A Model for the Administration of Access Control in Software Defined Networking using Custom Permissions

Abdullah Al-Alaj
Institute for Cyber Security
C-SPECC
Department of Computer Science
UTSA, San Antonio
Texas, USA
abdullah.al-alaj@utsa.edu

Ravi Sandhu
Institute for Cyber Security
C-SPECC
Department of Computer Science
UTSA, San Antonio
Texas, USA
ravi.sandhu@utsa.edu

Ram Krishnan
Institute for Cyber Security
C-SPECC
Department of Electrical and Computer Engineering
UTSA, San Antonio
Texas, USA
ram.krishnan@utsa.edu

***Abstract*—Role-based access control (RBAC) has been widely studied and applied in many domains including Software Defined Networks (SDN). Because the motivation behind adopting RBAC for SDN is to simplify the administration of network app authorizations, having an administrative model is a key component for managing the associations between different SDN entities, and thus determining the access rights of network apps. Currently, SDN environment is lacking such administrative model. Moreover, the operations provided by SDN services are coarse grained, which make it difficult to create administrative units necessary for access control administration. To address these problems, in this paper, we introduce an approach for creating custom SDN operations to extend the capabilities of SDN services and provide fine grained custom permissions specialized for the administration of access control in SDN. Then, with these extended features, we present SDN-RBACa, an administrative model to manage access control actions that define authorizations of network apps.**

Through proof of concept prototype and use cases, we demonstrate the usability of custom permissions and show how custom permissions enable and facilitate the administration of access control in SDNs.

***Index Terms*—Software Defined Networking, Security and privacy, Access control, Formal models, Network security.**

I. Introduction

The centralized SDN controller in conjunction with network operations provided by controller services result in a programmable network. This programmability allows network administrators to provide network services that enable more flexible, customized, and intelligent networking through apps. SDN offers the possibility for SDN apps to further extend the functionality of the network. These features and more make SDN suitable for technologies like Cloud Computing [1] and IoT [2].

Access rights of network apps must follow the minimum privilege principle. Recently, various methods have been proposed for adopting RBAC for the management of access rights of network apps [3]–[6]. Thus, administration of access rights of network apps is inevitable. In large SDNs and possibly large number of network apps, and with the possibility of an increased number of network services provided by the controller, the number of roles can be in the hundreds or thousands, and apps can be in the tens, hundreds or thousands. Managing permissions, roles, apps, and their interrelationships could be a tremendous task which need simplification.

In a prior work, we presented SDN-RBAC [3], a role based access control model for SDN apps. Because the motivation behind adopting RBAC [7], [8] for SDN is to simplify the administration of app authorizations, and because the most commonly carried out administrative activities in SDN-RBAC are maintaining the app-role and permission-role relations, in this paper we present an extension to SDN-RBAC operational model by introducing tasks, and then we present an administrative model, referred to as SDN-RBACa, for administering app-role and task-role relations. To the best of our knowledge, this is the first time in the literature a model is presented for the administration of access control in SDN.

For designing our administrative model, we adopt concepts from Uni-ARBAC [9] administrative model because it combines many of the administrative principles and novel concepts from many administrative models in the literature [10]–[16]. So, in our model, instead of administering individual permissions, permissions are combined into tasks which are assigned to roles as a unit. Moreover, roles and tasks are partitioned and assigned into administrative (or admin) units. Apps are assigned to app-pools from where individual apps are assigned to roles. Administrative users in an admin unit can assign apps to roles only if these apps (via app pools) and these roles are assigned to the admin unit in which this user is a member.

The rest of the paper is organized as follows. In Section II, we discuss related work. Section III describes administrative units in SDN. In Section IV, we describe an approach to create custom and proxy operations to enable the administration of access control in SDN. We discuss the concept of custom

978-1-7281-8543-9/20/$31.00 ©2020 IEEE
DOI 10.1109/TPS-ISA50397.2020.00032

permissions in Section V. Section VI describes the conceptual SDN-RBACa model and its formal definitions. In Section VII, we describe tasks and roles engineering for SDN using custom permissions. In Section VIII, we describe a proof-of-concept use case and its configuration. In Sections IX and X, we discuss implementation and performance evaluation of the operational model of SDN-RBACa. Finally, Section XI concludes the paper and outlines future work.

II. Literature Review

App authorization for SDN can be classified into two main categories. Firstly, permission-based app authorization which includes techniques wherein apps authorization is driven by direct permission-app assignment [17], [18]. The management of such authorization approach is a widely known problem. Secondly, role-based app authorization [3]–[6]. Because SDN's motivation is to simplify network management, and because RBAC's motivation is to simplify the administration of authorizations, it is very important to think about the administration of access control in SDN. However, none of the previous works addressed the administration of access control for SDN apps. To the best of our knowledge, this is the first time the administration of access control in SDN is discussed in the literature.

III. Administrative Units in SDN

A. The need for Administrative Units in SDN

Small SDN networks with small number of SDN apps and roles could be managed easily by a single administrator or a single admin unit that handles all network functions and all traffic types. As SDN networks grow larger with more apps, however, they become more complex and difficult to centrally manage all access control components and their associations by a single, fully-trusted administrative authority. Thus, access control administration has to be decentralized into multiple partially-trusted administrative authorities which are assigned appropriate power to change portions of the access control state. To satisfy this requirement, in our proposed administrative model, rather than having administrative roles, we adopt the concept of Administrative Units (AU) [9] to decentralize access control administration for SDN-RBAC [3].

In large SDNs, the need for specialized apps to deal with specific network traffic becomes more prominent. For example, Web load balancer, Web Firewall, VoIP load balancer, VoIP Firewall, etc. In order to be able to administer the associations between these apps and their roles, we have to engineer admin units based on the apps' network functions and the corresponding access rights. For example based on traffic types or organizational entities (e.g., department in a campus network, tenant's network slice, etc.).

Engineering of admin units requires that each admin unit manages an exclusive set of roles which is not under the authority of another admin unit. If administration is divided into multiple admin units, each specialized with one traffic type, for example Web admin unit, VoIP admin unit, Email admin unit, and FTP admin unit, this makes each admin unit responsible for managing exclusive set of roles that handle similar network functions. In this case, Web admin unit exclusively manages roles related to network functions that handle Web traffic. Similarly VoIP admin unit exclusively manages roles related to network functions that handle VoIP traffic. In another scenario, if admin units are divided based on organizational entities, multiple tenants for example, then each admin unit manages exclusive roles of one tenant. This admin unit authorizes SDN apps of this tenant (via role assignment) to independently operate on this tenant's resources.

