SDN-RBAC: An Access Control Model for SDN Controller Applications

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

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

***Abstract*—The architecture of Software-defined Networks provides the flexibility in developing innovative networking applications for managing and analyzing the network from a centralized controller. Since these applications directly and dynamically access critical network resources, any privilege abuse from controller applications could lead to various attacks impacting the entire network domain. As a result, the security concern is ranked one of the top issues that prevent enterprise and data center networks from adopting SDN. Since access control is a natural solution to the over-privilege problem and to address this critical security issue, we propose and implement a formal role-based access control model (*SDN-RBAC*) for SDN applications that helps in applying least of privilege principle at the level of applications and their sessions. We also identify different approaches in which the system can handle application sessions in order to reduce exposure to the network attack surface in case of application being compromised, buggy, or malicious.**

Through proof-of-concept prototype, we implemented our model with multi-session support in Floodlight controller and used hooking techniques to enforce the security policy without any change to the code of the Floodlight framework. The implementation verifies the model's usability and effectiveness against unauthorized access requests by controller applications and shows how the framework can identify application sessions and reject unauthorized operations in real time.

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

I. Introduction

Software Defined networking (SDN) has emerged with the crucial mission to provide high-level network abstraction and programmability. Therefore, SDN promises to provide the scale and versatility necessary for different fields including data centers, Internet of Things (IoT) [1], cloud computing and virtualization [2]. SDN gets more popularity due to the flexibility in developing controller applications (apps) for extending the capabilities of the SDN controller. However, the actual orchestration of dynamically allocating underlying resources to SDN apps with ensuring least of privilege is not trivial. So, a natural solution is building access control models for SDN apps.

978-1-7281-0875-9/19/$31.00 ©2019 IEEE

SDN apps that are residing in the SDN controller and written in the same language of the controller are of a major security concern. This is because they are compiled as part of the controller and have direct access to various controller native classes, their methods and data. Intuitively, the more permissions available to an app the more resources accessible through these permissions, the more exposed the network attack surface. As a result, applying the principle of least privilege is vital in access control for SDN apps. The key idea is to minimize the amount of operations available to an app at a given time.

In SDN, it is most likely that one controller app performs several networking tasks, either sequentially or concurrently. If the app executes all these tasks in one session this means higher exposure to the network attack surface in case of app being compromised, buggy, or malicious. This ensures that cooperation of multiple sessions is required to perform all the tasks of a complete SDN app process, either sequentially or concurrently, so that accountability can be enforced and damage caused by either a session mistake, or an accident or deception can be avoided or minimized.

This initiates the need for serious efforts in creating access control models for SDN controllers where, usually, human being has no direct control of the running apps. To address this issue we propose a Role-based Access Control Model for the apps residing in the SDN controller.

In this context the concept of a session has several motivations. It supports the least privilege principle in the sense that an app can delay the activation of roles currently unused in a session until they really required [3], [4]. Also, it permits delaying the creation of particular sessions until they really required. All these serve to reduce the amount of operations executable by an app at a given time which reduces the amount of resources accessible by these operations and thus the attack surface.

To address the above problems, we propose and implement a formal role-based access control model (*SDN-RBAC*) for SDN applications that helps in applying least of privilege principle at the level of applications and their sessions. We also identify different approaches in which the system can

handle application sessions in order to reduce exposure to the network attack surface.

Our contributions in this paper include the following:

- We present a formal role-based access control model (*SDN-RBAC*) for SDN controller apps.
- We provide the functional system specifications required for app session management and for making access control decisions at the session level.
- We provide different approaches in which the SDN controller can handle session instances of an app.
- We show how an app can be configured in the model via a use case scenario in which an app functionality and its assigned roles are distributed into two tasks which run simultaneously in independent sessions; thus, minimizing the exposure to surface attack in case of any security problem occurs.
- We implement our model, as proof-of-concept prototype, in Floodlight platform using hooking techniques without any change to the code of Floodlight native modules.
- We show the system's usability using a test app with multi-session execution and how it can effectively filter out unauthorized accesses from misbehaving sessions.

This paper is organized as follows. In Section II, we discuss related work. In Section III, we present *SDN-RBAC* formal definition and its functional specifications. Different approaches for handling app sessions are discussed in Section IV. Section V describes a use case configuration with a multi-session app. In Section VI we demonstrate the implementation of our RBAC system in Floodlight platform and show the system's usability using a test app. Section VII discusses performance evaluation of the framework. Finally, Section VIII presents conclusion and outlines future work.

