Hindawi
Security and Communication Networks
Volume 2018, Article ID 9178425, 14 pages
https://doi.org/10.1155/2018/9178425

Research Article

NOSArmor: Building a Secure Network Operating System

Hyeonseong Jo, Jaehyun Nam, and Seungwon Shin

KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea

Correspondence should be addressed to Seungwon Shin; claude@kaist.ac.kr

Received 15 September 2017; Accepted 11 January 2018; Published 20 February 2018

Academic Editor: Danda B. Rawat

Software-Defined Networking (SDN), controlling underlying network devices (i.e., data plane) in a logically centralized manner, is now actively adopted in many real world networking environments. It is clear that a network administrator can easily understand and manage his networking environments with the help of SDN. In SDN, a network operating system (NOS), also known as an SDN controller, is the most critical component because it should be involved in all transactions for controlling network devices, and thus the security of NOS cannot be highly exaggerated. However, in spite of its importance, no previous works have thoroughly investigated the security of NOS. In this work, to address this problem, we present the NOSArmor, which integrates several security mechanisms, named as *security building block* (SBB), into a consolidated SDN controller. NOSArmor consists of eight SBBs and each of them addresses different security principles of network assets. For example, while role-based authorization focuses on securing confidentiality of internal storage from malicious applications, OpenFlow protocol verifier protects availability of core service in the controller from malformed control messages received from switches. In addition, NOSArmor shows competitive performance compared to existing other controllers (i.e., ONOS, Floodlight) with secureness of network assets.

1. Introduction

Software-Defined Networking (SDN), decoupling the control plane from the data plane and enabling dynamic network control, is now widely accepted in both industry and academia. For example, Google and Facebook now employ this new technique (i.e., SDN) to improve the performance of their networking environments [1, 2] and Amazon has also significantly reduced the operational cost by adopting it [3]. Likewise, SDN is no more a buzzword but a feasible and impactful technology.

In SDN, the most critical component will be its control plane (also known as a network operating system or an SDN controller), because this control plane determines how to manage all underlying networking environments. In this context, the security of the control plane cannot be overestimated, because if the control plane is compromised or attacked, it will cause serious effects on the target SDN network (e.g., forward network flows to an unauthorized host).

Indeed, the security of SDN control plane has been already discussed by several pioneering researchers [4], and they have shown possible attack scenarios in SDN networks and presented countermeasures to deal with corresponding attack scenarios. For example, TopoGuard [5], Avant-guard [6], and FloodGuard [7] introduce attacks against SDN data plane and suggest defending methods. However, note that they mostly consider some specific SDN asset instead of protecting all (or most) SDN assets. TopoGuard addresses security issues related to network topology information, whereas Avant-guard and FloodGuard handle control channel security issue. In the case of SDNShield [8], SE-Floodlight [9], and Rosemary [10], they mainly address application layer security issues, but they do not consider other SDN assets. This trend implies that SDN security cannot be ensured with previous studies, dealing with different security concerns, because *security is only as strong as the weakest link*.

In this paper, we introduce the NOSArmor as a security-enhanced SDN controller that integrates multiple security mechanisms that protect security principles of network assets against attack vectors existing in SDN network. We first define eight network assets, which should be kept secure from attack vectors, and match them to related security principles in terms of CIA triad [11] based on security

requirements defined by open networking foundation (ONF) [12]. For example, internal storage stores myriad of network-related information (i.e., host, link), one of network assets, for network programmability and can be manipulated from both the application layer and the data plane. Therefore, controllers should guarantee confidentiality and integrity of this information. In addition, in order for SDN network to operate, core system of controllers should operate at all times and resources (i.e., CPU, memory) should be required to operate the core system. As a result, controllers also ensure availability of software, resource.

We present eight security mechanisms (we call them SBBs) that are utilized in NOSArmor to address security principles of network assets, and they are classified into two types depending on what attack vectors they address. Five SBBs address security issues originated from the application layer and three SBBs address security issues originated from the data plane. In this work, we focus on dealing with security issues affecting controller itself, not considering security issues related to a control channel between a controller and switches since transport layer security (TLS) can ensure confidentiality and integrity of control messages being transmitted through a control channel.

To summarize, the contributions of this paper are as follows:

(i) We carefully investigate the security issues of SDN and analyze them based on the basic security analysis method (i.e., CIA triad).

(ii) Based on our analysis, we design and implement a prototype of NOSArmor, which consists of eight SBBs for protecting different security principles of network assets.

(iii) We evaluate the NOSArmor under a range of attack scenarios to verify its effectiveness and to show that performance overhead caused by SBBs is small enough.

The remainder of the paper is structured as follows. Section 2 provides a brief overview of SDN and motivating examples. In Sections 3 and 4, we present network assets and system design to protect security principles of them, respectively. Next, Section 5 provides implementation of NOSArmor and evaluation for effectiveness and performance of SBBs. We discuss system limitation, comparison between NOSArmor and other SDN controllers (Section 6) and related work (Section 7). Finally, Section 8 concludes this work.

2. Background and Motivation

2.1. Software-Defined Networking (SDN). Unlike legacy network that both control and data plane are tightly coupled within a single network device, SDN architecture decouples the control plane from the data plane of switches and enables the control plane, referred to as a controller, to control multiple data plane elements at a centralized point. This centralized architecture provides network operators with two key features, global network view, and network programmability. Basically, all of the switches in the data plane should be connected to controllers to request control logic for packets. Therefore, controllers can construct network topology by communicating with connected switches and collect statistics information by sending a request to each switch. In addition, controllers provide application developers with lots of APIs related to reading network status and enforcing forwarding-related command to the switches, which enables them to implement any network functions they want.

2.2. Motivating Examples. While SDN architecture brings many benefits to network operators, centralized control plane exposes several high-value network assets to emerging threat vectors which do not exist in legacy network [13–15]. Recent studies have shown possible attack scenarios and proposed security mechanisms to address distinct security issues related to network assets [4–6, 8–10].

However, the problem is that they focus on specific problems of network assets. In this section, we motivate the need for why controllers should consider multiple security issues at the same time through prior works that present partial solutions to the security issues.

2.2.1. A Security Issue from the Data Plane. Controllers construct network topology based on OpenFlow messages received from switches with belief that network entities (e.g., host, switch) are benign. However, Hong et al. have shown topology poisoning attack that could be caused by malicious hosts and proposed TopoGuard as a countermeasure to prevent them from distorting host location information included in the internal storage [5].

Figure 1 illustrates how a malicious host manipulates host location stored in internal storage. First, (1) an attacker B sends forged packets containing the same source IP and MAC address as web server C to a connected switch. (2) Since the forged packets came from a port other than where the web server was connected, the switch triggers a new `PACKET_IN` message. (3) When a controller receives the `PACKET_IN` message from the switch, it compares host location included in the internal storage with it of the `PACKET_IN` message. In case of mismatched host location, the controller updates the host location with the location contained in the `PACKET_IN` message and (4) it inserts a flow rule to forward all incoming packets to the attacker. Then, (5) the switch forwards packets going to the web server into the attacker.

While TopoGuard focuses on securing the control plane from malicious hosts, SDNShield [8], SE-Floodlight [9], and Rosemary [10] mainly address security concerns against malicious applications. In the case of Avant-guard [6] and FloodGuard [7], they deal with a security concern of a different network asset (i.e., control channel) from TopoGuard. As a result, controllers applied security mechanisms proposed by these works may make wrong routing decisions due to erroneous host location information.