B. Granularity of SDN Network Operations

Practically, if a network operation provides access to a wide range of network resources, and these resources need to be managed by different admin units, this precludes the flexibility in engineering appropriate admin units. The flexibility stems from the presence of operations fine grained enough to provide the convenience in engineering set of roles exclusive for each admin unit.

Because engineering of admin units requires that each admin unit manages an exclusive set of roles and, to some extent, exclusive set of resources which can be accessed by permissions in these roles, it is vital for the operations/APIs exposed by the system under consideration to be fine grained enough to the level necessary to engineer these roles. Otherwise, engineering of such admin units will be infeasible.

Unfortunately, the currant state of the art SDN controllers doesn't provide such fine grained network operations. For example, an app with the permission to add a flow rule can insert a flow rule that manipulates any traffic type. Also, it can insert the flow rule in any switch reachable by the controller. From an administrative point of view, this precludes the coexistence of different admin units for access control in SDN. For example, to engineer Web AU and VoIP AU, it is necessary to engineer roles that only handle web traffic and other roles that only handle VoIP traffic. Each set of roles will be exclusively managed by its respective AU. Such capability is not possible by the native operations currently provided by SDN controllers.

A solution for this problem is to create refined versions of the the coarse grained operations in a way that satisfies the needs of fine grained access control for SDN apps, and enables engineering of various admin units necessary for access control administration. The refined version of an operation is called customized or custom operation as will be described in the following section.

IV. Custom and Proxy Operations

In this context, an SDN controller operation is a java API call submitted by an SDN controller apps to access network resources. We call these operations as *target operations* OP_{Target} since they are the current target by SDN apps and may be a target for the refinement process. We call the refined version as the *custom operation* OP_{custom}. So, a custom operation is the refined version of a target operation.

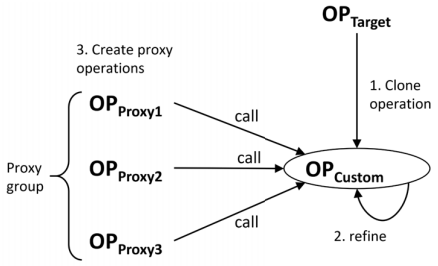

Fig. 1. Target, custom, and proxy operations.

In its simplest form, a custom operation can be created by first cloning the target operation, and then refining its code by adding a fine grained check on the desired attributes based on which an admin unit is defined. For example, because a web admin unit manages web-related roles, this requires the existence of network operations that handle only web traffic and disallow treatment of other traffic types. Thus, the target operation is refined by first creating a cloned custom operation and then adding a check inside the custom operation to make sure that accessed objects are web-related only.

However, in this approach, if each custom operation will check for a specific type of traffic (e.g., web, voip, ftp, email), then multiple custom operations must be created, one for each traffic type. And because custom operations are exact copies of target operations, plus a refinement code added to it, this approach has some problems: i) it significantly increases the number of lines of the native code in SDN controller, ii) it requires extra effort in refining multiple very close custom operations for one target operation, and iii) it increases the compilation time of the controller's code.

To avoid such problems, we create what we call *proxy operations* OP_{Proxy}. Each proxy operation calls one custom operation and passes a parameter value based on which the refinement will be done. The general process for creating custom and proxy operations and their interaction is schematically depicted in Fig. 1. The process starts by cloning the target operation $\text{OP}_{\text{Target}}$ that need to be refined. The new resulted operation $\text{OP}_{\text{Custom}}$ is modified first by adding a new formal parameter to its parameter list. Then it is further modified by adding statements to either check the accessed object against the refinement parameter value or adding statements to filter out unauthorized objects based on the refinement parameter value. Then multiple proxy operations $\text{OP}_{\text{Proxy}_i}$ can be created. Each $\text{OP}_{\text{Proxy}_i}$ contains a simple call to $\text{OP}_{\text{Custom}}$, and is designed to pass a hard coded refinement parameter value. This call passes a parameter value to the custom operation based on which the refinement will be done and specific objects will be accessed. For ease of reference and access control review, the name of the proxy operation should reflect the parameter value passed to its custom operation. Proxy Operations can be considered as abstractions of custom operations.

By customizing the operations in such a way, a proxy operation becomes not only fine-grained, but also expressive, which makes the design of access control policy and its administration simpler. Fig. 2 shows an example of creating a custom operation addFlow(.., traffic) for addFlow operation and then creating three proxy Operations addWebFlow, addVoIPFlow, and addFtpFlow. Using proxy operations makes $\text{OP}_{\text{Custom}}$ and its refinement parameter abstract from the app.

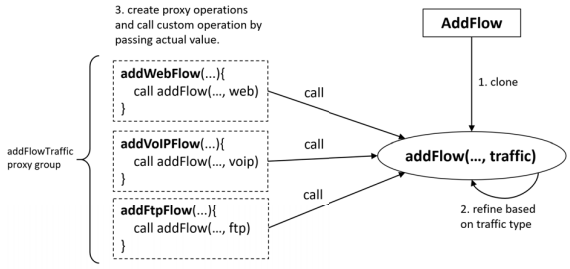

Fig. 2. Example of custom and proxy operations for the target operation addFlow .

V. Custom Permissions

Custom permissions are those permissions that are created using proxy operations. For example, as depicted in Fig. 2, instead of using the coarse-grained target operation addFlow to create the permissions (addFlow, FLOW-RULE), we create the custom operation addFlow(traffic) and its proxy operations addWebFlow and addVoIPFlow, then we create the custom permissions (addWebFlow, FLOW-RULE) and (addVoIPFlow, FLOW-RULE) for adding flow rules that handle Web and VoIP traffic, respectively.

A *proxy group* is the group of all operations that invoke the same custom operation and pass different refinement parameter values. Members in a proxy group allow access to different set of objects. Therefore, permissions composed of different proxy operations in one proxy group allows for the creation of specialized roles. This enables exclusive role management by different admin units.

VI. SDN-RBACa Model

In this section, we describe the SDN-RBACa administrative model, along with its formal definitions. The overall structure of SDN-RBACa is illustrated in Fig. 3. We consider SDN-RBACa in two parts: the operational model for SDN-RBAC with respect to regular roles and permissions as well as tasks which will be introduced shortly, and the administrative model for administering the app role and task-role relations of the former. These are discussed in the following subsections.

