A Formal Access Control Model for SE-Floodlight Controller

Abdullah Al-Alaj
Institute for Cyber Security,
Center for Security and Privacy
Enhanced Cloud Computing,
Department of Computer Science,
UTSA, San Antonio, Texas
abdullah.al-alaj@utsa.edu

Ravi Sandhu
Institute for Cyber Security,
Center for Security and Privacy
Enhanced Cloud Computing,
Department of Computer Science,
UTSA, San Antonio, Texas
ravi.sandhu@utsa.edu

Ram Krishnan
Institute for Cyber Security,
Center for Security and Privacy
Enhanced Cloud Computing,
Department of Electrical and
Computer Engineering,
UTSA, San Antonio, Texas
ram.krishnan@utsa.edu

ABSTRACT

Software defined networking (SDN) offers a promising approach for the next generation of networking technology. However, at present there is no widely accepted model for network applications authorization. One reason for lack of access control system is the absence of clear definition of an authorization model in SDN. Porras et al [9] recently developed SE-Floodlight for this purpose. They partly employed the notion of the well-known role-based access control (RBAC) model. They informally presented a role-based authorization system to manage applications access rights to network operations, submitted during the interaction between the application layer and the switch-side infrastructure. In this paper we develop a formal role-based authorization model in SDN using SE-Floodlight as a reference controller. Based on the formal model we discuss security aspects and propose some extensions. We also provide an administrative model for the authorization system. We show a configuration of the formal model for a use case scenario and discuss the security aspects of the authorization model and describe some security issues related to over-privileged apps, limitations of role hierarchy, app upgrading, and app downgrading problem. Finally, we propose a refined role hierarchy to address these problems.

CCS CONCEPTS

• **Security and privacy** → **Access control**; **Authorization**; *Formal security models*; • **Networks** → Network security;

KEYWORDS

Security; Software Defined Networks; Access Control; Role Based Access Control

ACM Reference Format:
Abdullah Al-Alaj, Ravi Sandhu, and Ram Krishnan. 2019. A Formal Access Control Model for SE-Floodlight Controller. In *ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (SDN-NFVSec '19), March 27, 2019, Richardson, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3309194.3309195

SDN-NFVSec '19, March 27, 2019, Richardson, TX, USA

ACM ISBN 978-1-4503-6179-8/19/03…$15.00
https://doi.org/10.1145/3309194.3309195

1 INTRODUCTION

Currently, SDN controllers [18–21] act as the interaction point between OpenFlow applications (apps) and the network infrastructure (data plane). Verifying apps' operations sent to the data plane is an absolute prerequisite for maintaining a consistent network security policy which is the responsibility of the controller. Thus, the absence of methods for controlling app's access rights on the infrastructure allows buggy or malicious apps to run arbitrary commands which makes SDN network vulnerable to various attacks from apps. Therefore, SDN OSs require an effective authorization mechanisms to ensure secure network operations [2, 10, 12].

The SDN controller is required to maintain the integrity of the network state and so the access control system has to ensure that the tasks carried out by apps preserve such integrity. Generally, network apps perform tasks including traffic engineering, monitoring, security services, etc., via executing network operations to program accessible OpenFlow-enabled switches within their controller domain. Network apps dynamically access network resources via generating network operations like flow rule insertion, network state inquiry, port configuration, etc. These operations are received by the controller which submits them to the switch. The switch, as a forwarding device, has no means to verify app operations and so absolutely trusts them and operates accordingly.

Prior studies have addressed access rights of SDN apps to protect against insecure access to data plane resources [2, 3, 5, 9, 14, 22]. SE-Floodlight [9, 17] is a security extension for Floodlight in which Porras et al *informally* proposed Role-based access control (RBAC) to enforce the security policy on the data exchange operations between network apps and the forwarding switches in addition to flow rule conflict resolution strategy driven by roles.

Rigorous conceptual modeling in the context of access control is important because it facilitates early detection and correction of system vulnerabilities and shortcomings before actual deployment. Since the SE-Floodlight authorization system is informally presented, in this paper we address this issue and formalize the system, show the key elements of the access control model and discuss the security aspects to facilitate searching for modifications and alternatives. Building a formal access control model helps in supporting the communication between researchers, developers and users for better understanding of the access control domain and for initiating further study.