2.2.2. A Security Issue from the Application Layer. One of security issues originated from malicious applications is enforcing a new flow rule (i.e., candidate flow rule) that

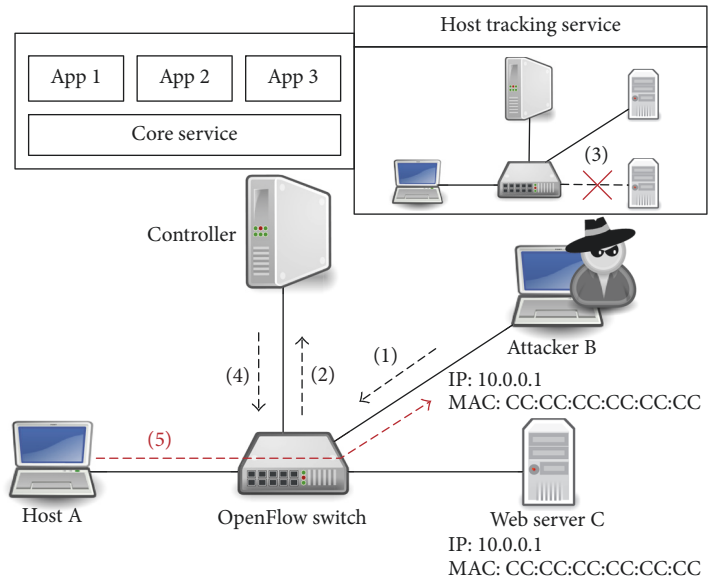

Figure 1: Topology poisoning attack scenario.

violates flow rules already inserted in switches. Since multiple applications for different purposes (e.g., load balance, security) are involved in determining routing paths of underlying switches, there is a possibility of rule conflict between a candidate flow rule and inserted flow rules. Porras et al. first present rule conflict detection and resolution mechanism to restrict malicious applications to enforce network policy that violates inserted flow rules by other applications [4].

However, SDNShield [8] and Rosemary [10], which deal with application layer security issues, do not address flow rule conflict issue either. Therefore, a security hole related to flow rule remains in SDN networks in which these controllers operate.

2.3. Problem Statement. As explained in the motivating examples, the previous studies mainly focus on handling specific security issues of network assets, which enables attackers to compromise controllers through vulnerabilities they cannot cover. Besides, since each study extends different open source controllers (e.g., Floodlight [16]) or designs a new controller (i.e., Rosemary [10]) to add new security features, it is hard to integrate multiple security mechanisms into a single controller. For example, Rosemary is a C application, whereas SE-Floodlight is an extension of Floodlight written in Java. Therefore, it is burdensome for network operators to port their security features to each other's controller without any modifications.

3. Network Asset

Due to physical separation of the control plane from the data plane, new attack vectors begin to emerge that do not exist in legacy network, which may severely affect confidentiality as well as integrity and availability of network assets. Since all of network assets are involved in routing decision of network applications directly or indirectly, if any security principles of network assets are being compromised, entire network may not operate normally.

Table 1: Network assets and related security principles.

Type	Network asset	C	I	A
Data	Host	√	√	-
	Link	√	√	-
	Device	√	-	-
	Statistics	√	-	-
	Flow rule	√	√	-
Software	Core system	-	-	√
Resource	CPU	-	-	√
	Memory	-	-	√

Therefore, we first define eight network assets that must be secure in SDN networks and classify them to related security principles based on security requirements defined by ONF [12]. In this paper, we focus on the following security principles of network assets: (i) *confidentiality*, a measure of whether controllers provide access control for sensitive network information, (ii) *integrity*, a measure of whether the sensitive network information should not be modified without agreement of controllers, and (iii) *availability*, a measure of whether controllers provide seamless network services.

Table 1 enumerates eight network assets depending on types of assets. First, in the case of data-related assets, since network information stored in internal storage of controllers is constructed based on incoming control messages sent from switches and is used by applications through APIs provided by controllers, it can be affected from both of applications and data plane elements. Thus, network operators need to

consider confidentiality and integrity of data-related network assets.

Second, in the case of software, resource-related assets, compromised applications could be used to affect availability of them to prevent controllers from operating normally. Thus, network operators should ensure availability of core service and resources (i.e., CPU, memory). Below, we describe in detail security issues for each network asset in terms of CIA triad [11].

3.1. Confidentiality Issue

Data-Related Assets. Most of current controllers provide varieties of northbound APIs ranging from accessing internal storage to enforcing switch commands. For example, ONOS [17] provides APIs such as `createOrUpdateHost`, `removeHost`, `removeDevice`, `changePortState`, and `FORWARD`. While these APIs help application developers to implement their own network applications, some APIs are fatal enough to affect the operation of entire network through breaking confidentiality of internal storage.

Suppose a malicious application invokes an API like removing host location or deleting a specific switch information at the internal storage of a controller. Basically, since multiple applications running on the controller share the internal storage, manipulating network information makes other benign applications to enforce wrong routing decisions due to erroneous network information. Therefore, a security mechanism restricting applications from accessing the internal storage or enforcing switch commands is required.

3.2. Integrity Issues

Host. A host manager included in contemporary controllers tracks host locations (e.g., connected switch, port) with IP and MAC addresses through incoming `PACKET_IN` messages and this information can be used by network applications to initiate communication paths between hosts.

However, the problem is that some controllers do not properly implement a message verification mechanism to check whether incoming `PACKET_IN` messages originate from benign hosts or not. According to Hong et al. [5], Floodlight [16] and old version of OpenDaylight [18] just support an empty-shell API that does not perform any functionality, called `isEntityAllowed`, rather than blocking `PACKET_IN` messages originating from spoofed host addresses. Therefore, malicious hosts can break integrity of host locations in the internal storage without agreement of the core system. To block forged messages from malicious hosts, controllers should support a security mechanism that verifies whether malicious hosts are involved in the host location update.

Link. Today's controllers use OpenFlow discovery protocol (OFDP) which leverages link layer discovery protocol (LLDP) to detect internal links between switches. However, as previous studies pointed out [5, 19], OFDP is vulnerable to link fabrication attack due to absence of LLDP message authentication.

Consequently, malicious hosts could participate in the link discovery process by monitoring LLDP messages from switches and inserting forged LLDP messages to fabricate nonexisting link at the data plane. To prevent malicious hosts from participating in link discovery process, controllers should be able to block LLDP messages from the port of switches connected to hosts.

Flow Rule. When switches cannot find any matching flow entries for incoming packets, they send `PACKET_IN` messages to a controller. Then, the controller forwards them to applications which listen to the `PACKET_IN` message, and the applications determine where to send packets.

The problem is that malicious applications may enforce a flow rule (i.e., candidate flow rule) that conflicts with already inserted flow rules. Porras et al. first introduce a following rule conflict example and present rule conflict detection engine [4]. Let us consider that a network security service (i.e., firewall) inserts a flow rule into a switch in the following equation:

$$\text{Host A} \longrightarrow \text{Host B}, \quad \text{drop packet.} \tag{1}$$

In this situation, a malicious application tries to insert a flow rule with multiple `SET` actions into the switch in the following equation:

$$\begin{aligned} &\text{Host A} \longrightarrow \text{Host C}, \quad \text{set A} \Longrightarrow \text{D}, \\ &\text{Host D} \longrightarrow \text{Host C}, \quad \text{set C} \Longrightarrow \text{B}, \\ &\text{Host D} \longrightarrow \text{Host B}, \quad \text{forward packet.} \end{aligned} \tag{2}$$

Basically, existing flow rule (1) does not allow host A to send packets to host B. However, candidate flow rule (2) modifies a source address from host A to host C and a destination address from host C to host B and finally forwards packets from host D to host B. It means that host A can bypass existing flow rule and access to host B through candidate flow rule. To prevent rule conflicts between applications, controllers should support rule conflict detection and mediation mechanism.