A. Introducing Tasks

We view a *task* as a named set of several related permissions that represent a unit of network function for SDN apps. Adopting tasks for SDN-RBAC [3] has some administrative motivations. i) Because custom permissions (see section V) increase the number of total permissions currently available in the SDN controller, using tasks reduces the extra management

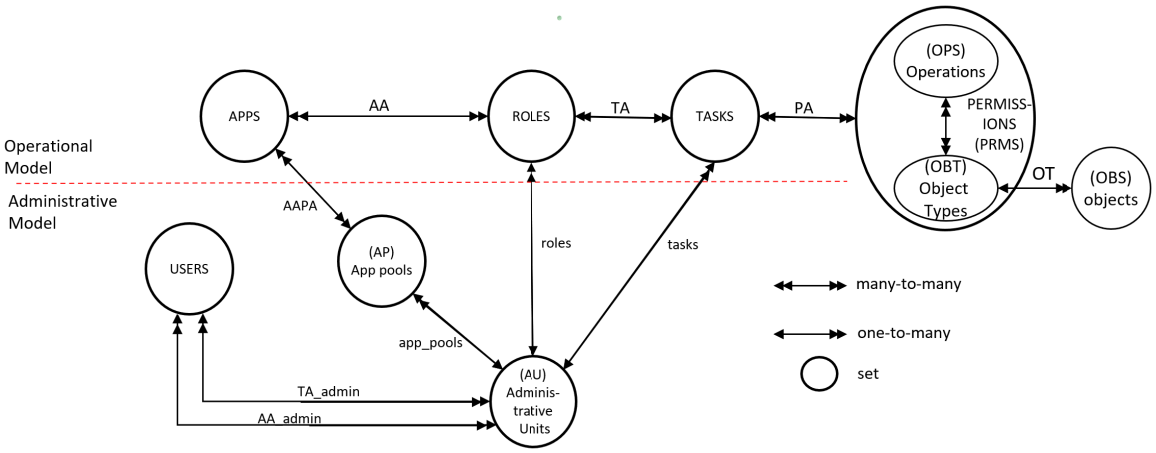

Fig. 3. Conceptual model of SDN-RBACa.

overhead entailed from these newly resulted custom permissions. ii) In role engineering process, task-to-role assignment is a more convenient abstraction than assigning individual permissions, especially when these permissions are related. Therefore, adopting tasks as a basic component in SDN-RBAC reduces administration overhead typically associated with managing fine-grained permissions. In the next subsection we show the SDN-RBACa operational model with tasks as a basic component.

B. SDN-RBACa Operational Model

The sets and relations in the top part of Fig. 3 represent the SDN-RBACa operational model, which is slightly different from the SDN-RBAC model [3]. The most distinguished difference is that there is a level of indirection in role-permission assignment, so permissions are assigned to tasks and tasks are assigned as units to roles. Adopting tasks has several motivations, as discussed in Section VI-A. App-role assignment remains unchanged from SDN-RBAC. For simplicity, we have not considered the SDN-RBAC concepts of sessions and role activation.

The SDN-RBACa operational model is formalized in Table I. The first six components from item 1 specify the basic sets carried over from SDN-RBAC. TASKS is the set of tasks added to SDN-RBAC. The last three sets belong to the administrative model (see section VI-C). Item 2 specifies the assignment relations in the operational model including the additional components which effect the additional indirection between permissions and roles via tasks. Item 3 shows the *type* derived function and shows the *authorized_perms* function which formalizes the interaction between the permission-task and task-role assignments. The authorization function in item 4 specifies the authorization required for an app to exercise a permission and access an object, which is that the permission must be authorized to at least one role assigned to the app.

C. SDN-RBACa Administrative Model

In this section we describe the SDN-RBACa administrative model illustrated in the lower part of Fig. 3, and formalized in Table I. First we use the notion of app-pools. Examples of app-pool include 'Web Load Balance Pool' and 'Web Security Pool' as will be described in the use case in Section VIII. Adopting app-pool facilitates the allocation of several apps that achieve similar network functions to an admin unit. The set of app-pools is denoted as AP. Apps are assigned to app-pools via the AAPA app to app-pool assignment relation which is formally specified in item 5 of Table I.

The set of administrative units is denoted as AU. SDN-RBACa requires that roles are partitioned into different admin units and each role is allocated to exactly one unit for administration. In other words, each admin unit manages an exclusive set of roles which is not under the authority of another admin unit. This roles partitioning is formally specified using the *roles* function in item 6 of Table I. The partitioning concept is further applied to tasks and app-pools via the *tasks* and *app_pools* functions in item 6 of Table I.

The result of roles, tasks, and app-pools partitioning is that an admin unit manages an explicitly assigned partition of roles, to which it can assign apps from an assigned partition of app-pools and tasks from an assigned partition of tasks. The outcome of this partitioning directly impacts the results of administrative user authorization functions specified in item 8 of Table I.

Assignment of administrative users to admin units can be done via the TA_admin or the AA_admin relation. An administrative user in TA_admin is authorized to perform the administrative actions which assign tasks to roles, while a user in AA_admin is authorized to perform the administrative actions which assign apps to roles. It should be mentioned that these capabilities can be separately assigned to two different administrative users, even though they assigned to one administrative unit. Such administrative actions bring apps

TABLE I
FORMAL DEFINITION OF SDN-RBACA ADMINISTRATIVE MODEL.

1.Basic Sets
- APPS is a finite set of SDN apps.
- OPS is a finite set of operations.
- OBS is a finite set of objects.
- OBTS is a finite set of object types.
- PRMS $\subseteq$ OPS $\times$ OBTS , set of permissions.
- ROLES is a finite set of roles.
- TASKS is a finite set of tasks.
- AP is a finite set of app-pools.
- USERS is a finite set of administrative users.
- AU is a finite set of administrative units.

2. Assignment Relations (operational):
- PA $\subseteq$ PRMS $\times$ TASKS, permission-task assignment relation.
- TA $\subseteq$ TASKS $\times$ ROLES, task-role assignment relation.
- AA $\subseteq$ APPS $\times$ ROLES, app-role assignment relation.
- OT $\subseteq$ OBS $\times$ OBTS, a many-to-one mapping an object to its type, where $(o, t_1) \in$ OT $\wedge$ $(o, t_2) \in$ OT $\Rightarrow$ $t_1 = t_2$.

3. Derived Functions (operational):
- type: (o: OBS) $\rightarrow$ OBTS, a function specifying the type of an object. Defined as type(o) = $\{t \in$ OBTS $|$ $(o, t) \in$ OT$\}$.
- authorized_perms(r: ROLES) $\rightarrow 2^{PRMS}$, defined as authorized_perms(r) = $\{p \in$ PRMS $|$ $\exists$t$\in$TASKS, $\exists$r$\in$ROLES : $(t, r) \in$ TA $\bigwedge$ $(p, t) \in$ PA$\}$.

4. App Authorization Function:
- can_exercise_permission(a: APPS, op: OPS, ob: OBS) = $\exists$r $\in$ ROLES : (op, type(ob)) $\in$ authorized_perms(r) $\bigwedge$ (a, r) $\in$ AA.