Table 1: Types of data exchange operations along with the minimum authorization role [9].

Type ID	Type of Data Exchange Operation	Minimum Auhorization Role	Open Flow Message Type
t1	Flow removal messages	APP	OFPT_FLOW_REMOVED
t2	Flow error reply	APP	OFPT_ERROR
t3	Echo requests	APP	OFPT_ECHO_REQUEST
t4	Echo replies	APP	OFPT_ECHO_REPLY
t5	Barrier requests	APP	OFPT_BARRIER_REQUEST
t6	Barrier replies	APP	OFPT_BARRIER_REPLY
t7	Switch get config	APP	OFPT_GET_CONFIG_REQUEST
t8	Switch config reply	APP	OFPT_GET_CONFIG_REPLY
t9	Switch stats request	APP	OFPT_STATS_REQUEST
t10	Switch stats report	APP	OFPT_STATS_REPLY
t11	Packet-In return	APP	OFPT_PACKET_IN
t12	Flow rule mod	APP	OFPT_FLOW_MOD
t13	Packet-Out	SEC	OFPT_PACKET_OUT
t14	Vendor actions	ADMIN	OFPT_VENDOR
t15	Vendor features	ADMIN	OFPT_FEATURES
t16	Switch port status	ADMIN	OFPT_PORT_STATUS
t17	Switch port mod	ADMIN	OFPT_PORT_MOD
t18	Switch set config	ADMIN	OFPT_SET_CONFIG

Although SE-Floodlight authorization system doesn't adopt RBAC sessions, it adopts the concept of role hierarchy, inheritance relation between roles which, compared to other authorization systems [3, 5, 15], make it the reference for our work in this paper for formalizing and analyzing apps authorization in SDN. Because the proposed formal model relies on data exchange operations between apps and the data plane that are driven by OpenFlow protocol, it can be adopted by other controllers.

The paper is organized as the following. Section 2 presents a background about SE-Floodlight authorization system. The formal access control model of SE-Floodlight is presented in Section 3. In section 4 we propose a formal administration model for SE-Floodlight. Section 5 shows the model configuration for an example use case. Possible extensions to the authorization model are presented in Section 6. Related work is presented in Section 7. Section 8 concludes the paper and highlights the future work.

2 APP AUTHORIZATION IN SE-FLOODLIGHT

The SE-Floodlight authorization system aims to control access to all OpenFlow messages exchanged between apps and the data plane. Each data exchange operation is abstracted using an OpenFlow operation type. For example, operations like insertion, deletion, and update of a flow rule are all of type 'Flow rule mod'. In SE-Floodlight, types of data exchange operations are assigned to roles and then roles are assigned to apps.

The authorization system identifies three authorization roles, namely, ADMIN, SEC, and APP with a total order role hierarchy. For instance, apps in SEC role indirectly have permission to receive packet-in notifications which is originally assigned to the lower role APP. This is analogous to role inheritance relation in RBAC model [16, 23].

Table 1 identifies the types of data exchange operation types that alternate between apps and OpenFlow switches along with the minimum role that must be assigned to an app to perform an operation of that type. The table also shows, for each operation type, the corresponding OpenFlow message type required to carry the operation.

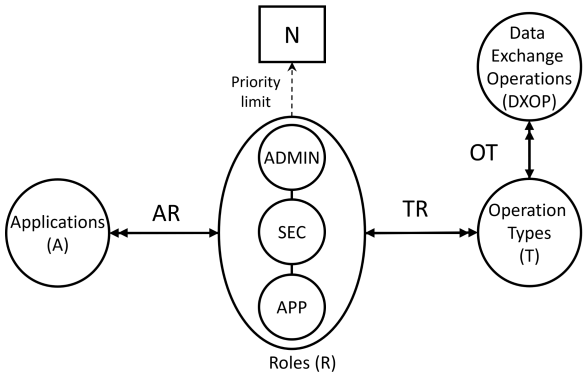

Figure 1: SE-Floodlight conceptual authorization model [1].