II. Related Work

Several proposals on access control for SDN apps exist in the literature. [5]–[7] described the access control system in terms of the set of operations (APIs) as the basic unit for restricting app's activities. Although in works like [8]–[10], roles were assigned to apps, nevertheless, the operation (API) was the basic construction block of roles. However, we think that, in access control systems, especially role-based ones, it is vital to describe which combinations of operation and object (or object type) exist in the system and constrain their actual use from unauthorized entities.

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 [5] proposed a permission system in which a permission set is directly granted to apps. The authors of [6] adopted the concept of PermOF. Inspired by Android permission system, [7] 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 [11] 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 authors in [12] introduced AEGIS using security access rules with API hooking to intercept app's execution flow to protect the controller.

Managing the aforementioned permission-based authorization systems is a tremendous task which needs simplification. We propose a role based authorization system for SDN to facilitate the administration of app authorization.

FortNOX [8] implements a role-based authorization system with three roles. FortNOX is extended and improved in SE-Floodlight [9]. In our previous work [13], we proposed a formal access control model for SDN apps based on SE-Floodlight as a reference controller and discussed some security aspects based on the model. Tseng et al [10], inspired by [9], proposed Controller-DAC with API request threshold and a priority for each app assigned either directly or via the role. SM-ONOS [14] proposed a permission system at four-level granularity. Based on API-level permissions from SM-ONOS, [15] proposed information flow control among apps for the ONOS controller.

None of the previous role-based approaches build on the concept of sessions for SDN apps which is an remarkable part of standard RBAC model [4]. However, in our work we describe different approaches for handling and deploying sessions in the context of SDN.

III. The *SDN-RBAC* Model

A. Formal Model

In this section we introduce *SDN-RBAC* with its basic element sets and functions. Being able to create roles for SDN apps, which contain optimum number of permissions, is one of the challenges in SDN environment. We believe that deciding which permissions to assign to which roles is completely up to the app's function. Currently, there is no any reference standard that states which kinds of controller apps should use which kind of permissions. There is also no satisfactory system that can identify the permissions appropriate for the different categories of apps. We believe that this topic by itself is a research area that needs further study.

SDN-RBAC has the following five basic components : Controller Apps ($APPS$), Roles ($ROLES$), Operations (OPS), Objects (OBS), and Object Types ($OBTS$). The conceptual model and the relations between the components of *SDN-RBAC* are shown in Fig.1, and discussed below.

- **Apps** $(APPS)$**:** The set of OpenFlow apps residing in the SDN controller.
- **Roles** $(ROLES)$**:** The authorization roles assigned to apps.
- **Objects** (OBS)**:** Data and objects (resources) managed by the controller and should be protected from unauthorized access. The controller manages these resources to maintain a consistent state of the network infrastructure. A specific port instance in a particular switch instance, and a device instance are examples of objects.

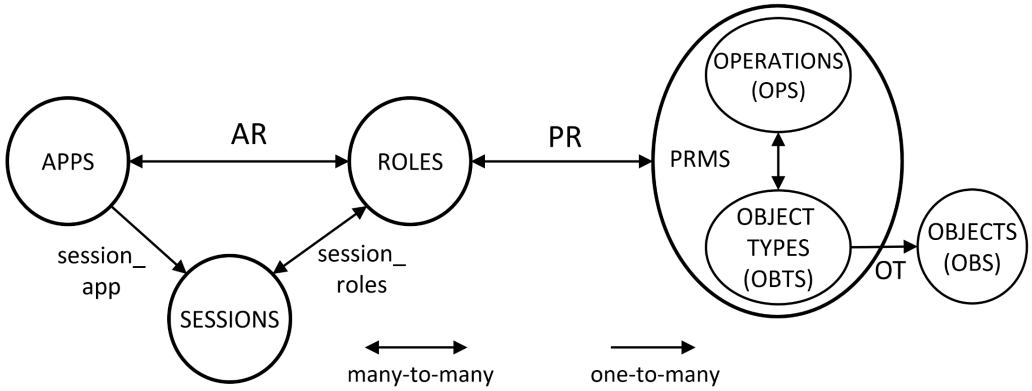

Fig. 1. Conceptual *SDN-RBAC* Model.

TABLE I
FORMAL DEFINITIONS OF *SDN-RBAC*.

