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

Abdullah Al-Alaj\(^1\), Ravi Sandhu\(^1\) and Ram Krishnan\(^2\)

\(^1\)Dept. of Computer Science  
\(^2\)Dept. of Electrical and Computer Engineering  
\(^1,2\)Institute for Cyber Security  
\(^1,2\)Center for Security and Privacy Enhanced Cloud Computing (C-SPECC)  
University of Texas at San Antonio, TX 78249

The Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA 2020)  
December 2, 2020
Agenda

- Motivation
- Access Control Administration in SDN
- SDN-RBACa Administrative Model
- Custom and Proxy Operations
- Custom Permissions
- Task and Role Engineering Custom Permissions
- Use-Case and Administrative Actions
- Evaluation and Comparison
- Conclusion and Future Work
Motivation

• RBAC has been applied in SDN.
• RBAC simplifies the administration of authorizations.
• Currently, role-based approaches for SDN are lacking such administrative model.

• Operations provided by SDN services are coarse grained.
• Extend the capabilities of SDN services and provide fine grained custom permissions.
Small SDN networks could be managed by a single administrator or a single admin unit.

Larger SDNs become more complex to centrally manage all access control associations by a single administrative authority.

Administration has to be decentralized into multiple AUs.

Administrative Units in SDN
Access Control Administration in SDN

- App-role and permission-role relations need management.
- In SDN-RBACa administrative model.
  - Indirect permission-role assignment.
  - **Permissions** are grouped into permission-pools (tasks).
  - **Tasks**: units of network functions.
  - **Apps** are grouped into app-pools.
  - **Administrative Units** for administering app-role and task-role relations.
SDN-RBACa Administrative Model (Conceptual)

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 U {<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 \ {<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 U {<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 \ {<a, r>}`.

8. **Administrative User Authorization Functions:**
   - `can_manage_task_role(u: USERS, t: TASKS, r : ROLES) = ∃au∈AU : (u, au) ∈ TA_admin ∧ r ∈ roles(au) ∧ t ∈ tasks(au))`.
   - `can_manage_app_role(u: USERS, a : APPS, r : ROLES) = ∃au∈AU : ((u, au) ∈ AA_admin ∧ r ∈ roles(au)) ∧ ∃ap∈AP : ((a, ap) ∈ AAPA ∧ ap ∈ app_pools(au))`.

6. **Administrative Units and Partitioned Assignment:**
   - `roles(au : AU) → 2^ROLES`, assignment of roles, where
     - `r ∈ roles(au_1) ∩ r ∈ roles(au_2) ⇒ au_1 = au_2`.
   - `tasks(au : AU) → 2^TASKS`, assignment of tasks, where
     - `t ∈ tasks(au_1) ∩ t ∈ tasks(au_2) ⇒ au_1 = au_2`.
   - `app_pools(au : AU) → 2^APP`, assignment of app-pool, where
     - `ap ∈ app_pools(au_1) ∩ ap ∈ app_pools(au_2) ⇒ au_1 = au_2`.

7. **Administrative User Assignment:**
   - `TA_admin ⊆ USERS × AU`.
   - `AA_admin ⊆ USERS × AU`.

---

@ Abdullah Al-Alaj
SDN-RBACa Administrative Model (Conceptual)

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 ∪ {(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 \ {(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 ∪ {(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 \ {(a, r)}`.

8. Administrative User Authorization Functions:
   - `can_manage_task_role(u: USERS, t: TASKS, r: ROLES) = ∃ au ∈ AU : (u, au) ∈ TA_admin ∧ r ∈ roles(au) ∧ t ∈ tasks(au)`.
   - `can_manage_app_role(u: USERS, a: APPS, r: ROLES) = ∃ au ∈ AU : ((u, au) ∈ AA_admin ∧ r ∈ roles(au)) ∧ ∃ ap ∈ AP : ((a, ap) ∈ AAPA ∧ ap ∈ app_pools(au))`.

6. Administrative Units and Partitioned Assignment:
   - `roles(au : AU) → 2^{ROLES}`, assignment of roles, where `r ∈ roles(au1) ∧ r ∈ roles(au2) ⇒ au1 = au2`.
   - `tasks(au : AU) → 2^{TASKS}`, assignment of tasks, where `t ∈ tasks(au1) ∧ t ∈ tasks(au2) ⇒ au1 = au2`.
   - `app_pools(au : AU) → 2^{AP}`, assignment of app-pool, where `ap ∈ app_pools(au1) ∧ ap ∈ app_pools(au2) ⇒ au1 = au2`.

7. Administrative User Assignment:
   - `TA_admin ⊆ USERS × AU`.
   - `AA_admin ⊆ USERS × AU`.

@ Abdullah Al-Alaj
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 $(o, t_1) \in OT \land (o, t_2) \in OT \Rightarrow t_1 = t_2$.

