ParaSDN: An Access Control Model for SDN Applications based on Parameterized Roles and Permissions

Abdullah Al-Alaj\textsuperscript{1}, Ram Krishnan\textsuperscript{2} and Ravi Sandhu\textsuperscript{1}

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

The 6th IEEE International Conference on Collaboration and Internet Computing (CIC 2020)
December 1 - 3, 2020
• Introduction to Software Defined Networking (SDN)
• Parameterized Roles and Permissions in SDN
• ParaSDN main components
• ParaSDN Parameter Engine
• ParaSDN Conceptual Model and Definitions
• Use-Case Security Configuration in SDN-RBAC
• ParaSDN Implementation & Evaluation
• Conclusion and Future Work
Introduction
Traditional Networks

Management Layer

Infrastructure Layer

Traditional Network

SDN Idea

Decoupling
Introduction
Software Defined Networks (SDN)

Applications
- Routing
- Firewall
- Load Balancing
- Intrusion Prevention
- Network Visualization
- Other

Network Programming APIs

Controller (e.g., Floodlight)

Infrastructure

OpenFlow Protocol

Flow table

Virtual Network Resources
- Topology
- Flow tables
- Switches
- Ports
- Traffic payloads
- Configurations
- VLANs
- Devices
- Other

Network Services
- Topology Service
- Routing Service
- Statistics Collection
- Switch Management
- Entry Pushing
- Device Management
- Link Discovery
- Other

Manage

Decoupling

Control plane

Data plane

APIs
- Network Services
- Virtual Network Resources

@ Abdullah Al-Alaj
Motivation

- Apps are authorized on object types (e.g., (addFlow, FLOW RULE)) → Fine grained access control is required.

Role: Flow Mod\textsubscript{1}
Assigned Perms:
- (addFlow, flow_rule\textsuperscript{sw0x1})
- (addFlow, flow_rule\textsuperscript{sw0x2})
- (addFlow, flow_rule\textsuperscript{sw0x3})

Role: Flow Mod\textsubscript{2}
Assigned Perms:
- (addFlow, flow_rule\textsuperscript{sw0x4})
- (addFlow, flow_rule\textsuperscript{sw0x5})
- (addFlow, flow_rule\textsuperscript{sw0x6})

Role: Flow Mod\textsubscript{3}
Assigned Perms:
- (addFlow, flow_rule\textsuperscript{sw0x7})
- (addFlow, flow_rule\textsuperscript{sw0x8})
- (addFlow, flow_rule\textsuperscript{sw0x9})

- Multiple very closely related roles are defined to achieve fine-grained access control.
- Roles are limited in membership.
Introducing Parameterized Roles and Permissions in SDN

Role: Flow Mod
Assigned Perms: (addFlow, FLOW RULE) (deleteFlow, FLOW RULE) (updateFlow, FLOW RULE) (readFlow, FLOW RULE)

dept = CS
depth = CIS
depth = CE

Controller (Floodlight)

Requires restriction
Proposed work

- Formal Access control model for SDN enhanced with role and permission parameters
- Authorization Framework extended with parameter engine and enforcement in SDN controller.

Fine-Grained and Scalable Access Control for SDN
Parameterized Permissions and Roles

- **Parameters**
  - name:value pairs.
  - Add restrictions on access to network resources.

- **Parameterized Roles:**
  \[(r_i , \{(par_1 , val_1 ), (par_2 , val_2 ), \ldots\})\]

  **Example:**
  \[
  (\text{Flow Mod}, \{(\text{dept}, \perp), (\text{traffic}, \perp)\})
  \]

- **Parameterized Permissions:**
  \[
  ((\text{op}_i, \text{ot}_i), \{(par_1 , val_1 ), (par_2 , val_2 ), \ldots\})
  \]

  **Example:**
  \[
  ((\text{addFlow}, \text{FLOW-RULE}), \{(\text{dept}, \perp), (\text{traffic}, \perp)\})
  \]

\[\perp = \text{Unknown}.\]
ParaSDN Conceptual Model
ParaSDN Formal Model Definition

1. Basic Sets:
   - APPS, ROLES, OPS, OBS, OBTS, PAR, and VAL: set of apps, roles, operations, objects, object types, parameters, and parameter values.
   - For each par ∈ PAR, Range(par) represents the parameter’s range, a finite set of atomic values. We assume VAL includes a special value “⊥” to indicate that the value of a parameter is unknown.
   - parType: PAR → {set, atomic} specifies parameter type as set of atomic valued.
   - PRMS ⊆ OPS × OBTS, set of ordinary permissions.
   - SESSIONS, set of sessions.