-	**Model Element Sets:**
-	$APPS$, $ROLES$, OPS, OBS and $OBTS$, a finite set of OpenFlow apps, roles, operations, objects and object types, respectively.
-	$PRMS = 2^{OPS \times OBTS}$, the set of permissions.
-	$SESSIONS$, a finite set of sessions.
-	**Assignment Relations:**
-	$PR \subseteq PRMS \times ROLES$, a many-to-many mapping permission-to-role assignment relation.
-	$AR \subseteq APPS \times ROLES$, a many-to-many mapping app-to-role assignment relation.
-	$OT \subseteq OBS \times OBTS$, a many-to-one relation mapping an object to its type.
-	**Mapping Functions**
-	$assigned\_perms(r : ROLES) \rightarrow 2^{PRMS}$, the mapping of role *r* into a set of permissions. Formally, $assigned\_perms(r) \subseteq \{p \in PRMS \mid (p, r) \in PR\}$.
-	$app\_sessions(a : APPS) \rightarrow 2^{SESSIONS}$, the mapping of an app into a set of sessions.
-	$session\_app(s : SESSIONS) \rightarrow APPS$, the mapping of session into the corresponding app.
-	$session\_roles(s : SESSIONS) \rightarrow 2^{ROLES}$, the mapping of session into a set of roles. Formally, $session\_roles(s) \subseteq \{r \in ROLES \mid (session\_app(s), r) \in AR\}$.
-	$type : OBS \rightarrow OBTS$, a function specifying the type of an object, where $(o, t_1) \in OT \wedge (o, t_2) \in OT \Rightarrow t_1 = t_2$
-	$avail\_session\_perms(s : SESSIONS) \rightarrow 2^{PRMS}$, the permissions available to an app in a session = $\bigcup_{r \in session\_roles(s)} assigned\_perms(r)$.

- **Object Types** $(OBTS)$**:** The type under which a specific object instance or group of object instances are categorized. For example, all port instances within the authority of $(VLANid = 10)$ can be associated to the type 'PORT-VLAN-5' and all ports within $(VLANid = 10)$ can be associated to the type 'PORT-VLAN-10'. Also, 'LINK-ACC' could be the type of all link instances attached to the hosts in the Accounting Department.
- **Operations (**OPS**):**. Operations performed on objects and exposed by the controller as a service. For example, the Device Service exposes operations to query the list of devices/hosts based on one of Mac Address, VLAN id, IPv4 Address, IPv6 Address, or a combination of them. Inserting flow rules and reading switch statistics are another examples of operations.
- **Sessions (**$SESSIONS$**):** A mapping between an app and an activated subset of app roles. An app can have multiple sessions and a session belongs to only one app.

As shown in Table I, permissions $PRMS$ is the set of all possible combinations between the set of operations OPS and object types $OBTS$. The function $type$ returns the type of an object that was associated to it via the OT relation.

A role can be assigned multiple permissions as expressed by the PR relation. An app might need to have multiple permissions to access different network resources which may result in requiring multiple roles. This is expressed by the AR relation. The function $assigned\_perms$ is used by the system to get the set of permissions attached to a role.

An app may have multiple session instances running at the same time. Each session may have different combinations of active roles adequate for accomplishing its task. It is important for authorization purposes to identify all instances of an app sessions, the system uses the function $app\_sessions$ for this purpose.

Each session executes on behalf of only one app which is constant during the session's lifetime. The system uses the function $session\_app$ to identify this app. Generally, at the time of session establishment and during the lifetime of a

TABLE II
SPECIFICATIONS OF SYSTEM FUNCTIONS.

Function	Authorization Condition	Update
$createSession(a : APPS, s : SESSIONS, ars : 2^{ROLES})$	$ars \subseteq \{r \in ROLES \mid (a, r) \in AR\} \wedge$ $s \notin SESSIONS$	$SESSIONS' = SESSIONS \cup \{s\}, app\_sessions'(a) = app\_sessions(a) \cup \{s\}, session\_roles'(s) = ars$
$deleteSession(a : APPS, s : SESSIONS)$	$s \in app\_sessions(a)$	$app\_sessions'(a) = app\_sessions(a) \backslash \{s\}$, $SESSIONS' = SESSIONS \backslash \{s\}$
$addActiveRole(a : APPS, s : SESSIONS, r : ROLES)$	$s \in app\_tsessions(a) \wedge (a, r) \in AR \wedge$ $r \notin session\_roles(s)$	$session\_roles'(s) = session\_roles(s) \cup \{r\}$
$dropActiveRole(a : APPS, s : SESSIONS, r : ROELS)$	$s \in app\_sessions(a) \wedge$ $r \in session\_roles(s)$	$session\_roles'(s) = session\_roles(s) \backslash \{r\}$
$checkAccess(s : SESSIONS, op : OPS, ob : OBS)$	$\exists r \in ROLES : r \in session\_roles(s) \wedge$ $((op, type(ob)), r) \in PR$	

session, an app can activate any subset of the roles attached to it that is suitable for the session's task to be accomplished. The function $session\_roles$ is used by the system to identify all roles currently active for a particular session. The effective permissions available to a session will then be the permissions assigned to all the effective roles activated by the app for that session.