3.3. Availability Issues

Core System. As an intermediary between application layer and data plane, core system of a controller must operate without unexpected termination. If not, entire network can be damaged since applications cannot insert flow rules into switches. Some previous studies have already shown possible attack scenarios against availability of core system and proposed defense mechanisms for multiple attack methods.

First, Shin et al. have shown that some initial controllers (i.e., NOX, Floodlight) run applications on their own processes to reduce performance overhead caused by interprocess communication (IPC) between a controller and applications [10]. If applications are benign and bugfree, this design choice has the advantage of providing high performance. However, this design poses a critical problem when core system operates with a misbehaving application

in the same process. If the buggy or misbehaving application invokes some critical functions (e.g., `exit`), it causes the core system to terminate and consequently makes it impossible for switches to decide where to send packets.

Second, since core system and applications operate as application programs of operating system, malicious applications can affect the core system via operating system. For example, they can invoke critical system calls like spawning new processes or accessing internal storage. To prevent malicious applications from affecting the core system, the core system should isolate applications and monitor what they do to the core system.

Finally, core system receives OpenFlow messages from connected switches via control channel. However, if they communicate with each other via TCP sessions, malicious users can intercept the messages and modify them with malformed messages which violate OpenFlow protocol. As a previous study has proved [20], core system may perform unexpected behaviors (e.g., process termination) when it receives malformed messages. Therefore, there is a need for a security mechanism to check whether incoming messages follow OpenFlow protocol specification [21].

CPU and Memory. Since a controller and applications run on the same machine, they have to share resources (e.g., CPU, memory, and file descriptor). It means that there is a way for malicious applications to affect availability of core system and other coresident applications. Shin et al. have already shown that some controllers (e.g., POX [22], Beacon [23]) are halted due to memory shortage caused by malicious applications which allocate memory or generate new thread iteratively [10]. To mitigate this effect, a controller should limit the maximum boundary of resource that applications can use.

4. System Design

In this section, we present the design of NOSArmor that addresses the security issues described in Section 3.

4.1. Design Consideration. Each network asset requires different security principles to be guaranteed for different attack vectors: malicious application, malicious host, and malicious user. Therefore, rather than designing an integrated security mechanism to address all security issues mentioned in Section 3, we design and implement eight distinct security mechanisms, which we have named *security building block* (SBB), that comply with following guiding principles.

(i) Applications should be isolated from core system of controller.

(ii) Access control for APIs provided by controller is required to restrict access to internal storage from applications.

(iii) Flow rules in switches should be tracked and rule conflict should be addressed between a candidate flow rule and active flow rules.

(iv) Access to operating system should be controlled.

(v) Resources for applications need to be monitored and managed.

(vi) OpenFlow control messages received from switches should be monitored and controlled.

Table 2: Eight SBBs for addressing security concerns existing in SDN network.

Security building block	Network asset	C	I	A
Role-based authorization	Data-related assets	√	-	-
Host location tracker	Host	-	√	-
Link verifier	Link	-	√	-
Rule conflict mediator	Flow rule	-	√	-
Process separation	Core system	-	-	√
OpenFlow protocol verifier		-	-	√
System call checker		-	-	√
Resource manager	CPU and memory	-	-	√

4.2. System Architecture. As illustrated in Figure 2, NOSArmor consists of core components (i.e., packet I/O engine, protocol parser, core service, and application handler) needed to perform minimal functionality as a controller and eight SBBs that cover security principles of network assets.

Each SBB is placed at different locations depending on what network assets they deal with. First, OpenFlow protocol verifier, link verifier, and host location tracker take control messages received from switches as input and check whether the messages conform to their security policies or not. Second, process separation, role-based authorization, and rule conflict mediator prevent network applications from compromising security principles of network assets. In the case of resource manager, system call checker, they monitor applications' behaviors regardless of control message processing to prevent malicious applications indirectly affecting core system of controllers via operating system.

In addition, each SBB inspects different types of control messages. For example, OpenFlow protocol verifier checks whether all types of messages follow its security policy or not, whereas link verifier and host location tracker inspect only specific types of messages (i.e., `PACKET_IN`, `PORT_STATUS`). In the case of rule conflict mediator, it checks `FLOW_MOD` messages originated from network applications to verify whether rule conflict exists or not.

Table 2 enumerates eight SBBs and related security principle of network asset they address. Below, we discuss in detail how each SBB works.

4.2.1. Role-Based Authorization. One of security issues related to confidentiality of data-related assets is absence of an access control mechanism for internal storage from applications, and several studies suggest an access control system for the control layer [8, 24–27]. To restrict APIs that applications can invoke, we define three roles (`admin`, `network`, and `management`) and grant each role to different available permissions as shown in Table 3, which can be extended by network operators with additional roles and fine-grained permissions. An application assigned `admin` role

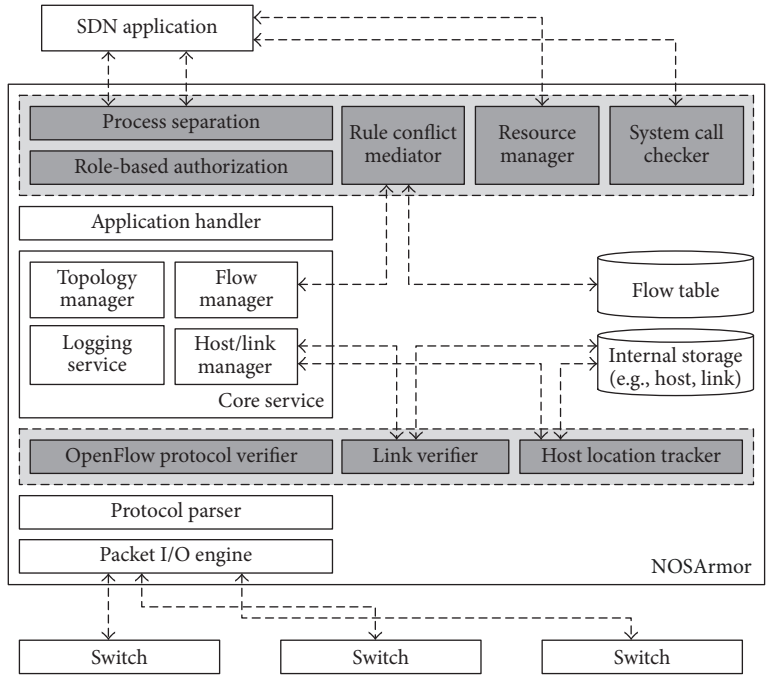

Figure 2: High-level overview of NOSArmor architecture.

Table 3: List of permissions granted to application roles.

Role	Permission
`Admin`	`write_flow, write_status, write_statistics`
	`read_flow, read_status, read_statistics`
`Network`	`write_flow`
	`read_flow, read_status, read_statistics`
`Management`	`read_flow, read_status, read_statistics`

can invoke any APIs from write operations (i.e., modifying network status stored in the internal storage to enforcing flow rules into switches) to read operations, whereas `management` role is only authorized to read operations.

4.2.2. Host Location Tracker. While controllers construct network topology by exchanging control messages with switches, they do not support a security mechanism to check whether incoming messages are caused by attacker's intervention or not. It provides a way for malicious hosts to break integrity of host information in the internal storage. To block attackers from participating in host location update, we place host location tracker between internal storage and host manager as shown in Figure 2. Our host location tracker follows an approach used by TopoGuard [5].

In the normal host location update, there are two conditions that must be satisfied. First, a switch's port which is connected to a host to be moved first should be down. Basically, hosts should disconnect from the connected switch to move to another location. Second, it should not be able to receive ICMP responses from host's original address. In NOSArmor, we only inspect switch's port status when a controller receives a `PACKET_IN` message that contains different location (i.e., switch, port number) from the internal storage about the same source IP address. If the port is alive, it considers that a malicious host sends a forged packet with spoofed IP address and does not update host location of benign host in the internal storage.