5. Administrative App-pools Relation:
- AAPA $\subseteq$ APPS $\times$ AP, app to app-pool assignment relation.

6. Administrative Units and Partitioned Assignment:
- roles(au : AU) $\rightarrow 2^{ROLES}$, assignment of roles, where r $\in$ roles(au_1) $\wedge$ r $\in$ roles(au_2) $\Rightarrow$ au_1= au_2.
- tasks(au : AU) $\rightarrow 2^{TASKS}$, assignment of tasks, where t $\in$ tasks(au_1) $\wedge$ t $\in$ tasks(au_2) $\Rightarrow$ au_1= au_2.
- app_pools(au : AU) $\rightarrow 2^{AP}$, assignment of app-pool, where ap $\in$ app_pools(au_1) $\wedge$ ap $\in$ app_pools(au_2) $\Rightarrow$ au_1= au_2.

7. Administrative User Assignment:
- TA_admin $\subseteq$ USERS $\times$ AU.
- AA_admin $\subseteq$ USERS $\times$ AU.

8. **Administrative User Authorization Functions:**
- can_manage_task_role(u : USERS, t : TASKS, r : ROLES) = $\exists$au$\in$AU : (u , au) $\in$ TA_admin $\bigwedge$ r $\in$ roles(au) $\bigwedge$ t $\in$ tasks(au).
- can_manage_app_role(u : USERS, a : APPS, r : ROLES) = $\exists$au$\in$AU : ((u, au) $\in$ AA_admin $\bigwedge$ r $\in$ roles(au)) $\bigwedge$ $\exists$ap$\in$AP : ((a , ap) $\in$ AAPA $\bigwedge$ ap $\in$ app_pools(au)).

9. **Administrative Actions:**
- assign_task_to_role(u: USERS, t: TASKS, r: ROLES) Authorization condition: can_manage_task_role(u, t, r) = True Effect: TA' = TA $\cup$ {(t, r)}.
- revoke_task_from_role(u: USERS, t: TASKS, r: ROLES) Authorization condition: can_manage_task_role(u, t, r) = True Effect: TA' = TA $\setminus$ {(t, r)}.
- assign_app_to_role(u: USERS, a: APPS, r: ROLES) Authorization condition: can_manage_app_role(u, a, r) = True Effect: AA' = AA $\cup$ {(a, r)}.
- revoke_app_from_role(u: USERS, a: APPS, r: ROLES) Authorization condition: can_manage_app_role(u, a, r) = True Effect: AA' = AA $\setminus$ {(a, r)}.

and permissions together and, in some critical SDN networks, they are best to be done by different network administrators.

Item 8 of Table I specifies the authorization functions for administrative users. The function can_manage_task_role returns whether a given admin user can assign/revoke a given task to/from a given role. The requirement is that this user must be assigned as TA_Admin to the unique admin unit which has exclusive authority over this role and this task.

Similarly, can_manage_app_role is an authorization function that returns true or false. This function specifies the conditions for a given user to assign/revoke a given app to/from a given role. The requirement is that this user must be assigned as AA_Admin to the unique admin unit which has exclusive authority over this role and over an app-pool to which this app is directly assigned via AAPA relation.

The last item in Table I formalizes the four administrative actions to assign/revoke a task to/from a role or an app to/from a role. This supports the reversibility principle which requires that administrative actions should be reversible. If an administrative user makes a mistake, they can go back.

VII. TASK AND ROLE ENGINEERING FOR SDN USING CUSTOM PERMISSIONS

In the following two subsections, we discuss the process of engineering tasks and roles using custom permissions. The abstract process is illustrated in Fig. 4 and an example is described in Section VII-B.

A. Engineering Tasks and Roles using Custom Permissions

Because each custom permission is created using a proxy operation, it enables access to a specific fine grained resource known prior to the task or role engineering process. Now, lets compare the use of target operations with proxy operations in creating a permission. As shown in Fig. 4, the three proxy operations (x11, x12, x13) are resulted from refining the target operation op1, and each one provides access to a resource with higher granularity compared to the target operation op1. This makes the custom permission p1 = (x11, ot) more fine grained compared to using op1 to create the same permission, i.e., (op1, ot), where ot is the object type. In turn, because we assign the custom permission p1 to task t1, this makes t1 a fine grained, or more specialized, task. Again, this is compared to using op1 in the first place to engineer the same task. As shown in Fig. 4, task t1 is engineered with the three custom permissions p1, p4, and p7 created using the proxy operations x11, x21, and x31, respectively. Each one provides more fine grained access, and thus makes task t1 more fine grained compared to using the target operations op1, op2, and op3 to create the same task. Notably, this process allows for the creation of more specialized tasks like t2 and t3 in the same way.

The granularity of access resulted from using proxy operations to create custom permissions escalates to roles. For example, roles r1, r2 and r3 in Fig. 4 provide more fine grained

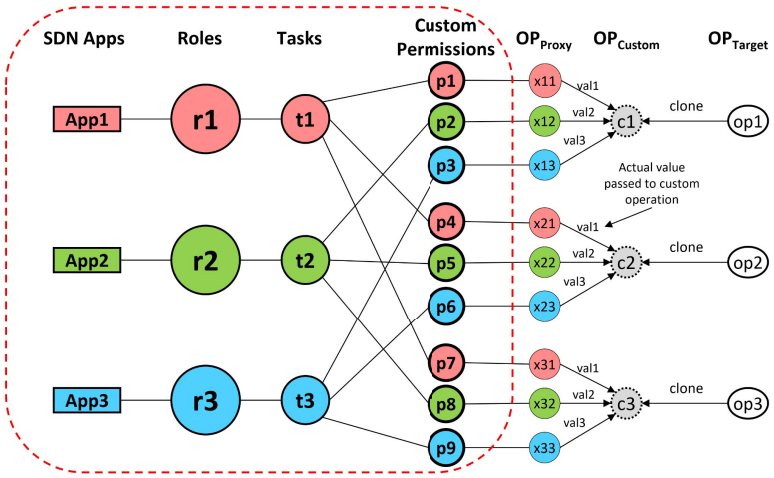

Fig. 4. Conceptual representation of the associations between custom permissions, tasks, roles, and apps.

and specialized access to network resources. Now, imagine that we want to engineer three admin units au1, au2, and au3, each specialized with managing resources accessed by t1, t2, and t3, respectively, then we simply assign each task and to its respective admin unit, and do the same thing with roles r1, r2, and r3. On the contrary, starting the process with op1, op2, and op3 to engineer these roles and tasks preclude the possibility of creating the required admin units.