The function $avail\_session\_perms$ returns the union of all permissions assigned to sesssion's active roles. It should be noted that object access requests from an app's session is authorized based on the type of the requested object. The *type* function is used for this purpose.

B. System Functions Specifications

System functions in SDN-RBAC define features for the creation and deletion of app sessions, role activation and deactivation in a session, and for calculation of an access decision. The specifications of system functions are shown in Table II, and discussed below.

- **createSession**: The system function $CreateSession$ creates a new session $s \in SESSIONS$ for a given app $a \in APPS$ as owner and an active role set $ars \in ROLES$ to be used during the session lifetime. The function is valid if and only if the active role set is a subset of the roles assigned to that app. The system updates the set $SESSIONS$ by adding s to it. This also updates $app\_sessions$ and the $session\_roles$ mappings.
- **deleteSession**: The function $deleteSession$ deletes a given session $s \in SESSIONS$ for a given app $a \in APPS$. The function is valid if and only if the session is owned by the given app. The system updates the set $SESSIONS$ by removing s. The mapping $app\_sessions$ is also updated by this removal.
- **addActiveRole** and **dropActiveRole**: For repairing the network security policy at runtime, adding or doping of session's active roles might be also required. The activation and deactivation of a roles during a session is done by the system functions $addActiveRole$ and $dropActiveRole$, respectively. Adding an active role is valid if and only if the role is assigned to the app, and the session is owned by that app. Drop an active role is valid if and only if the session is owned by the app and the role is an active role of that session.
- **checkAccess**: The function $checkAccess$ returns whether an app's session is or is not allowed to perform a given operation on a given object. The session has the privilege to perform the operation on that object if and only if a permission that combines that operation to the object type is assigned to (at least) one of the session's active roles set.

IV. SESSION HANDLING APPROACHES

In a multi-session SDN app, app sessions can have an independent existence and run sequentially or simultaneously without reference to each other. An atomic session instance is the one which has a self-contained task definition and is not dependent on other session instances (i.e., a session that is not described in terms of other sessions and has no interaction with other session instances). See Fig. 2 (a).

In other cases, it is possible to have inter-session dependency and the execution of one session affects another one. Inter-session dependency initiates the need for inter-session interaction at runtime as will as functions and conditions for session creation/deletion, nomination of session's active role set, and adding/dropping an active role to a session. Fig. 2 shows several cases for multi-session apps and various methods for inter-session interaction. The relations between different sessions from different apps is beyond the scope of this discussion.

An executing session instance may, conditionally, initiate the creation of one or several session instances. In other cases, a session is created when another session completes. Given a system with a complete view of an app's entire functionality, all possible sessions, the task that should be achieved by each session, and inter-session dependencies among them, then a complete view of session-to-session relations can be represented using a directed graph. For example, if session creation happens conditionally based on another session, a directed edge between these two sessions, starting from the initiating instance, may indicate the condition and the active role set required for session creation as indicated in Fig. 2(c) and (e), respectively.

The management of inter-session dependency and inter-session interaction can be done either by the developer (developer-driven approach), the app system (system-driven approach), or the sessions themselves (session-driven approach). We believe that the interaction among app sessions should be well defined and managed. In this section we discuss different approaches for handling session instances of an SDN controller app.

A. Developer-driven Session Handling

This approach requires that the developer has full and prior knowledge of all possible sessions and roles required for each one to achieve its task. This information is provided to the controller before app execution and the system is configured accordingly. i.e., the controller knows before app execution what session instances will be created, the tasks that will execute in each session, and the set of roles required for it to execute correctly.

For each session instance, the *developer* should specify at *design time* different session handling aspects: First, the task that will be achieved by each session. Second, the condition (or precondition) under which a particular session may be created/deleted (e.g., after exceeding a bandwidth consumption cap, after new device detected, at system start-up, etc.). It should be noted here that the developer knows in advance the condition/criteria of a particular session creation as it is fixed and hard coded in the application and cannot be configured at runtime by the administrator or the controller. Example, create *Data Cap Enforcing* session if a host exceeded a bandwidth consumption cap. So, this session will start only after this condition is met. Creating this session cannot happen under any other circumstances. Third, the active role set that should be activated during session creation (e.g., $Packet\text{-}in\,Handler$ and $Flow\,Mod$ roles for a one-hour *deep packet inspection* session that will temporarily inspect traffic payload incoming from black-listed hosts). Finally, adding or dropping an active role for a session (e.g., add $DeviceTracking$ role to the *transmission rate monitoring* session).