4.2.3. Link Verifier. To block malicious hosts from participating in link discovery process, controllers should be able to distinguish whether each switch port is connected to other switches or hosts and discard LLDP packets received from ports connected to hosts. To do this, we design link verifier based on an approach presented in TopoGuard [5] and place it between link manager and internal storage as shown in Figure 2.

Link verifier first defines extra properties (`ANY`, `HOST`, and `SWITCH`) for each switch port, and all ports of switches are initialized to `ANY` when switches are first connected to a controller. And then, when link verifier receives `PACKET_IN` messages related to LLDP packets, it changes the port type to `SWITCH`. In the case of receiving first hop traffic, port type is set to `HOST`. On the other hand, link verifier turns `SWITCH` and `HOST` port back `ANY` when it receives a `PORT_DOWN` signal included in `PORT_STATUS` messages. If link verifier receives LLDP packets from `HOST`, it does not process them and notify network operators what happened.

4.2.4. Rule Conflict Mediator. Enforcing a flow rule that is contrary to already inserted flow rules can break integrity of flow rule. To find a rule conflict and resolve it, we adopt an approach presented in FortNOX [4].

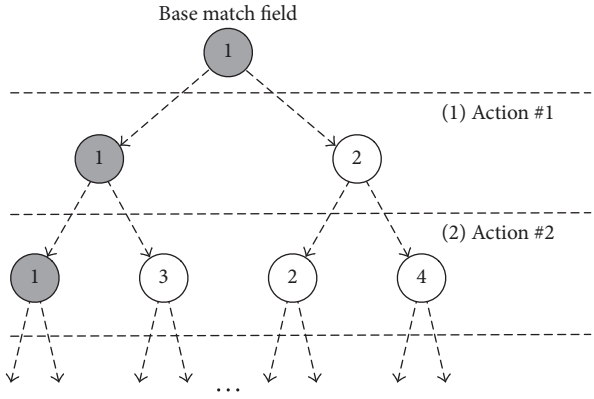

Figure 3: Computation of all possible match field sets by applying each action to a base match field.

Rule conflict mediator intercepts `FLOW_MOD` messages (we call them candidate rules) that applications try to enforce and checks whether rule conflict exists or not. If `FLOW_MOD` messages consist of multiple actions, rule conflict mediator computes all possible match field sets by applying each action to a base match field as illustrated in Figure 3. Then, it compares the sets with existing flow rules in switches. Basically, controllers do not know what flow rules switches currently have. To address this, we design a flow manager, which stores flow rules into the flow table in the internal storage when the controller installs flow rules into the switches and removes them according to the timeout of flow rules. When candidate rules conflict with an existing flow rule issued by an application having higher authorization, rule conflict mediator drops candidate rules and notifies network operators what application issued candidate rules.

4.2.5. Process Separation. While running a controller and applications at the same process provides high performance as illustrated in Figure 4(a), the availability of core system may be threatened by malicious applications. To enable controllers to provide continuous network service for data plane, we need to isolate applications from a controller and make them communicate using IPC.

As shown in Figure 4(b), each application will be executed as a new process which connects to the controller through IPC methods. In the core system, process separation is responsible for establishing a session and exchanging messages between a controller and applications. After the application is executed, (1) it first requests a connection to the controller with application id allocated by network operators. Process separation accepts the application request when the application id matches with a predefined application id. After session initiation process, (2) the controller forwards OpenFlow messages to which the application listens when it receives OpenFlow messages from switches. In this architecture, since a controller and applications operate as different processes, a malicious application cannot affect core system of controllers.

4.2.6. OpenFlow Protocol Verifier. The core system of a controller parses control messages and forwards the messages to appropriate core services according to message type. However, if control channel between a controller and switches is initiated using TCP, malicious users outside SDN network can intercept control messages and modify them with malformed messages which violate protocol specification. Then, the core system that receives malformed messages may perform unexpected behavior such as disconnecting from switches or causing the core system break down.

To prevent malformed messages from entering core service, OpenFlow protocol verifier checks whether incoming control messages follow protocol specification [21] at the entrance of controllers. If messages follow protocol specification, OpenFlow protocol verifier passes them to the core services of the controller. On the other hand, it drops them and notifies network operators of what malformed messages are coming.

OpenFlow protocol verifier checks the following two conditions: (i) *message length* and (ii) *field range* against incoming messages. First, for all OpenFlow messages, the length of incoming messages must be the same as the value of `length` field included in the header. Second, OpenFlow messages have various fields depending on the message type and each field should be set within a certain range. For example, `capability` field consisting of 32 bits of a `OFPT_FEATURES_REPLY` message should be set from `0x00` to `0xff`.

4.2.7. System Call Checker. The availability of core system may be indirectly affected by system calls invoked by malicious applications. Figure 5 illustrates how system call checker operates to control applications accessing the operating system. System call checker (1) first hooks system calls when applications invoke predefined critical system calls (e.g., spawns processes, shell) and (2) decides whether or not to deliver them to the operating system. When malicious applications try to invoke one of predefined critical system calls, the system call checker forcibly blocks them and records which application invokes a system call for network administrators. For serious cases, NOSArmor places process separation by default to terminate only malicious applications when it finds repeated malicious behaviors of applications.

4.2.8. Resource Manager. Since a controller and applications are located at the same machine, the availability of resources (e.g., CPU, memory) may be threatened by malicious applications through excessive resource usage. To mitigate this effect, resource manager limits the amount of resources available to each application.

As illustrated in Figure 6, (1) resource manager first reads resource file of each application that describes an expected resource usage of the application. Network operators can adjust the amount of resources that an application can use when the application requires more resources than it needs. And then, (2) it monitors resource usage of the application at regular time intervals and notifies network operators when the application consumes more resource than a predefined soft limit. If the application consumes more resource than the hard limit, it is forced to stop. Therefore, controllers could maintain available resources from malicious applications.

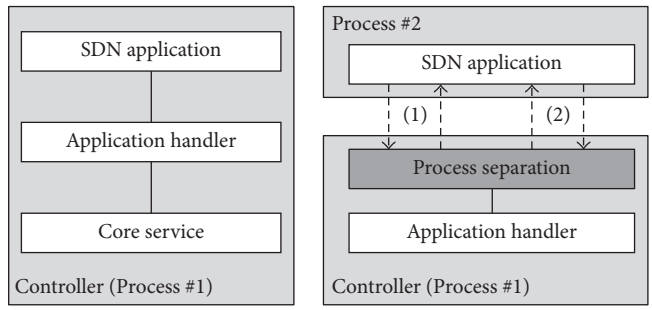

Figure 4: (a) A controller that operates as a single process with applications and (b) a controller that operates in different processes with applications.

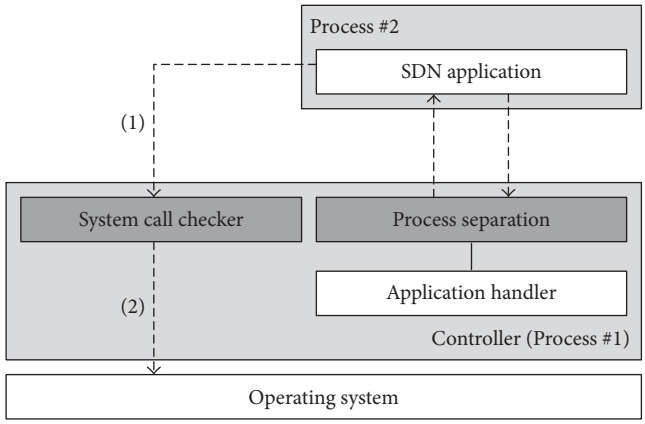

Figure 5: System call checker operating scenario.

