (\lambda^c_o - \lambda^c_s)\) as shown in (3). This will evaluate to 0 if subject \(s\) does not have a security category \(c\), but object \(o\) does, i.e., \((1 - (1-1-0)) = 0\). Such a case does not fulfill the access control constraint, and therefore the flow cannot be admitted, i.e., \(\alpha_f = 0\).

Function \(\text{cat}()\) works in a similar way if \(o\) is a receiver object. In case \(o\) is both a provider and receiver, function \(\text{cat}()\) evaluates to 1 if \(C_o = C_s\). As stated in constraint (7b) the function \(\text{cat}()\) 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 (7h) 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 \(x^f_{ij}\), 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 \(\text{orig}(\sigma_o, \sigma_s)\) as shown in (10), where \(\text{orig}()\) returns the security level depending on the type of object node \(o\), according to the defined flow control constraint in Section III-C.

\[
\text{orig}(\sigma_o, \sigma_s) = \begin{cases} 
\sigma_o, & \text{if } o \text{ is receiver} \\
\sigma_o, & \text{otherwise} 
\end{cases}
\]  

(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 (7g) and (7h) as the basis for our heuristic-based maximization algorithm discussed in the next section.

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 [24]. Although solvers such as Gurobi [25] 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. Algorithm (1) is a modification of Dijkstra’s shortest path algorithm where we add the access control and secure flow control constraints, similar to the constraints presented in (7). Specifically, we formulate the access control constraint in line 6 based on the introduced functions \(\text{lev}()\) as defined in (8) and \(\text{cat}()\) 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.

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., the sender node), and we quantify a policy conflict as the numerical difference between those security levels. 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 \( F_1 \subseteq F \) that fulfill the access control constraints, and second, compute paths between subjects and objects with the objective to minimize security policy conflicts. We define the access control constraints as follows:

\[
F_1 = \{ f \in F : \text{lev}(\sigma_o, \sigma_s) = 1 \text{ and } \text{cat}(\lambda_o^s, \lambda_s^o) = 1, \forall \lambda_o^s \in C_o, \forall \lambda_s^o \in C_s, o \in O, s \in S \} \tag{11}
\]

Next, for the set of legitimate flows \( F_1 \), we also define an objective function, \( conf \), to find a network configuration that accommodates all flows in \( F_1 \) while minimizing the policy conflicts on a path of a flow \( f \in F_1 \) 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 \( \text{orig}(\sigma_o, \sigma_s) \), and the security level \( \sigma_j \) of a node \( j \) on the path between \( s \) and \( o \) if \( \sigma_j < \text{orig}(\sigma_o, \sigma_s) \). More formally:

\[
\text{conf}(\sigma_o, \sigma_s, \sigma_j) = \begin{cases} 
\text{orig}(\sigma_o, \sigma_s) - \sigma_j, & \text{if } \sigma_j < \text{orig}(\sigma_o, \sigma_s) \\
0, & \text{otherwise}
\end{cases} \tag{12}
\]

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 \( \sigma_o \) of node \( o \) to a node \( j \) with level \( \sigma_j \), 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. For selecting a secure path, we want to give preference to those 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 \( \gamma \) and define our objective function as follows:

\[
\min \sum_{f \in F_1} \sum_{j \in E} x_{ij} \cdot \gamma^{\text{conf}(\sigma_o(f), \sigma_s(f), \sigma_j)} \tag{13}
\]

In (13), \( x_{ij} \) 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 \( \sigma_j \). The factor \( \gamma \) 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 \( 4^4 - 4 = 16 \), 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 \( \gamma \) (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 \( \gamma \) controls the selection of longer paths visiting nodes with a smaller security level difference. To always select paths with the smallest security level difference, \( \gamma \) 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 \( \gamma \) 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 \( x_{ij} \), the value of policy conflicts specified as \( \sum_{i,j \in E} x_{ij} \cdot \gamma^{\sigma_o - \sigma_j} \) and

\[\text{Fig. 3. Path selection based on security levels.}\]
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 \( \gamma \) is chosen appropriately. Given a candidate node for the path of flow \( f \) with a policy conflict of \( a \), we want to choose \( \gamma \) 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 \( \gamma^a \cdot \gamma^q > y \cdot \gamma^b \). 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 \( \gamma \) 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 \( x_{ij}^f \). 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 \( \kappa_{ij} \) of a link \((i,j)\in E\) is not exceeded by forwarding flows \( f \in F \) with a size of \( d_f^j \) per flow. In (14e) we assume that the flow size \( d_f^j \) 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 \( \kappa_{ij} \) with the residual capacity \( \tilde{\kappa}_{ij} \) 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 \( f \in F \) between a subject node \( s(f) \) and an object node \( o(f) \) with the objective function as defined in (14a).