With app co-existence, an app may insert a flow rule that causes a conflict with another pre-existing flow rule in the switch. For resolving conflicts, SE-Floodlight associates each role with a priority limit that represents the maximum priority value that can be assigned to flow rules produced by apps of this role. Flow rules produced by an app of a specific role have higher precedence than rules produced by apps in a lower role. An app also uses this value to prioritize its own set of flow rules within the sub-range of priorities corresponding to its role.

3 FORMALIZED SE-FLOODLIGHT ACCESS CONTROL MODEL

In this section we formalize the authorization model of SE-Floodlight [9] and describe its components.

3.1 Overview

The basic components of SE-Floodlight authorization model include: Apps (A), Roles (R), Data Exchange Operations (DXOP), and types of DXOPs. We also discuss the credential entity which is implicitly included in the model. The model and the relations between the components are shown in Figure 1, and discussed below.

Apps (A): The local and remote OpenFlow apps.

Roles (R): Roles in SE-Floodlight are used for two main purposes: app permission authorization and rule-based conflict resolution.

Data Exchange Operations (DXOP) and Operation types (T): These operations represent OpenFlow messages exchanged between the dataplane and the apps. These operations belong to different operation types. For example, flow table modification messages add, modify, modify_strict, delete, and delete_strict are grouped under the type 'Flow rule mod', represented by OFPT_FLOW_MOD message type. Also, operations that query the system for statistical information about flows, ports, tables, queues, etc, are all of type 'Switch stats request', represented by OFPT_STATS_REQUEST message type. Table 1 summarizes all these data exchange operations

[1] Arrows denote binary relations with the single arrowhead indicating the one side and double arrowheads the many side.

and their corresponding messages within an OpenFlow v1.0 stack.
Credentials: Credentials are used for both authorization and authentication. Each app is uniquely identified by an administrator-assigned credential which also contains the authorization role assigned to the app. The credential is added to each message sent by the app. When a message submitted, the role embedded within the credentials is extracted to identify the app and its role.

3.2 Formal SE-Floodlight Access Control Model

For this model, we assume the SDN infrastructure has multiple switches controlled by a single controller within the same slice. For simplicity and easier reference, we created a basic formal model without flow rule conflict resolution in Table 2 and an extended model including the conflict resolution in Table 3.

As shown from the definition in Table 2, an app can be assigned to only one role denoted by *AR* relation. *TR* relation denotes that an operation type can be assigned to one role. The type of each data exchange operation can be specified using *type* function. The *Authorization Rule* is stated based on all the previously defined relations and functions considering the effect of role inheritance.

The authorization model is extended in Table 2 in which the rule conflict algorithm contributes to authorizing 'add flow rule' operations. The function *priorityLimit* assigns a natural number to each role. This number represents the maximum priority an app in this role can assign to new flow rules submitted with 'add flow rule' operation. It is used to resolve conflicts between different flow rules. $Authorization\_rule_{op='add\ alow\ rule'}$ checks, using the *RCA* function, whether an app has the right to insert a new flow rule . Finally, $Authorization\_rule_{op \in DXOP-'add\ flow\ rule'}$ checks whether an app has the right to perform all operations other than 'add flow rule' operations that are not mediated by the *RCA* function.

It should be noted that if $Authorization\_rule_{op='add\ alow\ aule'}$ returns true, which means a successful addition of new flow rule, then the access control model updates the set *FT* of the target switch by adding the new flow rule and removing the conflicting rule, if any. This happens only if the result of rule conflict algorithm *RCA* returns 'Add' or 'Exchange'. Also, the model registers the flow rule priorities so that they can be used in future authorization decisions.

4 ADMINISTRATIVE MODEL

Next we discuss the administrative model that is used for the creation and maintenance of the system's basic element sets, functions, and relations. It is assumed that all administrative functions are performed by a network operator user with enough privileges. Table 4 formally specifies the administration functions to manage the apps, roles, operations, and operation types. The second column shows the condition required to perform the function and the third column shows the corresponding updates to the authorization system.

As shown in Table 4, administration functions for managing registration and de-registration of apps are *addApp* and *deleteApp*, respectively. Roles are created and removed from the system using *addRole* and *deleteRole* functions. Types are created and removed from the system using *addType* and *deleteType* functions. When a role is deleted, all assignment relations between the deleted role and any app or operation type must be found and deleted from the system as shown in the third column.