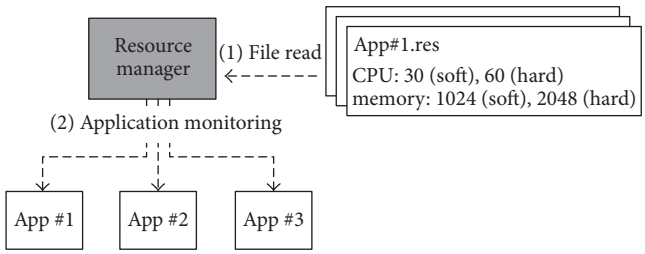

Figure 6: Resource manager operating scenario.

5. Evaluation

5.1. Implementation. We have implemented a prototype of NOSArmor as an extension of the Barista [28, 29] which is a base framework that provides a template which makes it easier to implement additional components. As additional components, we implement eight SBBs which are written in C and consist of 2,500 lines of code.

We have solved some implementation issues in developing SBBs. First, process separation employs Unix domain socket to allow applications running as standalone processes to communicate with a controller, but it causes considerable performance overhead due to IPC between applications and core service. To reduce this, each application initiates multiple connections with the controller and process separation employs epoll event model to improve performance of I/O multiplexing.

Second, rule conflict mediator should know what flow rules switches currently have to compare a newly candidate flow rule with active flow rules already inserted into the switches. However, while OpenFlow protocol is capable of asynchronously delivering messages to the controller when flow rules are removed from switches, it does not support a type of message to identify which flow rules switches have. To deal with this issue, we implement `flow manager` which stores flow rules when network applications insert flow rules into switches and removes flow rules when `HARD_TIMEOUT` has elapsed since flow rules were added. To synchronize current flow rules of switches between a controller and switches, `SOFT_TIMEOUT` and `HARD_TIMEOUT` of `FLOW_MOD` messages are set to be the same.

Third, we used SQLite [30] to store flow rules of switches, but it causes high overhead to read and write flow rules into the database. To reduce this overhead, we manage existing flow rules through hash tables and use five tuples (i.e., source IP address, source port number, destination IP address, destination port number, and protocol) as a key.

Finally, to prevent applications from affecting a controller via operating system, we implement system call checker as a shared object (`.so`) and link it with each application binary through `LD_PRELOAD` environment variable. Currently, system call checker hooks `POSIX_SPAWN`, `POSIX_SPAWNP` and network operators could easily extend functions to be monitored.

5.2. Evaluation Environment

Effectiveness Test Environment. To verify the effectiveness of SBBs in the NOSArmor, we perform black-box validation testing on each of them. In this test, we run three different types of attackers: (i) *malicious application* runs on a controller and takes malicious actions which violate security principles of network assets via APIs provided by the controller or system functions, (ii) *malicious host* operates on virtual network, Mininet [31], and takes malicious actions via script, and (iii) *malicious user* creates malformed

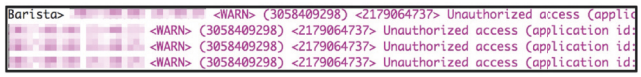

Figure 7: Application authorization test. Role-based authorization blocks API calls of an unauthorized application.

OpenFlow messages through modified Cbench [32], which is a performance testing tool for controllers and forwards them to controllers.

Performance Test Environment. We have called a controller that supports minimal functionality (i.e., forwarding) without any SBBs as a *base controller*. To measure and compare throughput of the base controller where different SBBs are deployed, we configured test environment with two machines, each for a controller and an OpenFlow message generator, Cbench [32], and connected them with 1 Gbps NIC. The configuration of machines is Intel E5-1650 CPU (12 cores, 3.60 GHz) and 64 GB of RAM.

5.3. Effectiveness Test

5.3.1. Role-Based Authorization. To verify the effectiveness of role-based authorization, we deploy a network application with least privilege (i.e., management) on a controller. As a management application, network statistics application is only authorized to read network status and cannot enforce flow rules into switches. As illustrated in Figure 7, when it invokes an API related to packet forwarding, role-based authorization blocks that API call and notifies network administrators of what happened. In addition, all messages are recorded and stored in log files for auditing.

5.3.2. Host Location Tracker. To show how host location tracker operates, we configure a virtual network as shown in Figure 8. In this topology, an attacker, host B, ultimately attempts to intercept host C's packets destined for host A (red dotted line in Figure 8).

As illustrated in the top of Figure 9, the benign host C sends ICMP requests to host A and receives ICMP replies before topology poisoning attack (black dotted line in Figure 8). To deceive host manager about the location of host A, the attacker first sends fake packets (i.e., with source IP and MAC address of host A) to host C. Since a switch B does not have a flow rule for host A coming from port 2, it sends a `PACKET_IN` message to determine where to send a packet. However, as illustrated in the bottom of Figure 10, even though the attacker tries to migrate the host A location in the internal storage, the host location tracker does not update the location of host A since the port of the switch connected to the host A is still alive. Therefore, the attacker cannot deceive the internal storage about the location of host A and he cannot communicate with benign host C.

5.3.3. Link Verifier. To prevent malicious hosts from injecting LLDP packets into link discovery process, link verifier checks whether incoming `PACKET_IN` messages come from a port of switch connected to other switches or not. To demonstrate how link verifier operates, let us consider a case where an attacker compromising benign hosts sends an LLDP packet to a connected switch.

Basically, since benign hosts should send and receive any packets to communicate with other hosts, host-connected ports should be set `HOST`. Therefore, as shown in Figure 11, link verifier can find LLDP injection through identifying a port type of the switch that sent an LLDP packet and records violations with warning messages.

5.3.4. Rule Conflict Mediator. To verify the effectiveness of rule conflict mediator, we explicitly insert a flow rule that blocks any packets from host A to host B on a switch through `flow pusher` application with `admin` role, which we have named *security rule*. In this situation, let us consider that a malicious application with `management` role tries to install a candidate flow rule (3), consisting of multiple `SET` actions, into the same switch where the security rule is installed.

$$\begin{aligned} &\text{Host A} \longrightarrow \text{Host C}, \quad \text{set A} \Longrightarrow \text{D}, \\ &\text{Host D} \longrightarrow \text{Host C}, \quad \text{set C} \Longrightarrow \text{B}, \\ &\text{Host D} \longrightarrow \text{Host B}, \quad \text{forward packet}. \end{aligned} \tag{3}$$

To check whether the candidate flow rule violates the security rule or not, rule conflict mediator first figures out the following possible match field sets (4) after applying each `SET` action included in the flow rule to a base match field.

$$\text{Host (A, D)} \longrightarrow \text{Host (B, D)} \quad \text{forward packet}. \tag{4}$$

Since one of the sets has the same `MATCH` field (i.e., host A → host B), but different forwarding policy (i.e., `FORWARD` and `DROP`), rule conflict mediator blocks the candidate flow rule which has lower priority.

5.3.5. Process Separation. To counter threats that malicious applications may interfere with core system of controllers, NOSArmor isolates applications from the core system. To evaluate the functionality of process separation, we deploy a malicious application which invokes an `exit` function to terminate the core system after a period of time the application starts running. As illustrated in Figure 12, since the application runs as a standalone process, it cannot have an impact on the operation of the core system.

5.3.6. OpenFlow Protocol Verifier. We modified Cbench [32] to fabricate malformed OpenFlow messages that do not follow OpenFlow 1.0 specification and forward them to a controller. In this situation, modified Cbench responds to a `FEATURES_REQUEST` message of a controller with a `FEATURES_REPLY` message which has an invalid `capability` field as shown in Figure 13. Basically, `capability` field uses only last 8 bits out of total 32 bits. OpenFlow protocol verifier drops OpenFlow messages which do not follow the specification and records into log files about what happened as illustrated in Figure 14. Network operators can identify which control channels are being attacked by checking the log files.

