An Autonomic and Policy-based Authorization Framework for OpenFlow Networks

Daniel Rosendo
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
Email: daniel.rosendo@gprt.ufpe.br

Patricia Takako Endo
Universidade de Pernambuco
Caruaru, Brazil
Email: patricia.endo@upe.br

Djamel Sadok and Judith Kelner
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
Email: {jamel, jk}@gprt.ufpe.br

***Abstract*—The Network Access Control (NAC) management is a critical task, especially in current networks that are composed of many heterogeneous things (Internet of Things) connected to share data, resources and Internet access. The Software-Defined Networking (SDN) simplifies the network design and operation, and offers new opportunities (programmability, flexibility, dynamicity, and standardization) to manage the network. Despite this, the access control management remains a challenge, once managing security policies involves dealing with a large set of access control rules, detecting conflicting policies, defining priorities, delegating rights, and reacting against network state changes and events. This work presents the HACFlow, a novel, autonomic, and policy-based framework for access control management in OpenFlow networks. HACFlow aims to simplify and automate the network management allowing network operators to govern rights of network entities by defining dynamic, fine-grained, and high-level access control policies. We analyzed the performance of HACFlow and compared it against related approaches.**

***Index Terms*—Software-defined Networks; Internet of Things networks; Security management; Policy-based management; Autonomic and cognitive management.**

I. Introduction

During the last decade, advances in the Internet architecture and communication system technologies together with the introduction of the Software-Defined Networking (SDN) and Internet of Things (IoT) paradigms contributed to the growth of the network and the number of interconnected heterogeneous devices. In this scenario, there are several devices in the network, each one with different level of access rights among them. These devices exchange information and interact with each other and with humans and machines. Ensuring the privacy of these entities by managing access rights to protect them from unauthorized access become a challenge [1].

From the network operator perspective, security tasks, such as managing the access rights for each network entity is complex and challenging. The configuration complexity is one of the main reasons for security breaches in enterprise networks [2]. Besides, manual configurations (prone to errors) and the continuous network reconfiguration (due to the dynamic nature of the network) also increase the management complexity.

Due to those concerns, there is a need for a more sophisticated access control solution based on high-level and autonomic policy implementations to minimize costs, network administrator effort, and errors [3], [4].

In [2], authors highlight some research challenges in SDN. The first (*Synchronization of Network Security and Network Traffic*) refers to the need for a synchronization between the network security and traffic due to network state changes and events. In the second (*Network Security Automation*), authors point the need to automate the network security configuration to avoid manual configurations which are prone to errors.

In [5], authors discuss challenges and management requirements in SDN. They point out: *From High-level Rules to Network Configuration* and *Autonomic and In-Network Management*. The first regards the need to reconstruct the lost low-level information while translating high-level rules into low-level configurations. While the second one regards the autonomic reaction against network events.

The Open Networking Foundation (ONF) [6] points out requirements for the SDN architecture. One of them regards the use of *Network Interaction Policies*. This requirement highlights the need to create mechanisms to express, distribute, delegate, and manage interaction policies that define which operations can be performed by network entities.

According to those requirements, research challenges, and problems to be solved, we propose the HACFlow. HACFlow aims to simplify and automate the management of access control policies in OpenFlow networks by providing mechanisms to: *i)* express, distribute, delegate, and manage interaction policies; *ii)* define dynamic, fine-grained, and high-level access control policies; *iii)* translate high-level security policies into network congurations; and *iv)* maintain the synchronization between the security policies and the network trafc.