Functions *assignApp* and *revokeApp* are used to create and delete a relation between apps and roles. Operation types are assigned to roles using *assignType* and revoked using *revokeType* function. Operations are assigned to their types using *assignOp* function and revoked using *revokeOp* function.

5 USE CASE SCENARIO

Applying this formal model by network operators depends on the use case under implementation. In this section we configure the formal model for a use case scenario and show the relevant authorization aspects.

In this use case we have five OpenFlow apps, namely Learning Switch (*LS*), Load Balancer (*LB*), Network Intrusion Prevention (*NIP*), Firewall (*FW*), and Operator Console (*OC*) app.

LS app requires insertion of flows in switches that routes packets of all devices in the topology after they have been learned using Packet-In messages, so it needs permission to operation types 'Flow rule mod' and 'Packet-In return'.

The *LB* app requires permission to collect flow statistics from the switches and to distribute traffic among deferent servers/links accordingly. It requires permission to operation of type 'Flow rule mod', 'Switch stats request' , and 'Switch stats report' so it is enough to be assigned the APP role.

The *NIP* app detects and blocks intrusion attempts on the network. In order to do so, it requires permission to receive packet-in notifications for performing packet inspection to drop malicious traffic and it needs permission to insert flow rules for forwarding sanitized traffic to the correct destination [24]. The Operation types required by this app are 'Flow rule mod' and 'Packet-In return'. Despite that it is enough to assign it to the APP role, this app is intended to enforce security policy and so should be assigned to the SEC Role in order to replace conflicting flow rules inserted by apps in APP role.

The *FW* app [24] that performs basic firewall tasks such as enforcing Access Control List (ACL) on OpenFlow switches. For each incoming Packet-In message, the firewall compares the header fields against each rule in the sorted list sequentially from the highest priority. The app matches every single packet against the firewall rules. Hence, the Forwarding app uses Packet-Out messages to forward each packet. It forwards a packet through sending a Packet-Out message with an appropriate action for an ALLOW decision, while it drops a packet through sending a Packet-Out message without specifying an Output action for a DROP decision. So this app needs permission to 'Packet-In return' and 'Packet-Out' operation types.

Finally, the (*OC*) app is capable of performing all operation types and to configure and read network state, so its assigned the ADMIN role. The configuration of the formal access control model for this use case scenario is given in Table 5.

6 DISCUSSION AND POSSIBLE EXTENSIONS

In this section we discuss some of the issues that may violate access control principles and call for modifications in the model.

-Over-privileged apps: SDN apps should be confined to the principle of least privilege. However, following the fixed set *R* and the *TR* relation as shown in Table 1 may grant an app the permission to one or more operations. All apps in the lowest role *APP* will have

Table 2: SE-Floodlight Authorization Model Definitions without Flow Rule Conflict Resolution.

- Basic Sets and Functions:
A: a finite set of OpenFlow apps.
T: a finite set of types of data exchange operations.
$R = \{ADMIN, SEC, APP\}$: a fixed set of three roles.
>: a total order on *R* where *ADMIN* > SEC and *SEC* > *APP*.
$AR \subseteq A \times R$, a many-to-one relation, i.e., $(a,r_1) \in AR \wedge (a,r_2) \in AR \Rightarrow r_1 = r_2$, mapping each app to one role.
$TR \subseteq T \times R$, a many-to-one relation, i.e., $(t,r_1) \in TR \wedge (t,r_2) \in TR \Rightarrow r_1 = r_2$, mapping each operation type to one role.
DXOP: a set of possible data exchange operations where each operation $op \in DXOP$ contains a flow rule and a priority if $o = $ 'add flow rule'.
type: $DXOP \longrightarrow T$, a function specifying the type of each operation. Equivalently viewed as a many-to-one relation $OT \subseteq DXOP \times T$, where $(o,t_1) \in OT \wedge (o,t_2) \in OT \Rightarrow t_1 = t_2$.
- Authorization Rule:
Authorization_rule: $A \times DXOP \longrightarrow \{T, F\}$, checks whether $a \in A$ has the right to perform an operation $o \in DXOP$.
Authorization_rule $(a : A, o : DXOP) \equiv (\exists r_1, r_2 \in R \cdot (a, r_1) \in AR \wedge (type(o), r_2) \in TR \wedge r_1 \geq r_2)$.