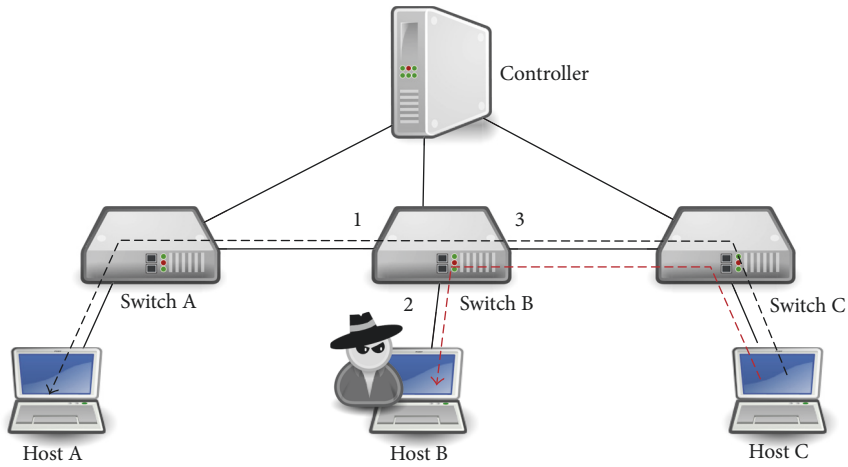

Figure 8: Test environment for evaluating the effectiveness of host location tracker.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=4.42 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=1.33 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.267 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.036 ms
```

Figure 9: Ping result from host A to C before topology poisoning attack.

```
mininet> h2 ./topo_poison.sh
*********************************************************
[MALICIOUS HOST] Spoofing IP, MAC address
*********************************************************
mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
```

Figure 10: Ping result from the attacker to host C after topology poisoning attack.

```
<ERROR> (0657791656) Violation: receive LLDP packets from HOST port [DPID:1, PORT:1]
<ERROR> (0657791656) Violation: receive LLDP packets from HOST port [DPID:1, PORT:1]
<ERROR> (0657791656) Violation: receive LLDP packets from HOST port [DPID:1, PORT:1]
```

Figure 11: The detection of link fabrication attack through link verifier.

```
Barista> [redacted] <INFO> (0657791656) Detected a new devi
[redacted] <INFO> (0657791656) Detected a new device (DPID:

*********************************************************************
[MALICIOUS APP] Malicious application try to terminate core process.
*********************************************************************

Barista>
```

Figure 12: A malicious application tries to terminate core system of a controller. The attack cannot affect the operation of core system due to application isolation.

```
0020  8f f8 37 a8 e4 a1 19 e9  26 51 4b f4 18 16 3a 42
0030  80 18 00 3a [redacted] 01 01 08 0a [redacted] e6 91
0040  66 87 b5 a2 01 ff ff ff  00 00 00 01 00 00 00 00
0050  00 00 00 01 ff ff ff ff  ff ff ff ff ff ff ff ff
0060  00 00 07 ff 00 01 1a c1  51 ff ef 8a [redacted]
```

(1) OFP_HEADER (2) DPID (3) CAPABILITY

Figure 13: OpenFlow messages not complying with OpenFlow protocol.

```
<ERROR> (0723076671)  2353538508 -> OFPT_FEATURES_REPLY | capabilities error
<ERROR> (0723076671)  2353538508 -> OFPT_FEATURES_REPLY | capabilities error
```

Figure 14: Malformed OpenFlow messages are blocked by OpenFlow protocol verifier with warning messages.

5.3.7. System Call Checker. Even though applications are isolated from a controller through process separation, they can indirectly affect controllers via an operating system. To mitigate this attack, NOSArmor deploys system call checker to restrict the applications from invoking privileged system calls. To verify the functionality of system call checker, we deploy a malicious application invoking a critical system call (i.e., `posix_spawn`). As illustrated in Figure 15, system call checker hooks and rejects `posix_spawn` when the malicious application invokes the system call to spawn a malicious process.

5.3.8. Resource Manager. We deploy a malicious application which keeps allocating memory on a controller to verify how resource manager handles this attack. First, when the controller starts its operation, it reads application configuration files (i.e., app.res) and enrolls available resource usage of applications into the operating system through system function (i.e., `rlimit`). In this situation, the malicious application is allowed to use 1.5 GB of memory.

After the application is executed, it allocates memory continuously. As shown at the bottom of Figure 16, kernel forcibly terminates the application that allocates more memory than is available.

```
Barista> Run command: ./malicious_app_2
*********************************************
Hooking posix_spawn function
This operation is not permitted.
*********************************************
posix_spawn: Unknown error -1

Barista>
```

Figure 15: The system call from a malicious application is blocked by system call checker.

```
Barista> [redacted] <INFO> (2907576801) Init - Resource management
rs_ext_app_cnt=1
External application configuration file: ../external_apps/ext_l2_forwarding/malicious_app.res

2016:12:07 10:15:16 <INFO> (2434229609) Init - Statistics management
2016:12:07 10:15:16 <INFO> (2434229609) Statistics request period: 1 sec
2016:12:07 10:15:16 <INFO> (4272823835) Init - Flow management
External application resource check complete