B. System-driven Session Handling

In this approach, the controller has full control on session handling. The developer only provides the set of roles required by the app and then she has no discretion on determining any of other sessions' properties at runtime. Shipped with adequate capabilities, the *controller* should have the ability to specify at *runtime* what session instances will be created and how to handle them. Given an app and the set of roles required by the app, the controller should figure out each task that might execute in a separate session and the set of roles required for it to execute correctly. This approach is challenging and the hardest to implement.

For each session instance, the *controller* should specify *dynamically* at *runtime* the various properties: First, the set of sessions required to achieve the entire app's functionality. Second, the condition under which a particular session instance may be created/deleted (e.g., after attack detected, completion of another session, change of risk value, etc.). It should be noted here that, in contrast to the developer-driven approach, the developer doesn't know why a particular session could be created/terminated. The criteria for creating a session could be computed dynamically by the controller or configured by the administrator at runtime. For example, creating *intrusion prevention* session based on the outcome of statistical analysis or risk assessment. Third, the active role set that should be activated during session creation (e.g., $Routing$ and $Link\,Handler$ roles for a session that recomputes shortest path after a new link discovery), and Finally, adding or dropping an active role for a session.

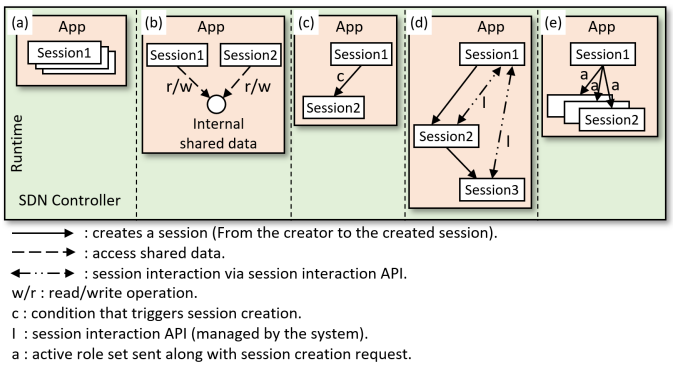

Fig. 2. Multi-session apps and methods for inter-session interaction. (a) App with atomic sessions. (b) Two sessions access shared data. (c) Conditional session creation. (d) Interaction via inter-session interaction APIs. (e) Active role set sent from master session to slave sessions.

C. Smart Sessions

For deploying this category of session handling, inter-session interaction should be conducted via a well defined set of session interaction APIs designed specifically for this purpose and managed by the system. The app developer should comply to these APIs during app design. These APIs allow sessions to get information about other sessions like names of currently active sessions, their active roles, their status (e.g., idle, up time, etc.) as will as they provide a way for passing information and notifications between sessions (e.g., results of calculations) as indicated in Fig. 2 (d).

A session is smart in the sense that it can take decisions based on the result of communications via inter-session interaction APIs. Thus, it can adjust its behavior to take knowledgeable decisions on future session interaction API calls and on different session handling aspects: First, the condition under which a particular session instance may be created/deleted (e.g., start *traffic redirection* session after an alarm is fired by *packet inspection* session). Second, the active role set that should be activated during session creation (e.g., $Packet\text{-}in\,Handler$ role and $Flow\,Mod$ role for a *deep packet inspection* session if *web-traffic filtering* session detected malicious payloads.), Third, adding or dropping an active role for a session (e.g., add $Device\,Tracking$ role to the *transmission rate monitoring* session).

TABLE III
ROLES ASSIGNED TO $DataUsageCapMngr$ APP AND OTHER SELECTED ROLES FROM SDN-RBAC.

Role	General Functionality
Device Handler	permissions for querying the controller about devices
Bandwidth Monitoring	permissions to read the bandwidth consumption for switch ports.
Flow Mod	permissions to insert/update/delete flow rules into a switch's flow tables.
Link Handler	permissions to get information about network links
Device Tracking	permissions to get notifications about changes on network devices (added, removed, Moved, Address Changed, etc.)
Port Handler	permissions to read information about ports and their status
Routing	permissions to get and compute routes between various source and destination nodes

D. Master-Slave Sessions

In this approach, a master session initiates the creation of one or several slave sessions and provides the system with the required roles to be activated for each one, as indicated in fig. 2(e). Slave sessions help the master session in achieving a subtask. So this approach has two restrictions: First, the active role set of any slave session should be a subset of the master session's active role set. Second, the master session cannot terminate during the life of a slave session. During its execution, master session passes control to slave sessions and waits until completion. When completed, each slave session passes results and control back to the master session. This approach can be considered a special case of smart-sessions approach as it can use inter-session interaction APIs. App developer should be aware of what sessions should be master and which ones should be slave and design the app to apply this dominance via these APIs.