\[
\min \sum_{f \in F} \sum_{i,j \in E} x_{ij}^f \cdot c^{\text{conf}}(\sigma_a(f), \sigma_s(f), \sigma_j)
\]

\[
\text{s.t.}
\]
\[
\sum_{i,(s,j) \in E} x_{ij}^f + 1 = \sum_{j,(s,j) \in E} x_{ij}^f, \quad \forall f \in F
\]
\[
\sum_{j,(o,j) \in E} x_{ij}^f - 1 = 0, \quad \forall f \in F
\]
\[
\sum_{i,j \in E} x_{ij}^f = \sum_{j,k \in E} x_{jk}^f, \quad \forall f \in F
\]
\[
\sum_{f \in F} x_{ij}^f \cdot d_f^j \leq \kappa_{ij}, \quad \forall (i,j) \in E
\]
\[
x_{ij}^f \in \{0, 1\}, \quad \forall (i,j) \in E, f \in F
\]

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. Subfeldt et al. [26] 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 camouflage techniques [27] 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 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 [22] 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 (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. More formally, for secure rule construction, we represent the set of fields supported for network layer $N_i \in N$, where $N$ is the set of all network layers, as $R_{N_i}$. 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 $R$

$$R = \bigcup_{N_i \in N : \forall N_j \leq N_i} R_{N_i}$$

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 \texttt{next}, which specifies 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 of a flow with specific protocol types, it must be ensured that the priority of that rule is higher than other rules for the flow which may allow forwarding for other protocol types (i.e., allowing most IP traffic, but disallowing any UDP over IP).

To resolve any flow rule conflicts between rule actions, we refer to existing frameworks such as Porras et al. [12]. 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 are able to properly generate a secure flow rule configuration and isolate the flows to ensure that confidentiality of information flow is preserved in the network.

```
Algorithm 3 GenerateFlowRule($R, A, P, next$)
1: rule.append("match :")
2: for all $r \in R$ do
3:   if $r \in P$ then
4:     rule.append($r = P(r)$)
5:   end if
6: end for
7: rule.append("action :")
8: if next != drop then
9:   for all $a \in A$ do
10:     rule.append($a$)
11:   end for
12: rule.append("next")
13: else
14:   rule.append("drop")
15: end if
16: return rule
```
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. 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 [28], [14].

A. Accommodating Network Flows

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) and heuristic algorithms (HA) on different network topologies. We first consider a realistic autonomous system (AS) network topology. Then, to model common data-center and cloud topologies, we consider different k-ary fat-tree networks [29], where $k$ is the port density of each switch in the network (e.g., 8, 12, and 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 [30] 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 any of the topologies, the number of flows routed reached, at a maximum, approximately 85% coverage. We also observe, 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 (here, randomly chosen), node labels (here, random), and the 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 in fact feasible in several network topologies of different sizes.

In the case of congested networks, we observe that the linear programming models show slightly better performance in terms of flow assignment compared to the heuristic algorithms. We note that despite the low flow coverage (∼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 randomness used in generating the networks did not hinder the flow coverage. Specifically, in real data-center or cloud networks, edge switches may carry similar traffic [31] 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.

Given that a majority of flows can be routed according to policy, we 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. Investigating the reasons behind this is an interesting direction for future work, where it may be possible to identify whether or not this 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.

We observe that the execution time of computing secure paths is strongly correlated with the number of switches in the network, scaling exponentially. We find that both the LP solver (Gurobi [25]) and the greedy algorithms can quickly compute the paths for a few hundred flows in a 8-ary tree on average between 0.5s to 1s in a Python-based implementation. When routing thousands of flows for the same network, the execution time can increase to between 7-8s, where the greedy algorithm performs faster (approximately 3.5-4s) as the number of flows increases further. However, large data-centers may
be composed of tens of thousands of nodes \cite{29}, and even mid-sized data-centers may contain several hundred or up to one-thousand nodes. In the latter case (for example, represented by a 16-ary fat-tree), solving the optimization can take nearly 45s. In enterprise-grade networks, the computation speed of MLSNet can be further improved by applying buffering of precomputed paths, implementation on hardware, or clustering of flows which is especially applicable in fat-tree networks.

Overall, the results show strong evidence for the ability of our framework to accommodate flows in real networks, while preserving confidentiality of information flow.