B. Custom Permissions with 'Flow Mod' Role

In this section, we use the 'Flow Mod' SDN role as an example to illustrate the creation of nine proxy operations for three target operations, namely, addFlow, deleteFlow, and readFlow. The example is depicted in Fig. 5. These target operations allow network apps to access flow rules that handle any type of traffic. However, if it is required to have three admin units, each specialized with one type of traffic, namely, Web, VoIP, and FTP, and if users in these three admin units assign the three target operations to apps (via permissions, tasks, and roles), this means that an app, specialized with Web flows for example, might unauthorizedly access flow rules that handle non-Web traffic, and thus cause threats to the network.

To solve this problem, we refine the three target operation addFlow, deleteFlow, and readFlow and create nine proxy operations classified into three proxy groups addFlowTraffic, deleteFlowTraffic, and readFlowTraffic as shown in Fig. 5. Each proxy operation in a proxy group can handle only one traffic type. Now, for the 'Web Admin Unit', which is specialized with Web traffic, three custom permissions, namely, (addWebFlow, FLOW-RULE), (deleteWebFlow, FLOW-RULE), and (readWebFlow, FLOW-RULE), will be created by picking the web-related proxy operation from each proxy group. These three custom permissions contribute to the engineering of the web-related tasks 'Web Traffic Forwarding' and 'Web Flow Viewing', which will be under exclusive authority of 'Web Admin Unit'. These two tasks will be assigned to the role 'Web Flow Mod', which also will be under exclusive authority of the same admin unit. This role can be assigned only by administrative users who are members in 'Web Admin Unit' to apps that handle web traffic and can be managed, via app-pools, by the same admin unit, such as 'Web Intrusion Prevention' app. The same idea applies to 'VoIP Flow Mod' and 'FTP Flow Mod' roles.

VIII. Proof of Concept Use Cases

A. Basic Use Case - Web Admin Unit

In this section we discuss a proof-of-concept use case to demonstrate the use of custom permissions in enabling the administration of SDN-RBAC. The use case is configured in the SDN-RBACa administrative model as shown in Table II.

The use case describes a scenario in which we have one admin unit, called 'Web Admin Unit'. This admin unit is specialized in managing web resources. It exclusively manages five web-related roles as listed in the set ROLES in item 1 of Table II. It also exclusively manages ten web-related tasks listed in the set TASKS. All these roles and tasks provide access to web resources, such as flow rules that handle web traffic, packet-in headers and payloads that contain web traffic, web pool servers, statistics about web flows, etc. These resources can be accessed via twenty six custom permissions as listed in the set PRMS. All these custom permissions are created using the proxy operations listed in the set OPS. The admin unit 'Web Admin Unit' manages the two web-related app-pools 'Web Load Balance Pool' and 'Web Security Pool' listed in the set AP. Members in these two pools are the three network apps, 'Web Intrusion Prevention', 'Web Application Firewall', and 'Web Load Balancer', specialized in web traffic, and thus require access to web resources. The apps are listed in the set APPS. The relation between the two app-pools and the three apps are specified in the AAPA relation shown in item 3 of Table II.

The functions *roles*, *tasks*, and *app_pools* in item 4 show the partitioned assignment of the five web-related roles, ten web-

TABLE II
CONFIGURATION OF THE ADMINISTRATIVE MODEL FOR THE USE CASE DESCRIBED IN SECTION VIII-A.

1.Basic Sets

- APPS = {Web Intrusion Prevention App, Web Application Firewall App, Web Load Balancer App}.
- ROLES = {Web Packet-In Handler, Web Packet Monitor, Web Flow Mod, Web Load Balancing, Web Stats Collector}.
- OPS = {readWebPacketInPayload, readWebPacketHeader, readWebRule, insertWebRule, updateWebRule, deleteWebRule, createWebPool, listWebPools, removeWebPool, updateWebPool, createWebMonitor, listWebMonitors, removeWebMonitor, updateWebMonitor, createWebVip, listWebVips, removeWebVip, updateWebVip, createWebMember, listWebMembersByPool, removeWebMember, updateWebMember, readWebFlowByteCount, readAggWebFlowByteCount, readWebFlowPacketCount, readAggWebFlowPacketCount}.
- OBTS = {PI-PAYLOAD, PI-HEADER, FLOW-RULE, LB-POOL, LB-MONITOR, LB-VIP, LB-POOL-MEMBER, FLOW-STATS}.
- OBS = set of all objects of types PI-PAYLOAD, PI-HEADER, FLOW-RULE, LB-POOL, LB-MONITOR, LB-VIP, LB-POOL-MEMBER, and FLOW-STATS.
- PRMS = {(readWebPacketInPayload, PI-PAYLOAD), (readWebPacketHeader, PI-HEADER), (readWebRule, FLOW-RULE), (insertWebRule, FLOW-RULE), (updateWebRule, FLOW-RULE), (deleteWebRule, FLOW-RULE), (createWebPool, LB-POOL), (listWebPools, LB-POOL), (removeWebPool, LB-POOL), (updateWebPool, LB-POOL), (createWebMonitor, LB-MONITOR), (listWebMonitors, LB-MONITOR), (removeWebMonitor, LB-MONITOR), (updateWebMonitor, LB-MONITOR), (createWebVip, LB-VIP), (listWebVips, LB-VIP), (removeWebVip, LB-VIP), (updateWebVip, LB-VIP), (createWebMember, LB-POOL-MEMBER), (listWebMembersByPool, LB-POOL-MEMBER), (removeWebMember, LB-POOL-MEMBER), (updateWebMember, LB-POOL-MEMBER), (readWebFlowByteCount, FLOW-STATS), (readAggWebFlowByteCount, FLOW-STATS), (readWebFlowPacketCount, FLOW-STATS), (readAggWebFlowPacketCount, FLOW-STATS)}.
- AP = {Web Load Balance Pool, Web Security Pool}.
- TASKS = {Web Deep Packet Inspection Task, Web Packet Header Inspection Task, Web Flow Viewing Task, Web Traffic Forwarding Task, Web Server Pool Management Task, Web Server Monitor Management Task, Web Pool VIP Management Task, Web Pool Member Management Task, Web Payload Statistics Collection Task, Web Packet Statistics Collection Task}.
- USERS = {web_functions_admin_user, web_apps_admin_user}.
- AU = {Web Admin Unit}.