V. USE CASE SCENARIO: A MULTI-SESSION APP

In this section we describe a use case scenario in which an SDN app has two tasks that run in two separate sessions. Table IV shows the configuration of the use case in SDN-RBAC. The app's main functionality is to limit the amount of traffic that any particular host transfers within a period of time. In order to achieve this, the app requires access to bandwidth consumption statistics of all hosts' attachment points. When a device exceeds the data usage cap, the app inserts a flow rule to rate-limit or temporarily quarantine a host who has exceeded the cap. We called the app $DataUsageCapMngr$.

To be able to get the required permissions, we associate the app with three roles described in the first three rows in Table III. These three roles are composed totally of eleven permissions. So, for space limitation and convenience, we avoid showing all permissions in the use case configuration in table IV. We show only selected permissions enough to understand the app's use case and its model configuration aspects.

Instead of executing the app in one monolithic process with all three roles active at once, we separate its functionality into two main tasks each to be running in a different session instance. We moved the sensitive task of inserting flow rules into a separate session.

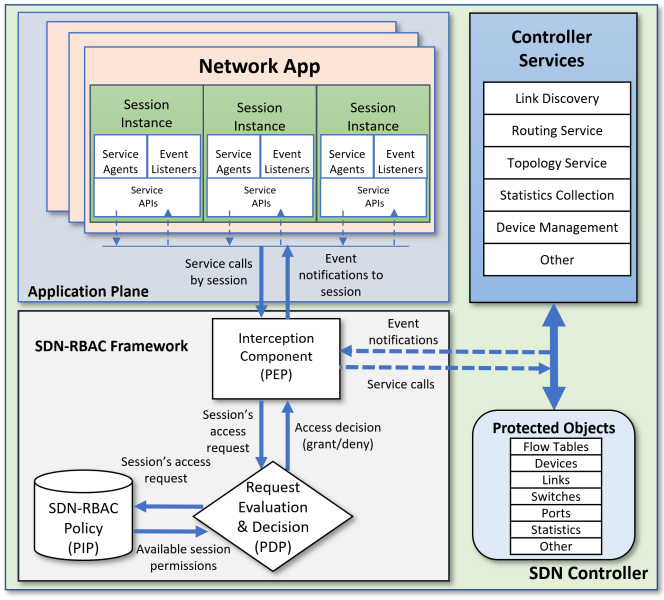

Fig. 3. Overview of SDN-RBAC architecture.

We called one session $DataUsageAnalysisSession$ which probes for statistics on a regular basis (every 5 seconds). This session reads the bandwidth consumption for switch ports, analyzes the data and stores the results into an object 'usageCapBlackList' managed by the system. This session is created with an active role set composed of two roles as shown in $session\_roles$ function in Table IV. The other session is called $DataCapEnforcingSession$ which requires inserting flow rules and so its active role set is composed of the $FlowMod$ role as shown in $session\_roles$ function in Table IV.

VI. FRAMEWORK IMPLEMENTATION

In order to demonstrate our proof-of-concept prototype, we developed and ran the framework in Floodlight platform v1.2 release [16]. 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 controller and hosts are virtual machines that has 2GB and run Ubuntu 14.04 OS server.

We implemented our RBAC system in Floodlight platform and used hooking techniques without any change to the code of Floodlight modules. We implemented hooking for all operations exposed by Floodlight services to controller apps. We used AspectJ [17] which is a seamless aspect-oriented extension to Java. Our system intercepts methods before execution. When a session issues a request the hooked API invokes the RBAC policy engine for performing access

TABLE IV
THE CONFIGURATION OF THE $DataUsageCapMngr$ AND ITS TWO SESSIONS AS A USE CASE IN SDN-RBAC[1].