3. Derived Functions (operational):
- type: (o: OBS) $\rightarrow$ OBTS, a function specifying the type of an object.
- Defined as type(o) = \{i $\in$ OBTS | (o, i) $\in$ 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 \land (p, t) \in PA\}.

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

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$_1$) $\land$ $r \in$ roles(au$_2$) $\Rightarrow$ au$_1$ = au$_2$.
- tasks(au : AU) $\rightarrow$ $2^{TASKS}$, assignment of tasks, where
  $t \in$ tasks(au$_1$) $\land$ $t \in$ 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$_1$) $\land$ 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$_{admin}$ $\land$ $r \in$ roles(au) $\land$ $t \in$ tasks(au).
- can_manage_app_role(u : USERS, a : APPS, r : ROLES) =
  $\exists$ au $\in$ AU : ((u, au) $\in$ AA$_{admin}$ $\land$ $r \in$ roles(au)) $\land$
  $\exists$ ap $\in$ AP : ((a, ap) $\in$ AAPA $\land$ 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)\}.
In large SDNs, specialized **apps** control/analyze and monitor/inspect specific network **traffic** type.

These apps should be authorized to access only traffic type they handle and not other type (via roles).

---

**Use Case using SDN-RBAC**

- **Motivation**

  - Web Load Balancers
  - Web Firewalls
  - etc.

  - VoIP Load Balancers
  - VoIP Firewalls
  - etc.

  - FTP Load Balancers
  - FTP Firewalls
  - etc.

  - Email Load Balancers
  - Email Firewalls
  - etc.

---

**Apps**

- **Web-specific apps:**
  - Web Load Balancers
  - Web Firewalls
  - etc.

- **VoIP-specific apps:**
  - VoIP Load Balancers
  - VoIP Firewalls
  - etc.

- **FTP-specific apps:**
  - FTP Load Balancers
  - FTP Firewalls
  - etc.

- **Email-specific apps:**
  - Email Load Balancers
  - Email Firewalls
  - etc.

---

**Roles**

- **Web-specific roles:**
  - Web Flow Mod
  - Web Load Balancing
  - etc.

- **VoIP-specific roles:**
  - VoIP Flow Mod
  - VoIP Load Balancing
  - etc.

- **FTP-specific roles:**
  - Ftp Flow Mod
  - Ftp Load Balancing
  - etc.

- **Email-specific roles:**
  - Email Flow Mod
  - Email Load Balancing
  - etc.
In large SDNs, specialized apps control/analyze and monitor/inspect specific network traffic type.

These apps should be authorized to access only traffic type they handle and not other type (via roles).

Use Case using SDN-RBACa - Motivation

Apps

- **Web-specific apps:**
  - Web Load Balancers
  - Web Firewalls
  - etc.

- **VoIP-specific apps:**
  - VoIP Load Balancers
  - VoIP Firewalls
  - etc.

- **FTP-specific apps:**
  - FTP Load Balancers
  - FTP Firewalls
  - etc.

- **Email-specific apps:**
  - Email Load Balancers
  - Email Firewalls
  - etc.

Roles

- **Existing, but problematic**

  - Flow Mod
  - Load Balancing
  - etc.
Custom and Proxy Operations

3. Create proxy operations

OP Proxy1

OP Proxy2

OP Proxy3

Proxy group

1. Clone operation

2. Refine (verify access to appropriate content)

3. Create proxy operations and call custom operation by passing actual value.

addWebFlow(...){
    call addFlow(..., web)
}

addVoIPFlow(...){
    call addFlow(..., voip)
}

addFtpFlow(...){
    call addFlow(..., ftp)
}

AddFlow

provides restrictive access to specific traffic type.

(ensure that flow rule handles correct traffic type)
Custom Permissions

- Custom permissions are those permissions that are created using proxy operations.

\[
\begin{align*}
(\text{OP}_{\text{Proxy}_1}, \text{ot}) \\
(\text{OP}_{\text{Proxy}_2}, \text{ot}) \\
(\text{OP}_{\text{Proxy}_3}, \text{ot}) \\
\ldots
\end{align*}
\]

A permission created using target operation.

(\text{addFlow}, \text{FLOW-RULE})

Custom permissions created using proxy operations:

\[
\begin{align*}
(\text{addWebFlow}, \text{FLOW-RULE}) \\
(\text{addVoIPFlow}, \text{FLOW-RULE}) \\
(\text{addFtpFlow}, \text{FLOW-RULE}) \\
(\text{createWebMember}, \text{LB-POOL-MEMBER}) \\
(\text{createVoIPMember}, \text{LB-POOL-MEMBER}) \\
(\text{createFtpMember}, \text{LB-POOL-MEMBER}) \\
(\text{readWebPacketInPayload}, \text{PI-PAYLOAD}) \\
(\text{readVoIPPacketInPayload}, \text{PI-PAYLOAD}) \\
\ldots
\end{align*}
\]
Task and Role Engineering using Custom Permissions - Example

SDN Apps
- Web Intrusion Prevention
- VoIP Load Balancer
- FTP Application Firewall