Table 3: SE-Floodlight Authorization Model Definitions with Flow Rule Conflict Resolution.

- Basic Sets and Functions:
All basic sets and functions from Table 2.
FR: a set of all possible flow rules where for each $fr_i \in FR$ there should be a priority.
priority_limit: $R \longrightarrow \mathbb{N}$, the mapping of role to the highest priority an app in $r \in R$ may assign to its flow rules, where *priority_limit*(*ADMIN*) > *priority_limit*(*SEC*) > *priority_limit*(*APP*).
S: Set of switches in the network slice.
FT: $S \longrightarrow 2^{FR}$, the set of flow rules currently in a switch's flow table.
rule: $DXOP \longrightarrow FR$, a function that returns the flow rule $fr_c \in FR$ of an operation $op \in DXOP$ given that $type(op) =$ 'Flow Rule Mod'.
priority: $FR \longrightarrow \mathbb{N}$, the mapping of a flow rule $fr_c \in FR$ to its priority.
RCA(fr_c: *FR*, pr_c:$\mathbb{N}$, s_t:S) $\longrightarrow$ {*Reject, Add, Exchange*}, a function uses rule-based conflict analysis described in [9] that returns the result of a request to add of new flow rule fr_c into $FT(s_t)$ submitted with priority pr_c. *'Reject'*, *'Add'*, *or 'Exchange'* indicates whether fr_c is rejected, added without removing pre-existing rules, or exchanged with a conflicting flow rule $fr_i \in FT(s_t)$, respectively.
- Authorization Rules:
$Authorization\_rule_{op='add\ flow\ rule'} : A \times S \longrightarrow \{T, F\}$, checks whether $a \in A$ has the right to insert a flow rule $rule(op)$ into $FT(s_t \in S)$.
$Authorization\_rule_{op='add\ flow\ rule'}\ (a : A, s_t{:}S) \equiv (\exists r_1, r_2 \in R \cdot (a, r_1) \in AR \wedge (type(op), r_2) \in TR \wedge r_1 \geq r_2) \wedge$
$(RCA(rule(op), priority(rule(op)), s_t) \in \{Add, Exchange\})$.
$Authorization\_rule_{op \in DXOP - 'add\ flow\ rule'} : A \times S \longrightarrow \{T, F\}$, checks whether $a \in A$ has the right to perform a non-flow-rule-insertion operation.
$Authorization\_rule_{op \in DXOP - 'add\ flow\ rule'}\ (a : A, s_t{:}S) \equiv (\exists r_1, r_2 \in R \cdot (a, r_1) \in AR \wedge (type(op), r_2) \in TR \wedge r_1 \geq r_2)$

the permission to add flow rules whereas some apps don't require this permission. For example, a billing app requires only to read statistics of the port connected to a host's device (NIC). It computes a monthly bill for a customer based on the sent and received bytes. APP role grants this app the permission to insert flow rules which violates the least privilege principle and this could be maliciously exploited to attack the controller or other apps. The network state can be modified and then the controller and other apps might take decisions based on this inconsistent state.

-App upgrading problem: To satisfy network security needs and respond to security threats, flow rules inserted by security apps should have higher priority than those of traffic engineering apps. Therefore, based on the SE-Floodlight access control model, such apps will be upgraded from APP to SEC role only to satisfy the priority requirement. As a result, they will be granted permission to 'packet-Out' operation type. This is a violation of the least privilege principle of access control. The NIP app, discussed in Section 5, is an example of this case.

-Limitations of role hierarchy: Some apps might need to perform different set of operations and the network operator wants to assign them the same priority limit. This is impossible in SE-Floodlight authorization system due to the total order relation between roles. Furthermore, assigning roles to apps based on the tasks they achieve is limited with the existence of only three role levels and the way operation types are assigned to roles in Table 1.