2. Assignment Relations:
   - OT ⊆ OBS × OBTS, a many-to-one relation mapping an object to its type, where
     \((o, ot_1) ∈ OT \land (o, ot_2) ∈ OT \Rightarrow ot_1 = ot_2\).
   - PVPAIRS ⊆ PAR × VAL, a many-to-many mapping parameter to value assignment relation.
     For convenience, for every pvpair = (par, val), pvpair ∈ PVPAIRS, let pvpair.par = par and pvpair.val = val.
   - PPRMS ⊆ PRMS × 2^{PVPAIRS}, a relation mapping a permission role to subset of (parameters, value) combinations.
     For convenience, for every pp = ((op, ot), PVPAIRS), pp ∈ PPRMS, let pp.op = op, pp.ot = ot, and pp.PVPAIRS = PVPAIRS.
   - PROLES ⊆ ROLES × PVPAIRS, a relation mapping a role to subset of combinations of parameters and their values.
     For convenience, for every pr = (r, PVPAIRS), pr ∈ PROLES, let pr.r = r, and pr.PVPAIRS = PVPAIRS.
   - PPA ⊆ PRMS × PROLES, a many-to-many mapping parameterized permission to parameterized role assignment relation.
   - AA ⊆ APPS × PROLES, a many-to-many mapping app to parameterized role assignment relation.

3. Derived Functions:
   - assigned_pperms: PROLES → 2^{PRMS}, the mapping of parameterized role into a set of parameterized permissions.
     Formally, assigned_pperms(pr) = \{pp ∈ PRMS — (pp, pr) ∈ PPA\}.
   - app_sessions: APPS → 2^{SESSIONS}, the mapping of an app into a set of sessions.
   - session_app: SESSIONS → 2^{APPS}, the mapping of a session into the corresponding app.
   - session_roles: SESSIONS → 2^{ROLES}, the mapping of session into a set of parameterized roles.
     Formally, session_roles(s) = \{pr ∈ PROLES — (session_app(s), pr) ∈ AA\}.
   - type: OBS → OBTS, a function specifying the type of an object defined as
     \[\text{type}(o) = \{t ∈ OBTS — (o, t) ∈ OT\}\] .
   - avail_session_pperms: SESSIONS → 2^{PRMS}, the parameterized permissions available to an app in a session.
     Formally, avail_session_pperms(s) = \bigcup_{pr ∈ session_roles(s)} assigned_pperms(pr).

4. Parameter Verification Functions:
   - VERIFIERS = \{V_1, V_2, ... , V_n\} a finite set of Boolean functions.
     For each \(V_i \in \text{VERIFIERS}, V_i : \text{SESSIONS} × \text{OPS} × \text{OBS} × \text{PVPAIRS} → \{\text{True, False}\}\).
   - param_verifier: OBTS × PAR → VERIFIERS, a function that maps a combination of object type and parameter to the corresponding verification function needs to be evaluated.
Parameter Value Assignment