2. Assignment Relations:

- PA = {{(readWebPacketInPayload, PI-PAYLOAD), (readWebPacketHeader, PI-HEADER), (readWebRule, FLOW-RULE)} ×
  {Web Deep Packet Inspection Task} ∪
  {(readWebPacketHeader, PI-HEADER), (readWebRule, FLOW-RULE)} × {Web Packet Header Inspection Task} ∪
  {(readWebRule, FLOW-RULE)} × {Web Flow Viewing Task} ∪
  {(insertWebRule, FLOW-RULE), (updateWebRule, FLOW-RULE), (deleteWebRule, FLOW-RULE)} × {Web Traffic Forwarding Task} ∪
  {(createWebPool, LB-POOL), (listWebPools, LB-POOL), (removeWebPool, LB-POOL), (updateWebPool, LB-POOL)} ×
  {Web Server Pool Management Task} ∪
  {(createWebMonitor, LB-MONITOR), (listWebMonitors, LB-MONITOR), (removeWebMonitor, LB-MONITOR),
  (updateWebMonitor, LB-MONITOR)} × {Web Server Monitor Management Task} ∪
  {(createWebVip, LB-VIP), (listWebVips, LB-VIP), (removeWebVip, LB-VIP), (updateWebVip, LB-VIP)} × {Web Pool VIP Management Task} ∪
  {(createWebMember, LB-POOL-MEMBER), (listWebMembersByPool, LB-POOL-MEMBER), (removeWebMember, LB-POOL-MEMBER),
  (updateWebMember, LB-POOL-MEMBER)} × {Web Pool Member Management Task} ∪
  {(readWebFlowByteCount, FLOW-STATS), (readAggWebFlowByteCount, FLOW-STATS)} × {Web Payload Statistics Collection Task} ∪
  {(readWebFlowPacketCount, FLOW-STATS), (readAggWebFlowPacketCount, FLOW-STATS)} × {Web Packet Statistics Collection Task}}.
- TA = {{Web Deep Packet Inspection Task, Web Packet Header Inspection Task} × {Web Packet-In Handler} ∪
  {Web Packet Header Inspection Task} × {Web Packet Monitor} ∪
  {Web Flow Viewing Task, Web Traffic Forwarding Task} × {Web Flow Mod} ∪
  {Web Server Pool Management Task, Web Server Monitor Management Task, Web Pool VIP Management Task,
  Web Pool Member Management Task} × {Web Load Balancing} ∪
  {Web Payload Statistics Collection Task, Web Packet Statistics Collection Task} × {Web Stats Collector}}.
- AA = {{Web Intrusion Prevention App} × {Web Packet-In Handler, Web Flow Mod} ∪ {Web Application Firewall App} ×
  {Web Packet Monitor, Web Flow Mod} ∪ {Web Load Balancer App} × {Web Flow Mod, Web Load Balancing, Web Stats Collector}}.
- OT = {(all payloads in packet-in message, PI-PAYLOAD), (all packet header objects, PI-HEADER), (all flow-rules, FLOW-RULE),
  (all server pools, LB-POOL), (all server monitors, LB-MONITOR), (all pools virtual IPs, LB-VIP), (all pool members, LB-POOL-MEMBER),
  (all flow statistics in flow rules, FLOW-STATS)}.

3. App-pools Relation:

- AAPA = {(Web Intrusion Prevention App, Web Security Pool), (Web Application Firewall App, Web Security Pool),
  (Web Load Balancer App, Web Load Balance Pool)}.

4. AU and Partitioned Assignment:

- roles(Web Admin Unit) = {Web Packet-In Handler, Web Packet Monitor, Web Flow Mod, Web Load Balancing, Web Stats Collector}.
- tasks(Web Admin Unit) = {Web Deep Packet Inspection Task, Web Packet Header Inspection Task, Web Flow Viewing Task,
  Web Traffic Forwarding Task, Web Server Pool Management Task, Web Server Monitor Management Task, Web Pool VIP Management Task,
  Web Pool Member Management Task, Web Payload Statistics Collection Task, Web Packet Statistics Collection Task}.
- app_pools(Web Admin Unit) = {Web Load Balance Pool, Web Security Pool}.

5. Administrative User Assignment:

- TA_admin = {(web_functions_admin_user, Web Admin Unit)}.
- AA_admin = {(web_apps_admin_user, Web Admin Unit)}.

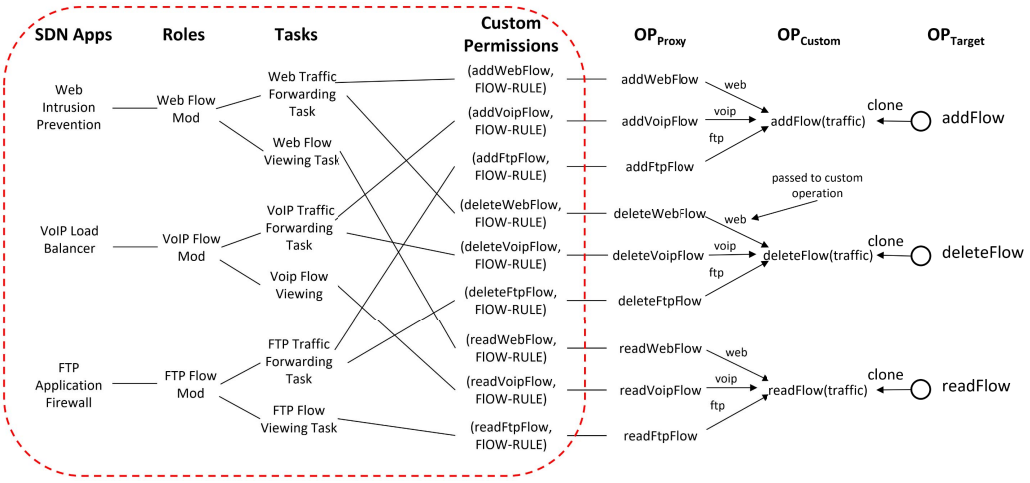

Fig. 5. Example of creating three roles using custom permissions and their associations with tasks and apps.

related tasks, and two web-related app-pools to the admin unit 'Web Admin Unit'. This admin unit has two administrative users, web_functions_admin_user and web_apps_admin_user. The former is authorized, via TA_admin relation, to assign/revoke tasks to/from roles, and the later is authorized, via AA_admin relation, to assign/revoke apps to/from roles. These two relations are specified in item 5 of Table II.

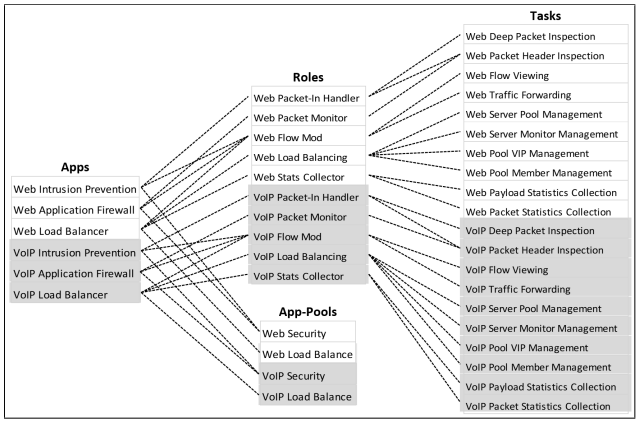

Fig. 6. 'Web Admin Unit' and 'VoIP Admin Unit' (gray) along with tasks, roles, and app pools they exclusively manage. The figure also shows apps that admin units can manage via app-pools.