II. HACFlow Framework

1) ***OrBAC API****:* The Organization Based Access Control (OrBAC) API is a sub-component of HACFlow. Its main role is to allow the definition of high-level and context-aware security policies in a fine-granular way. Besides, it provides mechanisms to detect and solve (by defining priorities) conflicting policies. The OrBAC API is composed of policy implementer, policy checker, policy parser, and policy inference. The *policy implementer* allows the creation of predicates (*Organization, Role, Activity, View, and Context*), entities (*Subject, Action, and Object*), and abstract permission and prohibition policies. Then, the *policy checker* checks for constraints and conflicts in

978-3-901882-98-2

those abstract policies. Next, the *policy parser* generates the concretes rules from the abstract policies. Lastly, the *policy inference* infer the concrete rules considering its circumstances. Figure 1 depicts the abstract and concrete levels of OrBAC.

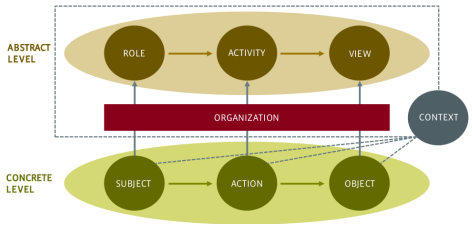

Figure 1: The OrBAC model. Adapted from: orbac.org.

2) ***Policy Skeleton****:* The Policy Skeleton generates a policy template file containing management configurations such as role delegation and revocation. Besides, it allows network operators to provide additional information about network entities. Those information allow HACFlow to derive the OpenFlow flow rules from the high-level security policies.

3) ***Entity Manager****:* The Entity Manager keeps a record of authenticated network entities. Its role is to maintain the synchronization of this record with the network. This record contains additional information obtained from the authentication. It includes the entity *identifier*, *IP/MAC address*, *connected switch*, and *switch port*. Such information is used by the policy translator to construct the OpenFlow flow rule.

4) ***Event Listener****:* The Event Listener autonomously processes network events and policy's context changes. Such events may be the result of a user authentication, a vulnerability alert detected by a security service, among others. The main role of the event listener is to maintain the synchronization between the high-level security policies and the network configurations. It allows the network operator to describe how to react in case malicious traffic is detected (see Section III).

5) ***Policy Translator****:* The autonomic policy translation allows network operators to define high-level goals without taking care of how they will be implemented in the network. As HACFlow allows network operators to define policies in a high-level way, such higher level of abstraction results in the loss of low-level network information (e.g. IP/MAC address, port number, connected switch, among others). Therefore, these low-level data must be reconstructed in the translation process. This way, while translating a high-level security policy into a low-level OpenFlow flow rule, the components of the HACFlow architecture (OrBAC API, Policy Translator, and Entity Manager) must work together to perform the translation. Figure 2 depicts the whole translation process. Once a user authenticates, HACFlow (step one) gets the user's high-level policies through the OrBAC and (step two) pass them to the Policy Translator. Next, the Policy Translator (step three) gets from the Entity Manager the low-level data and then translates the high-level rule (OrBAC) into the low-level rule (OpenFlow). Lastly, HACFlow (step four) returns the OpenFlow rules.

6) ***REST API****:* The Representative State Transfer (REST) API is the point of interaction with SDN applications. Through

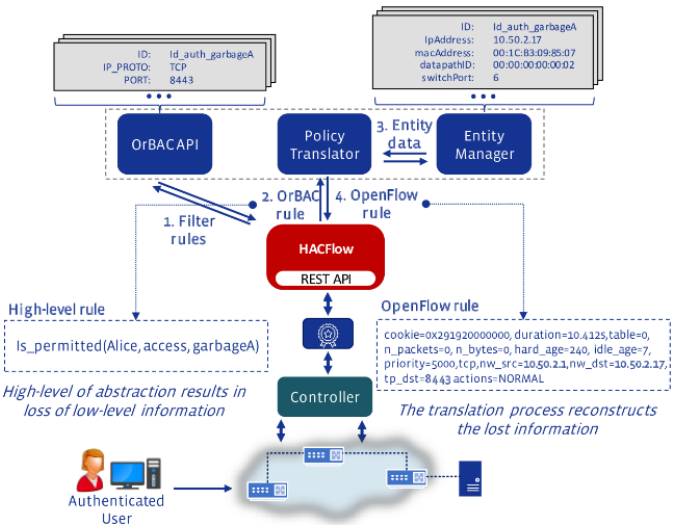

Figure 2: OrBAC to OpenFlow policy translation in HACFlow.

this API, the HACFlow framework receives network events and responds to them, maintaining the security policies synchronized with the network configurations. Besides, it provides methods to load policy file data, register authenticated entities, get entity's security policies, delegate roles, among others.