-App downgrading problem: When a role *r* is assigned to an app *a*, it provides the permission to possibly multiple operation types. If the network operator later wants to downgrade *a*, by revoking only one operation type *t* from *a* for example. Revocation of *t* cannot be directly applied to *a*, it should be done through *revokeType*(*t*,*r*) function. However, this kind of revocation doesn't work because *r* is most likely shared by multiple apps and this will downgrade all apps in *r*. This can be done only by creating a new role r' that

Table 4: Administrative Model for SE-Floodlight.

Function	Condition	Update
addApp(*a*)	$a \notin A$	$A' = A \cup \{a\}$
deleteApp(*a*)	$a \in A \wedge (a,r) \in AR$	$AR' = AR \setminus \{(a,r)\}$, $A' = A \setminus \{a\}$
addType(*t*)	$t \notin T$	$T' = T \cup \{t\}$
deleteType(*t*)	$t \in T \wedge (o,t) \in OT \wedge (t,r) \in TR$	$OT' = OT \setminus \{\forall (o,t) \in OT\}$, $TR' = TR \setminus \{(t,r)\}, T' = T \setminus \{t\}$
addRole(*r*)	$r \notin R$	$R' = R \cup \{r\}$
deleteRole(*r*)	$r \in R \wedge (a,r) \in AR \wedge (t,r) \in TR$	$AR' = AR \setminus \{\forall (a,r) \in AR\}$, $TR' = TR \setminus \{\forall (t,r) \in TR\}$, $R' = R \setminus \{r\}$
assignApp(*a*,*r*)	$a \in A \wedge r \in R \wedge (a,r) \notin AR$	$AR' = AR \cup \{(r,a)\}$
revokeApp(*a*,*r*)	$a \in A \wedge r \in R \wedge (a,r) \in AR$	$AR' = AR \setminus \{(a,r)\}$
assignType(*t*,*r*)	$t \in T \wedge r \in R \wedge (t,r) \notin TR$	$TR' = TR \cup \{(t,r)\}$
revokeType(*t*,*r*)	$t \in T \wedge r \in R \wedge (t,r) \in TR$	$TR' = TR \setminus \{(t,r)\}$
assignOp(*o*,*t*)	$o \in DXOP \wedge t \in T \wedge (o,t) \notin OT$	$OT' = OT \cup \{(o,t)\}$
revokeOp(*o*,*t*)	$o \in DXOP \wedge t \in T \wedge (o,t) \in OT$	$OT' = OT \setminus \{(o,t)\}$

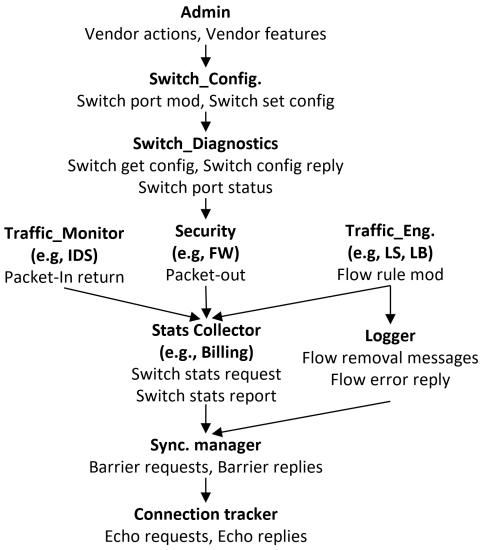

Figure 2: Proposed Role Hierarchy.

has all operation types in r except t then applying $assignApp(a, r')$. This scenario calls for a more flexible role hierarchy.

For addressing the above problems, we refine the total order hierarchy and propose a partial order role hierarchy as shown in Figure 2. We modify the *AR* relation to be many-to-many relation to allow for assigning multiple roles to one app. Types of operations are assigned to roles based on the task they achieve taking into consideration inherited permission types. In this hierarchy we consider only app interactions with the forwarding infrastructure and omit app requests to read and write controller data stores that maintain information about end hosts, network topology, etc.

Also, roles are organized based on the sensitivity of operation type set managed by this hierarchy. An operation type that may alter the network state should be part of higher roles whereas lower roles should encapsulate non-harmful operation types. The role name indicates the general function achieved by operation types in the role and its inherited permissions. For resolving rule conflicts, higher roles should have higher priority limit as they have more power in the authorization system. Incomparable roles should be assigned same priority limit or based on the network operator's configuration.