B. Extended Use Case - Multiple Administrative Units

In this section, we describe an extension to the use case of Section VIII-A and show how we can use multiple proxy operations from each proxy group to engineer multiple admin units. The use case is depicted in Fig. 6 with the two admin units 'Web Admin Unit' and 'VoIP Admin Unit'.

The VoIP-related tasks in Fig. 6 are engineered in the same way Web-related tasks are engineered. VoIP-related tasks are engineered using custom permissions which are created using VoIP-related proxy operations. For example, three custom permissions, namely, (addVoIPFlow, FLOW-RULE), (deleteVoIPFlow, FLOW-RULE), and (readVoIPFlow, FLOW-RULE) will be used to engineer the tasks 'VoIP Traffic Viewing' and 'VoIP Traffic Forwarding'. Both of these tasks will contribute to the engineering of 'VoIP Flow Mod' role.

Using the same approach, we can create other admin units, for example, 'Ftp Admin Unit' and 'Email Admin Unit'. It is clear that, by the power of proxy operations and the custom permissions created from them, it becomes more flexible to create more admin units, each one specialized with different type of traffic.

Table III shows examples of administrative user authorizations corresponding to some administrative actions based on the extended use case in this section. The table shows the results of the authorization function. The use case assumes the existence of four administrative users assigned to the two admin units as specified in Table IV.

IX. Implementation

To demonstrate the effectiveness of custom permissions with our access control model, we implemented a prototype on Floodlight, a Java based SDN controller. We developed and ran the prototype in Floodlight SDN controller v1.2 release [19]. The Floodlight platform is deployed on a virtual machine that has 8GB of memory and runs on Ubuntu 14.04 OS installation. We created a topology with three virtual switches (Open vSwitch v2.3.90) connected to each other and each switch is connected to two hosts. Switches are connected to the

```
The method net.floodlightcontroller.staticflowentry.IStaticFlowEntryPusherService.addWebFlow
is called by session net.floodlightcontroller.webtestapp.WebTestAppSession
01:34:14.691 WARN [n.f.rbac.RBAC:Thread-12] SDN-RBAC: security violation, "Access denied".
Unauthorized access requested by session (WebTestAppSession)
Reason: MatchField:TCP_DST - Incorrect port (25) in flow rule
Active roles set for this session: [Web Flow Mod]
01:34:14.824 INFO [n.f.l.i.LinkDiscoveryManager:Scheduled-3] Sending LLDP packets out of all
```

Fig. 7. Screenshot of authorization check result for addWefFlow proxy operation requested by WebTestApp - Access denied because of incorrect tcp_port number.

TABLE III
EXAMPLES OF ADMINISTRATIVE USER AUTHORIZATION FUNCTIONS CORRESPONDING TO SOME ADMINISTRATIVE ACTIONS. EXAMPLES BELONG TO EXTENDED USE CASE IN SECTION VIII-B

1 . Examples of Authorization Functions:
– can_manage_task_role(web_functions_admin_user, Web Traffic Forwarding Task, Web Flow Mod) = True ***Reason***: ∃Web Admin Unit ∈ AU : ((web_functions_admin_user, Web Admin Unit) ∈ TA_admin) ⋀ Web Flow Mod ∈ roles(Web Admin Unit) ⋀ Web Traffic Forwarding Task ∈ tasks(Web Admin Unit).
– can_manage_task_role(voip_functions_admin_user, Web Server Pool Management Task, Web Load Balancing) = False ***Reason***: Web Load Balancing ∈ roles(Web Admin Unit) ⋀ Web Server Pool Management Task ∈ tasks(Web Admin Unit), however, (voip_functions_admin_user, Web Admin Unit) ∉ TA_admin.
– can_manage_app_role(web_apps_admin_user, Web Intrusion Prevention App, Web Flow Mod) = True ***Reason***: ∃Web Admin Unit ∈ AU : ((web_apps_admin_user, Web Admin Unit) ∈ AA_admin) ⋀ Web Flow Mod ∈ roles(Web Admin Unit)] ⋀ ∃Web Security Pool ∈ AP: (Web Intrusion Prevention App. Web Security Pool) ∈AAPA ⋀ Web Security Pool ∈ app_pools(Web Admin Unit).
– can_manage_app_role(web_apps_admin_user, VoIP Application Firewall App, VoIP Flow Mod) = False ***Reason***: VoIP Flow Mod ∈ roles(VoIP Admin Unit) ⋀ (VoIP Application Firewall App , VoIP Security) ∈ AAPA ⋀ VoIP Security ∈ app_pools(VoIP Admin Unit)], however, (web_apps_admin_user, VoIP Admin Unit) ∉ AA_admin.
2 . Examples of Administrative Actions:
– assign_task_to_role(web_functions_admin_user, Web Traffic Forwarding Task, Web Flow Mod) is allowed ***Reason***: can_manage_task_role(web_functions_admin_user, Web Traffic Forwarding Task, Web Flow Mod) = True
– revoke_task_from_role(voip_functions_admin_user, Web Server Pool Management Task, Web Load Balancing) is not allowed ***Reason***: can_manage_task_role(voip_functions_admin_user, Web Server Pool Management Task, Web Load Balancing) = False
– assign_app_to_role(web_apps_admin_user, Web Intrusion Prevention App, Web Flow Mod) is allowed ***Reason***: can_manage_app_role(web_apps_admin_user, Web Intrusion Prevention App, Web Flow Mod) = True
– revoke_app_from_role(web_apps_admin_user, VoIP Application Firewall App, VoIP Flow Mod) is not allowed ***Reason***: can_manage_app_role(web_apps_admin_user, VoIP Application Firewall App, VoIP Flow Mod) = False

TABLE IV
ADMINISTRATIVE USER ASSIGNMENT RELATION FOR USE CASE IN SECTION VIII-B

Administrative User Assignment:
– TA_admin = {(web_functions_admin_user, Web Admin Unit), (voip_functions_admin_user, VoIP Admin Unit)}.
– AA_admin = {(web_apps_admin_user, Web Admin Unit), (voip_apps_admin_user, VoIP Admin Unit)}.

controller and hosts are virtual machines that has 2GB and run Ubuntu 14.04 OS server. We implemented the access control using AspectJ [20], a seamless aspect-oriented extension to Java. AspectJ ensures that all access requests (i.e., calls to proxy operations) from apps are intercepted by our access control components. This system can be deployed to all other Java-based SDN controllers.