- **Use case sets:**
- $APPS = \{DataUsageCapMngr\}$.
- $ROLES = \{Device\,Handler, Bandwidth\,Monitoring, Flow\,Mod\}$.
  D = set of all network devices. FT = set of all flow tables in all switches, PS = set of all port statistics in all switches.
- $OBS = \{D, FT, PS\}$.
- $OBTS = \{DEVICE, PORT\text{-}STATS, FLOW\text{-}TABLE\}$.
- $OT = \{(D, DEVICE), (PS, PORT\text{-}STATS), (FT, FLOW\text{-}TABLE)\}$.
- **Permissions:**
- $PRMS = \{p_1, p_2, p_3\}^1$ with
  $p_1 = (getAllDevices, DEVICE), p_2 = (getBandwidthConsumption, PORT\text{-}STATS), p_3 = (InsertRule, FLOW\text{-}TABLE)\}$.
- **Permissions assignment:**
- $PR = \{(p_1, Device\,Handler), (p_2, Bandwidth\,Monitoring), (p_3, Flow\,Mod)\}$.
- $assigned\_perms(Device\,Handler) = \{p_1\}^1, assigned\_perms(Bandwidth\,Monitoring) = \{p_2\}^1, assigned\_perms(Flow\,Mod) = \{p_3\}^1$
- **Role assignment:**
- $AR = \{(DataUsageCapMngr, Device\,Handler)$
  $(DataUsageCapMngr, Bandwidth\,Monitoring), (DataUsageCapMngr, Flow\,Mod)\}$.
- **Sessions:**
- $SESSIONS = \{DataUsageAnalysisSession, DataCapEnforcingSession\}$.
- $app\_sessions(DataUsageCapMngr) = \{DataUsageAnalysisSession, DataCapEnforcingSession\}$.
- $session\_app(DataUsageAnalysisSession) = \{DataUsageCapMngr\}$,
  $session\_app(DataCapEnforcingSession) = \{DataUsageCapMngr\}$.
- **Active role sets:**
- $session\_roles(DataUsageAnalysisSession) = \{Device\,Handler, Bandwidth\,Monitoring\}$.
- $session\_roles(DataCapEnforcingSession) = \{Flow\,Mod\}$.

[1]Sets with this mark in the table include minimum elements enough to understand the use case. Remaining elements are avoided for more convenience and readability.

verification and reply back. This system can be deployed to all other Java-based SDN controllers.

An overview of SDN-RBAC framework architecture is shown in Fig. 3. It contains three main components: interception component which represents the system's policy enforcement point (PEP), request evaluation component which represents the policy decision point (PDP), and SDN-RBAC policy which represents the policy information point (PIP) in the system.

We developed the $DataUsageCapMngr$ app described in Section V and configured in the SDN-RBAC model as shown in Table IV. The first session $DataUsageAnalysisSession$ is designed to probe for port bandwidth statistics on a regular basis (every 5 seconds). After reading the bandwidth consumption for switch ports and analyzing the data to find cap limit violations, it stores the list of hosts who has exceeded the cap limit into a list 'usageCapBlackList' managed by the system. The second session $DataCapEnforcingSession$ is designed to check periodically (every 60 seconds) for black listed hosts in order to insert flow rules to isolate them from the network. It reads the object 'usageCapBlackList' for this manner.

It should be noted that there is no direct interaction between these two sessions. They are running simultaneously and not directly dependent on each other. i.e., one won't crash/stop/start based on the state of the other. Also, the active role set is not provided by either one to the other and neither of them adds/drops an active role for the other. The tasks and roles associated with each session is determined at design time. This deployment is compliant with the 'Developer-driven' session handling approach described in Section IV-A and uses

```
roller.statistics.IStatisticsService.getBandwidthConsumption, PORT-STATS)
The method net.floodlightcontroller.topology.ITopologyService.getAllLinks
is called by session net.floodlightcontroller.datausagemngr.DataUsageAnalysisSession
16:36:31.982 WARN [n.f.rbac.RBAC:Thread-12] SDN-RBAC: security violation, "Access denied".
Unauthorized access requested by session [DataUsageAnalysisSession]
Reason: None of session active roles contains a corrseponding permission
Active roles set for this session: [Device Handler, Bandwidth Monitoring]
16:36:32.630 INFO [n.f.l.i.LinkDiscoveryManager:Scheduled-3] Sending LLDP packets out of a
```

Fig. 4. Snapshot of authorization check result for *getAllLinks()* operation requested by $DataUsageAnalysisSession$ - Access Denied.

the inter-session interaction method indicated in Fig. 2 (b).

During the lifetime of the app, our access control system keeps mediating all sessions access requests for performing security authorizations. It can identify each session, mediate each access request and send it for authorization check based on SDN-RBAC configuration. To show that our system can identify and reject any unauthorized operations submitted at the session level, we forced $DataUsageAnalysisSession$ to practice the permission $(getAllLinks, LINK)$ which is assigned to the role $Link\,Handler$. This role is not a member of the active role set of $DataUsageAnalysisSession$. Thus, $(getAllLinks, LINK)$ it is not a member of the available session permissions. A snapshot of the execution result is shown in Fig. 4. It shows how our system can identify and reject this unauthorized access from this session. On the other hand, Fig. 5 shows how $DataUsageAnalysisSession$ was able to pass the authorization check when $getBandwidthConsumption$ operation was called.

VII. PERFORMANCE EVALUATION

For evaluating the performance of our proposed system, we selected fifty operations covered by nineteen different roles

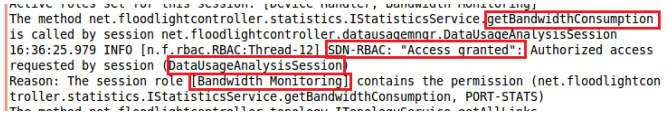

Fig. 5. Snapshot of authorization check result for *getBandwidthConsumption()* operation requested by $DataUsageAnalysisSession$ - Access Granted.