\textbf{B. Defending Against Attacks}

We then implement the network scenario shown in Figure \ref{fig:scenario} with the SDN simulator Mininet \cite{32}, assigning security labels to the nodes and configuring the network to use our framework MLSNet. Here, we use SDNMap \cite{28} to demonstrate that our framework can mitigate the attack techniques discussed in Section \ref{sec:attacks}. 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 (identify 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 \ref{fig:lattice} to configure the network. Running the MLSNet system at the SDN controller, we assign the security classification of \textit{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.

\textbf{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 \textit{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. 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 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, are not discovered by the attacker.

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 could not obtain any other information about the other hosts or protocols supported by them, and their capabilities were only limited to scanning public nodes over the protocols allowed by the defined security label (here, ARP, IP, and TCP).

Indeed, future work may investigate optimal labeling and relabeling strategies that can respond to the current network traffic profile in order to dynamically enforce least-privilege and further reduce the threat surface.

\textbf{Mitigating packet spoofing.} With respect to packet spoofing,
ing, we observe that MLSNet can significantly reduce the threat surface. SDNMap reports that it is not able to spoof IP addresses from hosts which can’t be discovered, since our framework will prevent the deployment of a rule for an unrecognized node, and therefore additional flow rule features cannot be reconstructed. By including information from all network layers (as discussed in Section V) and deploying rules that only allow flows according to the given policy, we are able to prevent attacks resulting from these forms of spoofing.

Although, we note again that MLS cannot detect all forms of packet spoofing, and we rely on other SDN-based defenses to detect packet spoofing of existing nodes or against the topology discovery service [9]. Despite this, it can mitigate the threat of data exfiltration from packet spoofing with secure routing paths. For example, consider a compromised node with a lower security level (e.g., public) spoofing a node of higher level (e.g., top-secret) to communicate with another top-secret node, but through two other public nodes. The adversary would have to compromise both public nodes in between in order for them to actually forward the top-secret flow—as it is against security policy.

Mitigating eavesdropping. In addition to mitigating the threat of lateral movement, MLSNet also prevents the attacker from eavesdropping on communication between nodes of higher-classification. Since the controller prevents flow rules that direct higher-level traffic toward public nodes from being generated, the attacker was unable to capture any traffic passing between the secret and confidential nodes at 10.0.0.4 and 10.0.0.6. Further, any responses induced from the probes were also blocked from generating a new flow rule to direct traffic toward the attacker, preventing eavesdropping. As a result, MLSNet is able to significantly limit the ability of SDNMap to reconstruct flow rules.

**VII. RELATED WORK**

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 [33], [34], [3]. 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, and denial-of-service [35], [36], [37]. 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 [38] and have become increasingly important as more information is being put online [39]. 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 [19], [20], firewalls 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. This motivates our work for developing a solution to combine the benefits offered by each of these methods. We exploit multilevel security to accomplish this, providing guarantees about who in the network may access what data.

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 [40], table access in a relational database [7], as well as a primitive for securing information flow between variables in programming languages [41].

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 et al. [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) [42] to provide this service transparently to the entire network.

**VIII. CONCLUSION**

In this paper, 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 scope conflicts in real-time.