III. HACFlow in a Step-by-Step Way

A. Step-by-step: High-level Policy Definition

Next, we describe how a network operator can express their high-level goals into high-level security policies. Those policies may be permission or prohibition that say who can access what and in which circumstances.

Required steps:

i) Create the *Organization*, *Role*, *Activity*, *View*, and *Context* predicates.
ii) Create the *Subject*, *Action*, and *Object* entities.
iii) Assign entities of the previous step to a *Class definition*.
iv) Link these predicates and entities using the *Permission and Prohibition* relationships to compose and obtain the high-level policy.
v) Manually solve conflicting policies detected by OrBAC.
vi) Load the security policy file data into HACFlow.

Completed these six steps, HACFlow is able to provide the security policies to be enforced in the network. Figure 3 summarizes the previous six steps to define a high-level policy in HACFlow (steps A[i) to v)] and B[vi)]).

B. Step-by-step: Dynamic Security Policies

Suppose that a network operator wants to control the access to a network resource (webmail, printer, smart bulb, among others) imposing some circumstances (day of a week, an hour of a day, etc.). The operator decided to allow the access during 8AM to 18PM from Monday to Friday. To implement this circumstance the operator must:

i) Create the *Context* predicate.
ii) Create the context definition (determines circumstance).

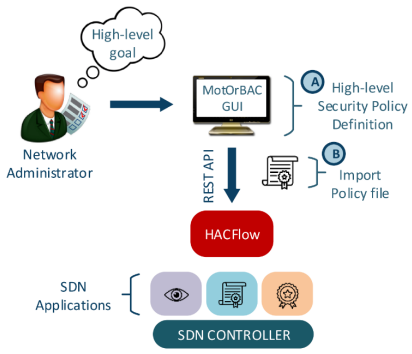

Figure 3: High-level policy definition in HACFlow.

Once a security policy linked to this circumstance became out of context (19pm) HACFlow autonomously reconfigure the network to comply with the operator's needs. Required steps:

i) The OrBAC API notifies the affected security rules.
ii) The Policy Translator translates the security rule into OpenFlow flow rules.
iii) HACFlow sends the OpenFlow flow rules to be enforced and change the network configuration.

After HACFlow reacts against a dynamic permission policy, the network entities linked to this rule will be able (if the policy became active) or not (if the policy became inactive) to access a network resource.

C. Step-by-step: Reacting against Network Events

Next, we describe how a network operator can configure HACFlow to interpret a vulnerability alert sent by a security monitoring system (Distributed Denial of Service (DDoS) or Deep Packet Inspection (DPI)) on the network. As an example, suppose that a security monitoring system detects an attack and sends an alert notifying that the network is vulnerable. In such case, the operator wants to immediately block any access to the storage service to protect the overall enterprise information. To make this configuration the operator must:

i) Create the *Object* entity (representing the network event) and assign it to a *Class definition*.
ii) Create the *Context* predicate.
iii) Create the context definition (determines circumstance).

Supposing that in step one the *Object* entity *"networkSecurityState"* was created and assigned to a class with the attribute *"state"* and the value *"safe"* as the default network state. Once HACFlow is notified about the vulnerability alert, it automatically changes the *object's* attribute value from *"safe"* to *"vulnerable"*. As a result, the policy becomes out of context and HACFlow reconfigures the network to block the access to the storage service to protect the enterprise data. Therefore, HACFlow will execute the following tasks:

i) Through the OrBAC API, change the entity attribute (created in step one above) according to the alert received.
ii) The OrBAC API notifies the affected security rules.
iii) The Policy Translator translates the security rules.
iv) Send the flow rules to change the network configuration.