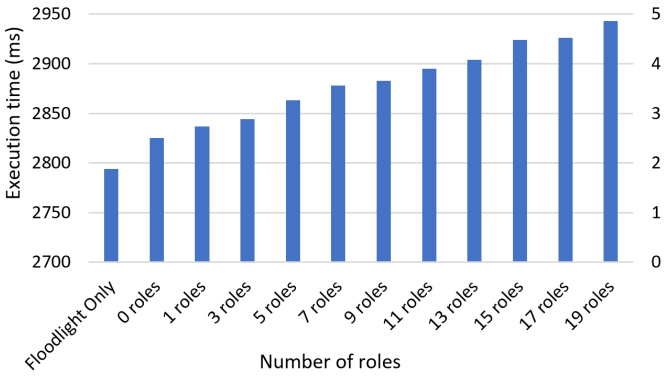

Fig. 6. Average execution time required to finish the tested operations, including and excluding SDN-RBAC.

and we incrementally assigned these roles to one test app. Despite the fact that this app doesn't require all these roles, the purpose of this test is (1) to check the difference in execution time between Floodlight with and without SDN-RBAC and (2) to check the effect of the increased number of roles on Floodlight performance.

In each test we executed the same fifty operations thousand times and measured the average execution time with different number of roles. For each test we changed the number of roles assigned to the app. It should be noted that the more roles assigned to the test app, the more operations passed the authorization check, the more time required to finish execution.

We started the timer at the beginning of the controller's *main()* function and it was ended after executing a loop with thousand iterations with each iteration calling the set of fifty operations. We repeated each test for 10 times and the average elapsed time was reported for each test as shown in Fig. 6. Each execution time, except for the first one, includes boot-up time, the time for loading the SDN-RBAC policy and creating the corresponding relations.

In the first test, we called all operations with Floodlight only with SDN-RBAC framework inactive. Then we ran the controller with our SDN-RBAC system activated and our test app assigned different number of roles as shown in Fig. 6. Although, it is required that any SDN app to be assigned at least one role to operate in the controller, we ran the application with zero roles for testing only. In this test all app requests are rejected early. With the increasing number of roles, more operations was executed and thus longer execution time was required. This test shows that SDN-RBAC framework adds a slight overhead on the performance of Floodlight controller.

VIII. Conclusion and Future Work

In this paper, we proposed *SDN-RBAC*, a role-based access control model for SDN controller apps to help in applying least of privilege principle at the level of apps and their sessions. We identified different approaches in which the system can handle app sessions. We also, demonstrated our framework by implementing a prototype and demonstrating its usability in a popular SDN controller. As a future work, we plan to work on extending *SDN-RBAC* to include other security aspects for SDN apps like static and dynamic separation of duty and conflict resolution between apps' operations.

Acknowledgment

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

References

[1] N. Bizanis and F. A. Kuipers, "Sdn and virtualization solutions for the internet of things: A survey," *IEEE Access*, vol. 4, pp. 5591–5606, 2016.
[2] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, 2013.
[3] K. Z. Bijon, R. Krishnan, and R. Sandhu, "Risk-aware rbac sessions," in *International Conference on Information Systems Security*. Springer, 2012, pp. 59–74.
[4] 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.
[5] 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.
[6] S. Scott-Hayward *et al.*, "Operationcheckpoint: Sdn application control," in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*. IEEE, 2014, pp. 618–623.
[7] J. Noh *et al.*, "Vulnerabilities of network os and mitigation with state-based permission system," *Security and Communication Networks*, vol. 9, no. 13, pp. 1971–1982, 2016.
[8] 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.
[9] P. A. Porras *et al.*, "Securing the software defined network control layer." in *NDSS*, 2015.
[10] 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.
[11] X. Wen, *et al.*, "Sdnshield: Reconciliating configurable application permissions for sdn app markets," in *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2016, pp. 121–132.
[12] H. Padekar *et al.*, "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, 2016, pp. 51–61.
[13] 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.
[14] C. Yoon *et al.*, "A security-mode for carrier-grade sdn controllers," in *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 2017, pp. 461–473.
[15] B. Ujcich *et al.*, "Cross-app poisoning in software-defined networking," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 648–663.
[16] Floodlight-Project. (2019) http://www.projectfloodlight.org/.
[17] AspectJ. (2019) Aspectj: A seamless aspect oriented extension to java. https://www.eclipse.org/aspectj/.