REFERENCES

[1] J. Jang-Jaccard and S. Nepal, “A survey of emerging threats in cybersecurity,” Journal of Computer and System Sciences, vol. 80, no. 5, pp. 973–993, 2014.
[2] D. E. Denning, “A lattice model of secure information flow,” Communications of the ACM, 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 2006 IEEE Symposium on Security and Privacy (S&P’06). IEEE, 2006, pp. 15–pp.
[4] L. Spitzner, “Honeypots: Catching the insider threat,” in 19th Annual Computer Security Applications Conference, 2003. Proceedings. IEEE, 2003, pp. 170–179.
[5] O. S. Saydjari, “Multilevel security: reprise,” IEEE security & privacy, vol. 2, no. 5, pp. 64–67, 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 Transactions on Knowledge and Data Engineering, vol. 9, no. 2, pp. 292–301, 1997.
[8] W.-P. Lu and M. K. Sundareshan, “A model for multilevel security in computer networks,” IEEE Transactions on Software Engineering, vol. 16, no. 6, pp. 647–659, 1990.
[9] S. Hong, L. Xu, H. Wang, and G. Gu, “Poisoning network visibility in software-defined networks: New attacks and countermeasures.” in NDSS, 2015.
[10] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, “Flow wars: Systemizing the attack surface and defenses in software-defined networks,” IEEE/ACM Transactions on Networking (TON), vol. 25, no. 6, pp. 3514–3530, 2017.
[11] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, “Sphinx: Detecting security attacks in software-defined networks.” in NDSS, 2015.
[12] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, “A security enforcement kernel for openflow networks,” in HotSDN. ACM, 2012.
[13] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, “Flowguard: building robust firewalls for software-defined networks,” in HotSDN. ACM, 2014.
[14] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, “Adversarial network forensics in software defined networking.” in Proceedings of the 2017 ACM Symposium on SDN Research.
[15] W. Dawoud, I. Takouna, and C. Meinel, “Infrastructure as a service security: Challenges and solutions,” in 2010 the 7th International Conference on Informatics and Systems (INFOS). IEEE, 2010, pp. 1–8.
[16] C. Tankard, “Advanced persistent threats and how to monitor and deter them,” Network security, vol. 2011, no. 8, pp. 10–19, 2011.
[17] M. Antikainen, T. Aura, and M. Sareli, “Spook in your network: Attacking an sdn with a compromised openflow switch,” in Nordic Conference on Secure IT Systems. Springer, 2014, pp. 229–244.
[18] S. Khan, A. Gani, A. W. A. Wahab, M. Guizani, and M. R. Khan, “Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art,” IEEE Communications Surveys & Tutorials, vol. 19, no. 1, pp. 303–324, 2017.
[19] D. Senie and P. Ferguson, “Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing,” Network, 1998.
[20] L. Savu, “Cloud computing: Deployment models, delivery models, risks and research challenges,” in 2011 International Conference on Computer and Management (CAMAN). IEEE, 2011, pp. 1–4.
[21] S. Feghhi and D. J. Leith, “A web traffic analysis attack using only timing information,” IEEE Transactions on Information Forensics and Security, vol. 11, no. 8, pp. 1747–1759, 2016.
[22] “Openflow,” https://www.opennetworking.org/sdn-resources/openflow.
[23] D. E. Bell and L. J. LaPadula, “Secure computer system: Unified exposition and Multics interpretation,” Tech. Rep. ESD-TR-75-506, 1976.
[24] R. M. Karp, “Reducibility among combinatorial problems,” in Complexity of computer computations. Springer, 1972.
[25] “Gurobi,” http://gurobi.com.
[26] A. Sabelfeld and D. Sands, “Declassification: Dimensions and principles,” Journal of Computer Security, 2009.
[27] X. Cai, R. Nithyanand, and R. Johnson, “Cs-buflo: A congestion sensitive website fingerprinting defense,” in Proceedings of the 13th Workshop on Privacy in the Electronic Society. ACM, 2014.
[28] “Sdnmap open source tool,” https://github.com/SDNMap

13
[29] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in ACM SIGCOMM Computer Communication Review, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[30] N. Spring, R. Mahajan, and D. Wetherall, “Measuring isp topologies with rocketfuel,” ACM SIGCOMM Computer Communication Review, 2002.

[31] T. Benson, A. Akella, and D. A. Maltz, “Network traffic characteristics of data centers in the wild,” in Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010, pp. 267–280.

[32] “Mininet - realistic virtual sdn network emulator,” [http://mininet.org/], accessed: 2017-11-06.

[33] L. Bauer, S. Garriss, and M. K. Reiter, “Detecting and resolving policy misconfigurations in access-control systems,” ACM Trans. Inf. Syst. Secur., vol. 14, no. 1, 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, 2004.

[35] M. Conti, N. Dragoni, and V. Lesyk, “A survey of man in the middle attacks,” IEEE Communications Surveys & Tutorials, vol. 18, no. 3, pp. 2027–2051, 2016.

[36] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in 2010 IEEE symposium on security and privacy. IEEE, 2010, pp. 305–316.

[37] O. Osanaiye, K.-K. R. Choo, and M. Dlodlo, “Distributed denial of service (ddos) resilience in cloud: Review and conceptual cloud ddos mitigation framework,” Journal of Network and Computer Applications, vol. 67, pp. 147–165, 2016.

[38] B. Harris and R. Hunt, “Tcp/ip security threats and attack methods,” Computer communications, vol. 22, no. 10, pp. 885–897, 1999.

[39] F. B. Shaikh and S. Haider, “Security threats in cloud computing,” in 2011 International conference for Internet technology and secured transactions. IEEE, 2011, pp. 214–219.

[40] V. Varadharajan and S. Black, “A multilevel security model for a distributed object-oriented system,” in [1990] Proceedings of the Sixth Annual Computer Security Applications Conference. IEEE, 1990, pp. 68–78.

[41] D. Volpano and G. Smith, “A type-based approach to program security,” in Colloquium on Trees in Algebra and Programming. Springer, 1997, pp. 607–621.

[42] M.-K. Shin, K.-H. Nam, and H.-J. Kim, “Software-defined networking (sdn): A reference architecture and open apis,” in 2012 International Conference on ICT Convergence (ICTC). IEEE, 2012, pp. 360–361.