App access rights can be managed by manipulating *AR* and *TR* relations or by creating/deleting a role inheritance relation. Each role in this partial order hierarchy encapsulates less permission types compared to the roles of the total hierarchy of SE-Floodlight and the application-level roles of [5]. We don't propose direct assignment of permission types to apps since we believe it would be a management burden for network operators.

This role hierarchy avoids the limitations of the total role hierarchy with a more flexible and finer grained operation type-to-role assignment. Based on this hierarchy, apps can be assigned appropriate roles and thus network operators can avoid *over-privileged apps*. Also, as a result of this flexible app-role assignment, *app upgrading problem* is fixed because priorities for incomparable roles can be configured by network operator. In this case app upgrading will not be conducted only for priority limit reasons.

App downgrading problem is solved by this hierarchy since each role has a small number of strongly related operation types that can be granted as all-or-none basis. As a result, for an app a that is assigned role r, revoking type t from a can be done by invoking $revokeApp(a,r)$ followed by $assignApp(a, r')$ where r' is the immediate ascendant or r.

7 RELATED WORK

A number of security issues have been identified concerning SDN apps [1, 6, 11, 12] with specific issues related to app authorization.

Several approaches have been proposed to protect SDN resources from unauthorized access by network apps. We classify SDN apps authorization into two main categories: Firstly, Permission-based app authorization which includes techniques wherein apps authorization is driven by direct permission-app assignment. Secondly, role-based app authorization in which app authorization is driven by permission-to-role followed by role-to-app assignment.

PermOF [13] proposed a permission system in which a permission set is directly granted to apps. The authors of [2] adopted the concept of PermOF and defined a permission set to which apps must subscribe on initialization. Inspired by Android permission system, [8] proposed a permission system based on OpenFlow messages' states that can be used as the unit to which the permission details can be applied. SDNShield [14] presented a permission system with two-level permission abstraction comprised of permission tokens assigned directly to apps and permission filters for limiting token's effective scope.

The permission system in [4] provides a registration service which apps can use to register themselves. It incorporates app identification as well as a permission negotiation process. The authors in [7] introduced AEGIS to prevent against malicious network apps. Security access rules are defined based on the relationships between

Table 5: Configuration of the Formal Access Control Model defined in Table 2 for the Use Case Scenario in Section 5.

$A = \{LS, LB, NIP, FW, OC\}$,
$R = \{APP, SEC, ADMIN\}$ with a total order > on R ,as defined in Table 2,
$T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, t_{18}\}$, as labled in Table 1,
$AR = \{(LS, APP), (LB, APP), (NIP, SEC), (FW, SEC), (OC, ADMIN)\}$,
$TR = \{(t_i, APP), (t_{13}, SEC), (t_j, ADMIN) \mid (t_i \in T \mid 1 \leq i \leq 12, t_j \in T \mid 14 \leq j \leq 18))$,
$DXOP = \{'add\ flow\ rule', 'packet\ in', 'flow\ stats', 'packet\ out'\}$,
$Type('add\ flow\ rule') = '\ Flow\ rule\ mod', Type('packet\ in') = '\ Packet - In\ return',$
$Type('flow\ stats') = '\ Switch\ stats\ request' = '\ Switch\ stats\ report', Type('packet\ out') = '\ Packet - Out',$
$AuthorizationRule(LS, '\ add\ flow\ rule') = true, AuthorizationRule(LB, '\ add\ flow\ rule') = true,$
$AuthorizationRule(FW, '\ add\ flow\ rule') = true,$
$AuthorizationRule(LS, '\ packet\ in') = true, AuthorizationRule(LB, '\ packet\ in') = true, AuthorizationRule(NIP, '\ packet\ in') = true,$
$AuthorizationRule(FW, '\ packet\ in') = true\ AuthorizationRule(OC, '\ packet\ in') = true,$
$AuthorizationRule(LB, '\ flow\ stats') = true, AuthorizationRule(FW, '\ packet\ out') = true.$

apps and data in the SDN controller. Their system uses API hooking to intercept the app execution flow to protect the controller. Prior to SE-Floodlight, FortNOX [22] implements a role-based authorization with three roles. SE-Floodlight [9] is an extension and improvement of the FortNOX. Tseng et al [15], inspired by [9], proposed Controller-DAC with API requests threshold and a priority for each app assigned either directly or via the role.