- Parameter values assigned via assignApp administrative action propagate automatically from role parameters to permission parameters.

```
<table>
<thead>
<tr>
<th>Function</th>
<th>Authorization Condition</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignPPPerm(pp, pr)</td>
<td>$pp \in \text{PPRMS } \land \text{ pr } \in \text{PROLES } \land (pp, pr) \notin \text{PPA}$</td>
<td>$\text{PPA'} = \text{PPA} \cup {(pp, pr)}$</td>
</tr>
</tbody>
</table>
| assignApp(a, pr, valset) | $a \in \text{APPS } \land \text{ pr } \in \text{PROLES } \land$ \text{valset } \in \text{VAL } \land (a, pr) \notin \text{AA}$ | //Assign values to role parameters.  
For each $\text{pr}_{\text{pvpair}_i} \in \text{pr.PVPAIRS}, v_i \in \text{valset}, 1 \leq i \leq |\text{pr.PVPAIRS}|$ do  
$\text{pr}_{\text{pvpair}_i},\text{val} = v_i$  
//Pass parameter values from pr to its member parameterized permissions.  
For each $pp \in \text{PPRMS} : (pp, pr) \in \text{PPA}$ do  
For each $\text{pr}_{\text{pvpair}_i} \in \text{pr.PVPAIRS}, pp_{\text{pvpair}_i} \in \text{pp.PVPAIRS}, 1 \leq i \leq |\text{pr.PVPAIRS}|$ do  
$pp_{\text{pvpair}_i},\text{val} = \text{pr}_{\text{pvpair}_i},\text{val}$  
$\text{AA'} = \text{AA} \cup \{(a, pr)\}$ |
```

@ Abdullah Al-Alaj
ParaSDN Framework Architecture

Application Plane

Network App

Controller Services

Protected Objects

ParaSDN Policy (PIP)

Request Evaluation & Decision (PDP)

Access control layer

Interception Component (PEP)

Authorized request

Request, Parameters

Available parameterized permissions

Session’s access request

Session’s access request (grant/deny)

Access decision (True/False)

Param Check Point (PCP)

Verifiers Retrieval Point (VRP)

Verifiers Map

ob. type, param

Param Verification Points (PVP)

V_i

T/F

V_j

T/F

V_n

T/F

V_i()
• The General functionality of the Parameter Engine is distributed among multiple components:
  • Parameter Check Point (PCP),
  • Verifiers Retrieval Point (VRP), and
  • multiple Parameter Verification Points (PVPs).
ParaSDN Implementation

Timer Started

Timer Ended

Access control layer

Interception Component (PEP)

Request Evaluation & Decision (PDP)

Session's access request

Available parameterized permissions

Request, Parameters

Access decision (grant/deny)

Session's access request

ParaSDN Policy (PIP)

Authorized request

Param Check Point (PCP)

Param Checkers Retrieval Point (VRP)

ob. type, Param. set

verifiers list

ob. type, param_i

V_i

V_i

V_n

Parameter Verification Points (PVP)

Protected Objects

Topology
Flow Tables
Statistics
Links
Devices
Other

Controller Services

Topology Service
Routing Service
Statistics Collection
Link Discovery
Device Management
Other

Session
Service Agents
Event Listeners
Service APIs

Session
Service Agents
Event Listeners
Service APIs

Application Plane

Network App

Controller

Request by session

Result to session

Result

Read/Write
Use-Case & Security Configuration in ParaSDN Model

Data Usage Cap Mngr

Intrusion Prevention App

Controller

depth = CS

vlan_id = 1

Host1

Host2

web server CS

0x1

0x3

web server CE

Host5

Host6


depth = CE

vlan_id = 2

0x2

Host3

Host4

Host3

Host4
1. Model Basic Sets:
- APPS = \{Data Usage Cap Mgr, Intrusion Prevention App\}
- ROLES = \{Device Handler, Bandwidth Monitoring, Flow Mod, Packet-In Handler\}
- OPS = \{queryDevice, getBandwidthConsumption, addFlow, readPacketInPayload\}
- OBS = D \cup PS \cup FR \cup PIP, where D = set of all network devices, PS = set of all port statistics in all switches, FR = set of all flow rules, and PIP = set of all packet-in messages.
- OBTS = \{DEVICE, PORT-STATS, FLOW-RULE, PI-PAYLOAD\}
- PAR = \{vlan_id, attachment_point, dept, traffic\}
- Range(vlan_id) = \{1, 2\}, Range(attachment_point) = \{0x1:1, 0x1:2, 0x2:2, 0x2:3\}
- Range(dept) = \{CS, CE\}, Range(traffic) = \{web\}
- parType(vlan_id) = atomic, parType(attachment_point) = set, parType(dept) = set, parType(traffic) = atomic.
- PRMS = \{(queryDevice, DEVICE), (getBandwidthConsumption, PORT-STATS), (addFlow, FLOW-RULE), (readPacketInPayload, PI-PAYLOAD)\}
- SESSIONS = \{DataUsageAnalysisSession, DataCapEnforcingSession, IntrusionPreventionSession\}