We created a simple test app, WebTestApp and assigned it to the role 'Web Flow Mod'. Thus, it can access web flow rules only. We designed the app to insert a flow rule with the matching field TCP_DST equals to the SMTP port 25. Our refined custom operation addFlow is designed to consider ports 80 and 443 for web traffic. As a result, the proxy operation addWebFlow allows only ports 80 and 443 to be used for flow rule insertions and denies any rule with other ports. The purpose of this test app is to demonstrate how our access control system checks custom permissions and rejects unauthorized access. We created a flow rule and set the TCP_DST match field to 25 using the java instruction: matchbuilder.setExact(MatchField.TCP_DST, TransportPort.of(25));. This causes an access violation since the TCP port number failed the refinement check in the custom permission addFlow. A screen-shot of the output console is shown in Fig. 7.

X. PERFORMANCE EVALUATION

To evaluate the effectiveness of our access control system utilizing custom permissions, we created a test app and selected fifty proxy operations, from which we created fifty custom permissions. These custom permissions are assigned to eighteen tasks and ten different roles. We incrementally assigned these roles to the test app which runs in one session. Despite the fact that this app doesn't require all these roles, the purpose of this test is to check the overhead caused by our access control on the system's performance by reporting

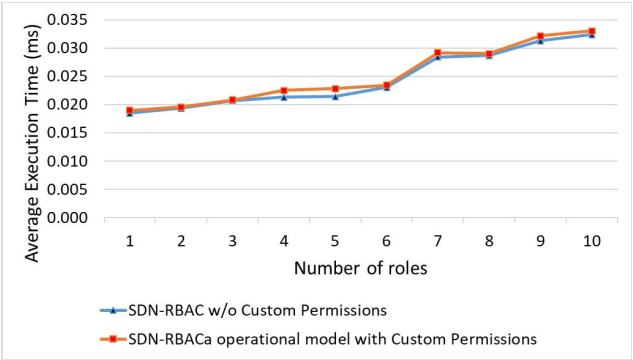

Fig. 8. Average authorization time in SDN-RBAC and SDN-RBACa Operational Model.

the execution time with different security policies. We change the security policy by changing the active role set of the app's session. In the first security policy one role is assigned to the session's active role set, in the second policy two roles where assigned, and so on until ten roles.

For each security policy, the session executes all fifty proxy operations. The system is set to compute the authorization delay imposed by the access control components to finish execution and make an access control decision for each proxy operation submitted by the session. The timer starts when the call is intercepted by AspectJ hook, and stops when the access decision is calculated based on the available custom permissions for the session. The total time is calculated for all fifty proxy operations. We repeated this test hundred times for each security policy. For overhead comparison, we performed the same test on SDN-RBAC, but without custom operations. The average elapsed authorization times calculated for SDN-RBACa operational model and the SDN-RBAC model are reported as shown in Fig. 8. It should be noted here that delay times does not include floodlight's boot-up time, the time for loading the policy and creating the corresponding relations.

This evaluation shows that the authorization check of the SDN-RBACa operational model takes an average of 0.0252 ms on the floodlight controller while SDN-RBAC takes 0.0245 ms on average. As a result, the SDN-RBACa operational model adds an overhead of around 2.9% to the authorization framework. This observed latency is negligible. Therefore, we believe that the operational model of SDN-RBACa with custom permissions introduces acceptable overhead to the controller for the sake of access control administration.

XI. Conclusion and Future Work

In this paper, we introduced an approach for creating custom SDN operations to extend the capabilities of SDN services and provide fine grained custom permissions specialized for the administration of access control in SDN. Then, we presented, SDN-RBACa, an administrative model to manage the associations between network applications and other access control entities. Through proof of concept prototype implementation and use cases, we demonstrated the usability of custom permissions and showed how custom permissions enable and facilitate the administration of access control in SDNs.

In future work, the custom permissions can be further refined and demonstrated in practical use cases and implementations of the administrative model.

Acknowledgment

This work is partially supported by NSF CREST Grant HRD-1736209 and CNS-1553696.

References

[1] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Sdn-based cloud computing networking," in *2013 15th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2013, pp. 1–4.

[2] M. Ojo, D. Adami, and S. Giordano, "A sdn-iot architecture with nfv implementation," in *2016 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2016, pp. 1–6.

[3] A. Al-Alaj, R. Krishnan, and R. Sandhu, "Sdn-rbac: An access control model for sdn controller applications," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*. IEEE, 2019, pp. 1–8.

[4] A. Al-Alaj, R. Sandhu, and R. Krishnan, "A formal access control model for se-floodlight controller," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. ACM, 2019, pp. 1–6.

[5] P. Porras *et al.*, "A security enforcement kernel for openflow networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 121–126.

[6] Y. Tseng *et al.*, "Controller dac: Securing sdn controller with dynamic access control," in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[7] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[8] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[9] P. Biswas, R. Sandhu, and R. Krishnan, "Uni-arbac: A unified administrative model for role-based access control," in *International Conference on Information Security*. Springer, 2016, pp. 218–230.

[10] R. Sandhu, V. Bhamidipati, and Q. Munawer, "The arbac97 model for role-based administration of roles," *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 1, pp. 105–135, 1999.

[11] R. Sandhu and Q. Munawer, "The arbac99 model for administration of roles," in *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*. IEEE, 1999, pp. 229–238.

[12] S. Oh and R. Sandhu, "A model for role administration using organization structure," in *Proceedings of the seventh ACM symposium on Access control models and technologies*, 2002, pp. 155–162.

[13] J. Crampton, "Understanding and developing role-based administrative models," in *Proceedings of the 12th ACM conference on Computer and communications security*, 2005, pp. 158–167.

[14] J. Crampton and G. Loizou, "Administrative scope: A foundation for role-based administrative models," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 2, pp. 201–231, 2003.

[15] N. Li and Z. Mao, "Administration in role-based access control," in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, 2007, pp. 127–138.

[16] H. Wang and S. L. Osborn, "An administrative model for role graphs," in *Data and Applications Security XVII*. Springer, 2004, pp. 302–315.

[17] X. Wen *et al.*, "Towards a secure controller platform for openflow applications," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 171–172.

[18] S. Scott-Hayward *et al.*, "Operationcheckpoint: Sdn application control," in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*. IEEE, 2014, pp. 618–623.

[19] Floodlight-Project. (2020) https://floodlight.atlassian.net.

[20] AspectJ. (2020) Aspectj: A seamless aspect oriented extension to java. https://www.eclipse.org/aspectj/.