SM-ONOS [5] proposed a permission system at four-level granularity. First, code packages are classified as either app or non-app OSGi bundles. Next, app bundles are assigned either admin or user role with the appropriate permissions. Non-administrative API-permissions then granted to apps followed by network-level permissions. Based on API-level permissions from SM-ONOS, [3] proposed information flow control among apps for the ONOS controller.

8 CONCLUSION AND FUTURE WORK

In this paper, we formalized a role-based authorization model for SDN using SE-Floodlight as a reference controller and proposed an administration model. Then we showed a configuration of the formal model for a use case scenario. We then discussed the security aspects of the authorization model and described some security problems related to over-privileged apps, limitations of role hierarchy, app upgrading, and app downgrading problems. Finally, we proposed a solution to overcome the mentioned problems.

As a future work, first, we plan to design an access control model that includes a wider set of SDN operations. Second, we plan to present a sophisticated access control model using RBAC standard and Attribute-based access control (ABAC) terminology in order to achieve a fine-grained access control within a holistic view to resources in SDN environment.

ACKNOWLEDGMENTS

This work is partially supported by NSF CREST Grant HRD-1736209 and DoD ARL Grant W911NF-15-1-0518.

REFERENCES

[1] Ijaz Ahmad, Suneth Namal, Mika Ylianttila, and Andrei Gurtov. 2015. Security in software defined networks: A survey. *IEEE Communications Surveys & Tutorials* 17, 4 (2015), 2317–2346.
[2] Scott-Hayward el al. 2014. Operationcheckpoint: Sdn application control. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*. IEEE.
[3] B. Ujcich et al. 2018. Cross-App Poisoning in Software-Defined Networking. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 648–663.
[4] C. Banse et al. 2015. A secure northbound interface for sdn applications. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, Vol. 1. IEEE, 834–839.
[5] C. Yoon et al. 2017. A Security-Mode for Carrier-Grade SDN Controllers. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM.
[6] D. Kreutz et al. 2013. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 55–60.
[7] H. Padekar et al. 2016. Enabling dynamic access control for controller applications in software-defined networks. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*. ACM, 51–61.
[8] J. Noh et al. 2016. Vulnerabilities of network OS and mitigation with state-based permission system. *Security and Communication Networks* 9, 13 (2016).
[9] Porras P. A et al. 2015. Securing the Software Defined Network Control Layer.. In *NDSS*.
[10] R. Sherwood et al. 2010. Can the production network be the testbed?. In *OSDI*.
[11] Scott-Hayward et al. 2013. SDN security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*. IEEE, 1–7.
[12] Scott-Hayward et al. 2016. A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials* 18, 1 (2016), 623–654.
[13] X. Wen et al. 2013. Towards a secure controller platform for openflow applications. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 171–172.
[14] X. Wen et al. 2016. Sdnshield: Reconciliating configurable application permissions for sdn app markets. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*. IEEE, 121–132.
[15] Y. Tseng et al. 2017. Controller DAC: Securing SDN controller with dynamic access control. In *Communications (ICC), IEEE International Conference on*. IEEE.
[16] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. 2001. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)* 4, 3 (2001), 224–274.
[17] Security Enhanced Floodlight. 2018. https://www.sdxcentral.com/projects/openflow-sec-security-enhanced-floodlight/.
[18] Floodlight-Project. 2018. http://www.projectfloodlight.org/.
[19] Ryu SDN Framework. 2018. http://osrg.github.io/ryu/.
[20] ON.Lab. ONOS. 2018. http://onosproject.org/.
[21] The OpenDaylight platform. 2018. https://www.opendaylight.org/.
[22] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. 2012. A security enforcement kernel for OpenFlow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*. ACM.
[23] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. 1996. Role-based access control models. *Computer* 29, 2 (1996), 38–47.
[24] Changhoon Yoon, Taejune Park, Seungsoo Lee, Heedo Kang, Seungwon Shin, and Zonghua Zhang. 2015. Enabling security functions with SDN: A feasibility study. *Computer Networks* 85 (2015), 19–35.