2. Assignment Relations:
- OT = \{id, DEVICE\} : d ∈ D \bigcup \{ps, PORT-STATS : ps ∈ PS\} \bigcup \{fr, FLOW-RULE : fr ∈ FR\} \bigcup \{pip, PI-PAYLOAD : pip ∈ PIP\}
- PPRMS = \{(queryDevice, DEVICE), (vlan_id, 1)\}, \{(getBandwidthConsumption, PORT-STATS), (attachment_point, 1)\}, \{(addFlow, FLOW-RULE), (dept, 1), (traffic, 1)\}, \{(readPacketInPayload, PI-PAYLOAD), (attachment_point, 1)\}
- PROLES = \{Device Handler, (vlan_id, 1)\}, \{Bandwidth Monitoring, (attachment_point, 1)\}, \{Flow Mod, (dept, 1), (traffic, 1)\}, \{Packet-In Handler, (vlan_id, 1)\}
- PPA = \{(queryDevice, DEVICE), (vlan_id, 1)\}, \{Device Handler, (vlan_id, 1)\}, \{getBandwidthConsumption, PORT-STATS, (attachment_point, 1)\}, \{Bandwidth Monitoring, (attachment_point, 1)\}, \{addFlow, FLOW-RULE, (dept, 1), (traffic, 1)\}, \{Flow Mod, (dept, 1), (traffic, 1)\}, \{readPacketInPayload, PI-PAYLOAD, (attachment_point, 1)\}, \{Packet-In Handler, (vlan_id, 1)\}

3. Derived Functions:
- assigned_pperms((Device Handler, ([vlan_id, ⊥])) = \{\{(queryDevice, DEVICE), ([vlan_id, ⊥])\}\}
- assigned_pperms((Bandwidth Monitoring, ([attachment_point, ⊥])) = \{\{getBandwidthConsumption, PORT-STATS, ([attachment_point, ⊥])\}\}
- assigned_pperms((Flow Mod, ([dept, ⊥], (traffic, ⊥))) = \{\{addFlow, FLOW-RULE, ([dept, ⊥], (traffic, ⊥))\}\}
- assigned_pperms((Packet-In Handler, ([attachment_point, ⊥])) = \{\{(readPacketInPayload, PI-PAYLOAD), ([attachment_point, ⊥])\}\}

4. Parameter Verification Functions:
- VERIFIERS = \{DeviceVlan, VStatsAttachpoint, VRuleSwitch, VRuleTraffic, VPInAttachpoint\}
- parameter_verifier((DEVICE, vlan_id)) = DeviceVlan.
- parameter_verifier((PORT-STATS, attachment_point)) = VStatsAttachpoint.
- parameter_verifier((FLOW-RULE, dept)) = VRuleSwitch.
- parameter_verifier((FLOW-RULE, traffic)) = VRuleTraffic.
- parameter_verifier((PI-PAYLOAD, attachment_point)) = VPInAttachpoint.
ParaSDN Evaluation

- Test app with 50 ops covered by 10 different roles.
- Report authorization time for all 50 requests.
- Different security policies (parameters and roles).
- Test repeated 100 times for each security policy.
- Average authorization time is calculated.
- Floodlight’s boot-up time is ignored.
ParaSDN Evaluation

On average: ParaSDN adds 0.031 ms overhead compared to 0.025 for SDN-RBAC.
In this work:

• We proposed ParaSDN, a formal access control model that provides fine grained capabilities for SDN using the concept of parameterized roles and permissions.

• We implemented a proof of concept prototype in an SDN controller.

Future research

• Extend the model to suit the needs for multi-controller environments in SDN-Enabled technologies like IoT and Cloud infrastructures.