13575 [redacted]  20  0 0.290g 0.219g 0.001g R  6.0  5.7   0:00.06 malicious_app
13575 [redacted]  20  0 0.537g 0.467g 0.001g R 10.0 12.1   0:00.16 malicious_app
13575 [redacted]  20  0 1.033g 0.947g 0.001g R 10.0 24.5   0:00.26 malicious_app
13575 [redacted]  20  0 1.277g 1.206g 0.002g S  5.0 31.2   0:00.31 malicious_app
13575 [redacted]  20  0 0.000g 0.000g 0.000g Z  1.0  0.0   0:00.32 malicious_app
```

Figure 16: Resource manager restricts excessive memory consumption of applications.

5.4. Performance Test

5.4.1. Latency. While NOSArmor can improve the security of a base controller by deploying SBBs, it requires additional processing time for SBB's security logic. We measure the time it took each SBB to process a single control message. In the case of system call checker, resource manager, since they do not involve in control message processing, we do not take into account latency caused by them.

The latency for each SBB is illustrated in Table 4. We can see that most SBBs have similar latencies (2.61 μs on average). Among them, process separation shows the highest latency (35.3 μs) due to IPC communication between a controller and applications. In the case of rule conflict mediator, since it should inspect all of the flow rules of related switches, it has relatively high latency (6.96 μs).

5.4.2. Throughput. To compare the impact of SBBs on the controller performance, we measure the throughput of a set of SBBs along with base controller. Figure 17 illustrates the throughput for each SBB according to the number of switches. The throughput of base controller is highest regardless of the number of switches. Besides, as we increase the number of switches from 4 to 8, most of the SBBs obtain comparable throughput with the base controller except for process separation (from 400K to 1.4M responses/s on average). Process separation always shows the lowest throughput (from 320K to 880K responses/s) due to IPC communication. In the case of OpenFlow protocol verifier, it should check all types of messages; thus notable throughput degradation occurs when the number of switches is lower than 4.

Table 4: Latency for each SBB.

Security building block	Latency (μs)
Role-based authorization	2.33
Host location tracker	2.75
Link verifier	2.24
Rule conflict mediator	6.96
Process separation	35.3
OpenFlow protocol verifier	3.12

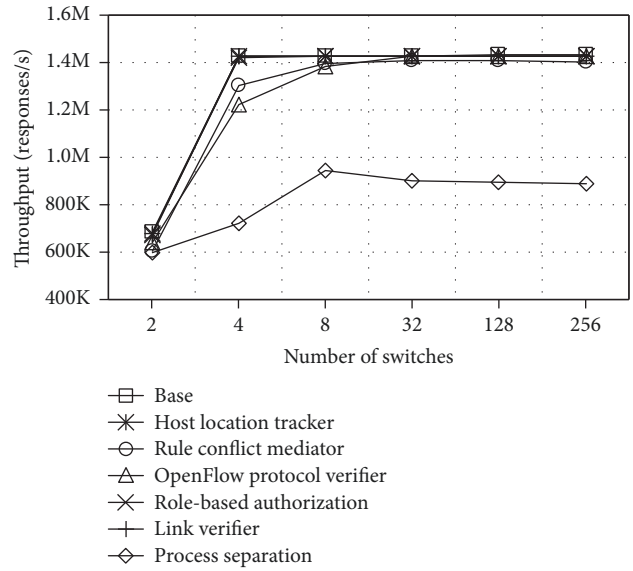

Figure 17: Throughput comparison between a base controller and SBBs.

Next, we measure the throughput of base controller when three SBBs that have the most impact on the performance of controller are deployed at the same time, process separation, OpenFlow protocol verifier, and rule conflict mediator. As illustrated in Figure 18, the throughput degrades every time adding each SBB and converges to that of the worst-performing SBB (i.e., process separation). Even though three SBBs are deployed at the same time, NOSArmor can guarantee 875,915.1 responses/s.

Figure 19 presents throughput comparison between NOSArmor and other controllers, ONOS and Floodlight. As one of the initial controllers, Floodlight shows the lowest throughput among controllers, which saturates at 40,918.4 responses/s, while ONOS shows the highest performance with 32 switches (i.e., 934145.1 responses/s). In the case of NOSArmor, even though it includes multiple security features, it shows better performance as compared to other controllers.

6. Limitation and Discussion

Even though our work addresses multiple security issues of different network assets existing at the control plane, there are some issues that still need to be addressed. First, while our work mainly focuses on security issues that may occur

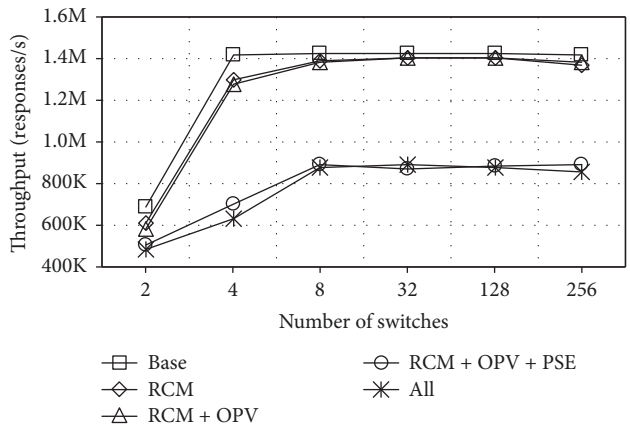

Figure 18: Throughput comparison according to combination of SBBs. *Note*. RCM: Rule Conflict Mediator, OPV: OpenFlow Protocol Verifier, and PSE: Process SEparation.

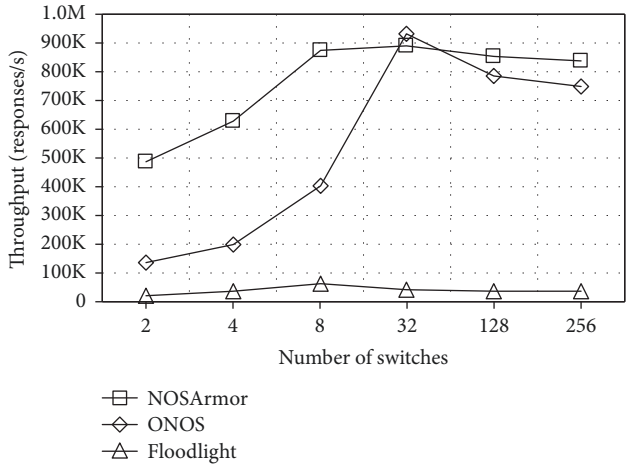

Figure 19: Throughput comparison between NOSArmor, ONOS, and Floodlight.

in a single controller instance, we do not consider distributed controller environment for a large-scale network that most widely used controllers (e.g., ONOS, OpenDaylight) support. Since each controller instance is responsible for maintaining part of the network in the distributed environment, they should communicate each other to know entire network topology via east/west interface. Therefore, this new interface could be a target of attackers.

Second, there is a significant difference in performance and functionality depending on how each SBB is implemented. For example, Rosemary also supports application isolation but shows higher performance than NOSArmor. With regard to functionality, SDNShield provides more fine-grained permissions than NOSArmor for access control. As a future work, we plan to investigate how to improve performance and functionality of each SBB.

Table 5 shows what security features NOSArmor and several previous works support. While Rosemary, SE-Floodlight, and SDNShield deal with security issues that could arise from malicious applications, TopoGuard focuses on security issues that originate from data plane elements. Therefore, each study cannot defend a controller against attack vectors presented in the other studies. In the case of NOSArmor, since NOSArmor integrates most possible security solutions to protect a NOS, it can provide a higher degree of security than previous works.

Table 5: Comparison of security features supported by NOSArmor, Rosemary [10], SE-Floodlight [9], SDNShield [8], and TopoGuard [5].

Security building block	NOS	R	SE	S	T
Role-based authorization	√	√	√	√	-
Host location tracker	√	-	-	-	√
Link verifier	√	-	-	-	√
Rule conflict mediator	√	-	√	-	-
Process separation	√	√	√	√	-
OpenFlow protocol verifier	√	-	-	-	-
System call checker	√	√	-	-	-
Resource manager	√	√	√	-	-

7. Related Work

There have been several early studies investigating security issues in SDN network [13–15]. Kreutz et al. explore potential threat vectors that may enable the exploit of SDN vulnerabilities [13]. Kloti et al. perform security analysis of OpenFlow protocol using STRIDE threat model [33] and attack tree modeling methods [14]. Benton et al. also present a brief overview of the vulnerabilities in OpenFlow protocol [15]. In addition, there are some works [34–38] that analyze vulnerabilities that exist in SDN network and present relevant studies dealing with vulnerabilities. Finally, Yoon et al. systemize the attack surfaces existing in SDN network and evaluate them against three popular SDN controllers [39]. Below, we introduce previous works that present concrete countermeasures against specific security issues (i.e., application layer, data plane) or detect unknown vulnerabilities.

Security Research for SDN Controller. Several previous studies propose security mechanisms against malicious applications [4, 8–10, 24, 40]. Yoon et al. [24] and Wen et al. [8, 40] implement permission-based access control system to prevent malicious applications from accessing the control plane. Shin et al. introduce robust and secure controller, Rosemary, which addresses diverse security issues concerning the security and robustness of controllers [10]. Porras et al. present a SE-Floodlight that addresses lots of security issues such as rule conflict, access control, and application isolation caused by malicious applications [4, 9].

The control plane can also be affected by hosts and switches in the data plane [5, 41, 42]. Hong et al. [5] and Deng et al. [41] suggest network topology poisoning attacks by injecting malicious packets from data plane elements, which make controllers misunderstand the state of data plane. Mattos et al. present authentication and access control mechanism for SDN networks [42]. Malicious hosts could also compromise availability of control plane through control

channel flooding [6, 7]. To mitigate this, Shin et al. apply SYN cookie concept to switches to verify whether host connection is benign or not [6]. Besides, Wang et al. also suggest efficient and protocol-independent defense framework to mitigate control channel saturation attack [7].