Roles
- Web Flow Mod
- VoIP Flow Mod
- FTP Flow Mod

Tasks
- Web Traffic Forwarding Task
- VoIP Traffic Forwarding Task
- FTP Traffic Forwarding Task
- Web Flow Viewing Task
- VoIP Flow Viewing
- FTP Flow Viewing Task

Custom Permissions
- (addWebFlow, FLOW-RULE)
- (addVoipFlow, FLOW-RULE)
- (addFtpFlow, FLOW-RULE)
- (deleteWebFlow, FLOW-RULE)
- (deleteVoipFlow, FLOW-RULE)
- (deleteFtpFlow, FLOW-RULE)
- (readWebFlow, FLOW-RULE)
- (readVoipFlow, FLOW-RULE)
- (readFtpFlow, FLOW-RULE)

OPProxy
- addWebFlow
- addVoipFlow
- addFtpFlow
- deleteWebFlow
- deleteVoipFlow
- deleteFtpFlow
- readWebFlow
- readVoipFlow
- readFtpFlow

OPCustom
- addFlow(traffic)
- deleteFlow(traffic)
- readFlow(traffic)

OPTarget
- clone
- addFlow
- deleteFlow
- readFlow
In large SDNs, specialized apps control/analyze and monitor/inspect specific network traffic type. These apps should be authorized to access only traffic type they handle and not other type (via roles).

**Apps**
- **Web-specific apps:** Web Load Balancers, Web Firewalls, etc.
- **VoIP-specific apps:** VoIP Load Balancers, VoIP Firewalls, etc.
- **FTP-specific apps:** FTP Load Balancers, FTP Firewalls, etc.
- **Email-specific apps:** Email Load Balancers, Email Firewalls, etc.

**Roles Required, and available**
- **Web-specific roles:** Web Flow Mod, Web Load Balancing, etc.
- **VoIP-specific roles:** VoIP Flow Mod, VoIP Load Balancing, etc.
- **FTP-specific roles:** Ftp Flow Mod, Ftp Load Balancing, etc.
- **Email-specific roles:** Email Flow Mod, Email Load Balancing, etc.
Functional Administrative Units for SDN

- Relations between apps and roles should be managed by different administrative units.

**Administrative Units**

- **Web Admin Unit**
  - **Roles**: {Web Flow Mod, Web Load Balancing, etc.}
  - **App-Pools**: {Web Security, Web Load Balance, etc.}

- **Email Admin Unit**
  - **Roles**: {Email Flow Mod, Email Load Balancing, etc.}
  - **App-Pools**: {Email Security, Email Load Balance}

- **VoIP Admin Unit**
  - **Roles**: {Email Flow Mod, VoIP Load Balancing, etc.}
  - **App-Pools**: {VoIP Security, VoIP Load Balance}

- **FTP Admin Unit**
  - **Roles**: {FTP Mod Email, FTP Load Balancing, etc.}
  - **App-Pools**: {FTP Security, FTP Load Balance}
Use-Case and Administrative Actions

Tasks, roles, and app-pools in white are exclusively managed by: **Web Admin Unit**
Tasks, roles, and app-pools in gray are exclusively managed by: **VoIP Admin Unit**

Administrative User Assignment:
\[ \text{TA\_admin} = \{(\text{web\_functions\_admin\_user}, \text{Web Admin Unit}), (\text{voip\_functions\_admin\_user}, \text{VoIP Admin Unit})\} \]

Example:

1. **Administrative Action to assign task to a role:**
   \[ \text{assign\_task\_to\_role(web\_functions\_admin\_user, \text{Web Traffic Forwarding Task}, \text{Web Flow Mod}) is allowed.} \]
   
   **⇒ Authorization Function:**
   \[ \text{can\_manage\_task\_role(web\_functions\_admin\_user, \text{Web Traffic Forwarding Task}, \text{Web Flow Mod}) = True.} \]
   
   **Reason:**
   \[ \exists \text{Web Admin Unit} \in \text{AU} : (web\_functions\_admin\_user, \text{Web Admin Unit}) \in \text{TA\_admin} \land \text{Web Flow Mod} \in \text{roles(Web Admin Unit)} \land \text{Web Traffic Forwarding Task} \in \text{tasks(Web Admin Unit)}. \]
Evaluation and Comparison

- Evaluation of SDN-RBACa operational model with tasks and proxy permissions.
- Test app with 50 proxy operations ops covered by 10 different roles.
- Report authorization time for all 50 requests.
- Different security policies.
- Test repeated 100 times for each security policy.
- Average authorization time is calculated.
- Overhead of around 2.9% on average to the authorization framework.
Conclusion and Future Work

• This work presented SDN-RBACa, an administrative model for role based access control in SDN.

• An approach to extend the capabilities of SDN services for creating custom SDN permissions specialized for the administration of access control in SDN.

• Through proof of concept prototype implementation and use cases, we demonstrated the usability of custom permissions.

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