Vulnerability Detection. Even though many studies suggest possible attack cases and countermeasures, there may be attacks that have not yet been discovered. Lee et al. propose DELTA which is a penetration tool for evaluating SDN components and automatically finds new vulnerabilities in controllers [20]. Besides, Dhawan et al. introduce SPHINX to detect both known and potentially unknown attacks [43].

8. Conclusion

While SDN architecture brings many benefits to network operators, it exposes several high-value network assets to new threat vectors that do not exist in traditional network. Since controllers act as an intermediary between the application layer and the data plane, if controllers cannot ensure security principles of network assets existing at the control layer, entire network may not operate normally.

To address this, we first define what network asset needs to be protected and present the NOSArmor, which integrates eight security mechanisms named as SBB, to protect network assets from attackers. In addition, through our in-depth evaluation (i.e., effectiveness, performance), we demonstrate that NOSArmor shows competitive performance compared to existing other controllers with secureness of network assets.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grants funded by the Korean government (MSIT) (no. 2016-0-00078, Cloud Based Security Intelligence Technology Development for the Customized Security Service Provisioning).

References

[1] D. Foster, Google's networking lead talks sdn challenges for the next decade, https://www.linux.com/blog/event/open-networking-summit/2017/5/networking-challenges-next-decade.

[2] S. M. Kerner, Why facebook does sdn, http://www.enterprisenetworkingplanet.com/datacenter/why-facebook-does-sdn.html.

[3] J. Bort, Exclusive: Here's what happened when cisco lost a $1 billion deal with amazon, http://www.businessinsider.com/source-cisco-1b-amazon-deal-led-to-insieme-sdn-2013-10.

[4] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pp. 121–126, Association for Computing Machinery, Helsinki, Finland, August 2012.

[5] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: new attacks and countermeasures," in *Proceedings of the Network and Distributed System Security Symposium*, Internet Society, San Diego, Calif, USA, February 2015.

[6] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 413–424, Berlin, Germany, 2013.

[7] H. Wang, L. Xu, and G. Gu, "FloodGuard: a DoS attack prevention extension in software-defined networks," in *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 239–250, Rio de Janeiro, Brazil, 2015.

[8] X. Wen, B. Yang, Y. Chen et al., "SDNShield: Reconciliating configurable application permissions for SDN App markets," in *Proceedings of the 46th IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016*, pp. 121–132, France, July 2016.

[9] P. Porras, S. Cheung, M. Fong, K. Skinner, and V. Yegneswaran, "Securing the Software Defined Network Control Layer," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '15)*, San Diego, CA, USA, 2015.

[10] S. Shin, Y. Song, T. Lee et al., "Rosemary: a robust, secure, and high-performance network operating system," in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS '14)*, pp. 78–89, Scottsdale, Ariz, USA, November 2014.

[11] C. Perrin, The cia triad, Dostopno na: http://www.techrepublic.com/blog/security/the-cia-triad/488.

[12] ONF, Security requirement, https://www.opennetworking.org/sdn-resources/technical-library.

[13] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 55–60, Hong Kong, China, August 2013.

[14] R. Kloti, V. Kotronis, and P. Smith, "OpenFlow: A security analysis," in *Proceedings of the 2013 21st IEEE International Conference on Network Protocols (ICNP '13)*, Institute of Electrical and Electronics Engineers, Goettingen, Germany, October 2013.

[15] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 151-152, Hong Kong, August 2013.

[16] FloodLight, Open sdn controller, http://floodlight.openflowhub.org/.

[17] P. Berde, M. Gerola, J. Hart et al., "ONOS: Towards an open, distributed SDN OS," in *Proceedings of the 3rd ACM SIGCOMM 2014 Workshop on Hot Topics in Software Defined Networking (HotSDN '14)*, pp. 1–6, ACM, New York, NY, USA, August 2014.

[18] OpenDayLight, Open source network controller, http://www.opendaylight.org/.

[19] T. Alharbi, M. Portmann, and F. Pakzad, "The (in)security of Topology Discovery in Software Defined Networks," in *Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks, LCN 2015*, pp. 502–505, USA, October 2015.

[20] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. Porras, "DELTA: A Security Assessment Framework for Software-Defined Networks," in *Proceedings of the Network and Distributed System Security Symposium*, 2017.

[21] OpenFlow, Specification 1.0, https://www.opennetworking.org/.

[22] POX, Python network controller, http://www.noxrepo.org/pox/about-pox/.

[23] D. Erickson, "The beacon openflow controller," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 13–18, ACM, Hong Kong, August 2013.

[24] Open Networking Laboratory, Security-Mode ONOS, https://wiki.onosproject.org/display/ONOS/Security-Mode+ONOS.

[25] I. Alsmadi, "The integration of access control levels based on SDN," *International Journal of High Performance Computing and Networking*, vol. 9, no. 4, pp. 281–290, 2016.

[26] J. Noh, S. Lee, J. Park, S. Shin, and B. B. Kang, "Vulnerabilities of network OS and mitigation with state-based permission system," *Security and Communication Networks*, vol. 9, no. 13, pp. 1971–1982, 2016.

[27] M. Wang, J. W. Liu, J. Chen, and J. Mao, "Perm-guard: Authenticating the validity of flow rules in software defined networking," *Journal of Signal Processing Systems*, vol. 86, no. 2-3, pp. 157–173, 2017 (Chinese).

[28] J. Nam, H. Jo, Y. Kim, P. Porras, V. Yegneswaran, and S. Shin, "Bridging the architectural gap between NOS design principles in software-defined networks," in *Proceedings of the the 2017 Symposium*, pp. 637-637, Santa Clara, California, September 2017.

[29] J. Nam, H. Jo, Y. Kim, P. Porras, V. Yegneswaran, and S. Shin, "Barista: an event-centric nos composition framework for software-defined networks," in *Proceedings of the INFOCOM 2018-IEEE Conference on Computer Communications*, 2018.

[30] SQLite, Sql database engine, https://www.sqlite.org/.

[31] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets '10)*, 19:6, 19:1 pages, ACM, October 2010.

[32] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, On controller performance in software-defined networks., Hot-ICE 12 (2012) 1–6.

[33] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Threat modeling-uncover security design flaws using the stride approach," *MSDN Magazine-Louisville*, pp. 68–75, 2006.

[34] W. Li, W. Meng, and L. F. Kwok, "A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures," *Journal of Network and Computer Applications*, vol. 68, pp. 126–139, 2016.

[35] I. Alsmadi and D. Xu, "Security of software defined networks: a survey," *Computers & Security*, vol. 53, pp. 79–106, 2015.

[36] Z. Shu, J. Wan, D. Li, J. Lin, A. V. Vasilakos, and M. Imran, "Security in Software-Defined Networking: Threats and Countermeasures," *Mobile Networks and Applications*, vol. 21, no. 5, pp. 764–776, 2016.

[37] J. Spooner and S. Y. Zhu, "A review of solutions for sdn-exclusive security issues," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 7, no. 8, 2016.

[38] N. Dayal, P. Maity, S. Srivastava, and R. Khondoker, "Research Trends in Security and DDoS in SDN," *Security and Communication Networks*, vol. 9, no. 18, pp. 6386–6411, 2016.

[39] C. Yoon, S. Lee, H. Kang et al., "Flow wars: systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3514–3530, 2017.

[40] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for OpenFlow applications," in *Proceedings of the 2013 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013*, pp. 171-172, China, August 2013.

[41] S. Deng, X. Gao, Z. Lu, and X. Gao, "Packet Injection Attack and Its Defense in Software-Defined Networks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 695–705, 2018.

[42] D. M. F. Mattos and O. C. M. B. Duarte, "Authflow: authentication and access control mechanism for software defined networking," *Annals of Telecommunications*, vol. 71, no. 11-12, pp. 607–615, 2016.

[43] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting Security Attacks in Software-Defined Networks," in *Proceedings of the Network and Distributed System Security Symposium*, 2015.