D. Step-by-step: Role Delegation

HACFlow allows network operators to delegate and revoke security rules among network entities. Therefore, the network operator must execute the following single step:

i) Define in HACFlow the network entity and the role being granted or revoked.

Next, HACFlow automatically implements the role delegation or revocation executing the following steps:

i) Assigns/unassigns the network entity to a role.
ii) Notifies the delegated/revoked security rules.
iii) The Policy Translator translates the security rules into OpenFlow flow rules.
iv) HACFlow sends the OpenFlow flow rules to be enforced and change the network configuration.

Once the delegation has been implemented, the network entity is able to access the network resources granted to it.

IV. Evaluation and Comparison

A. HACFlow Performance Evaluation

1) ***Network State Change - Vulnerability Alert****:* Next, we present the required time to HACFlow block the access of a user to a server after a vulnerability alert is detected. When a network alert is triggered by a security monitoring system the SDN application notifies HACFlow to reconfigure the network. Then, HACFlow processes the four steps described in Subsection III-C. From the results (see Figure 4a) we point that HACFlow needed 8.24 ms in average (32.1% of the total time) to react to a vulnerability alert, resulting in a fast reaction if compared to a manual reconfiguration (human-intervention).

2) ***Dynamic Security Policy****:* Dynamic security policies are context-aware policies that may have their state changed (active or inactive) depending on some circumstances (day of a week, an hour of a day, and so on). HACFlow is be able to automatically react to these contextual changes, not requiring any manual per-device reconfiguration of the network devices. In this experiment, we simulate a security policy being out of context, that means, being out of a circumstance imposed by the network operator. According to results (see Figure 4b), we point out that HACFlow required 7.57 ms in average (30.9% of the total time) to react (execute the three steps described in Subsection III-B) to a single dynamic security policy.

3) ***Role Delegation****:* Once a delegation occurs, HACFlow assigns the new security rules to the granted network entity and sends the rules to be enforced in the network. According to the results (see Figure 4c) the role delegation is the management task in HACFlow that requires a longer time (time to execute the four steps described in Subsection III-D). Despite this, requiring 119.45 ms in average (30.8% of the total time) to delegate a single role linked to a single security policy is still a fast time if you consider the task complexity and if you compare to a manual role delegation.

4) ***High-level to Low-level Policy Inference****:* We analyzed the scalability of HACFlow to infer the OpenFlow flow rules. In this analysis HACFlow infers 1, 4, 16, 32, 64, 128, and 256 rules.We divided the flow rule inference process into two

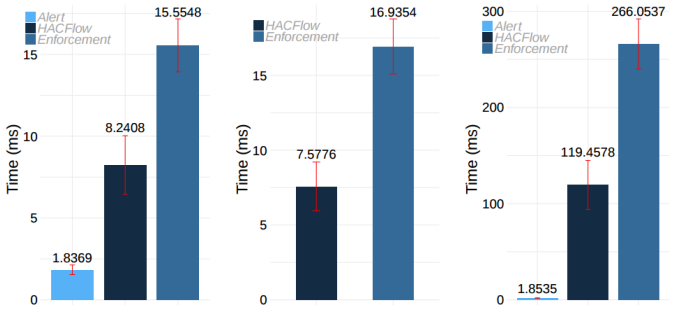

Figure 4: **(a)** Reaction against a vulnerability alert. **(b)** Reaction against a dynamic policy. **(c)** Role delegation.

steps. The first (*1. Security rule filter*) refers to the filtering of rules through the OrBAC API and the extraction of low-level of the network entities. The second one (*2. Policy translation*) refers to the process that obtains the OpenFlow flow rules from the OrBAC rules. From results (Figure 5), we highlight that HACFlow needed 0.4791ms to translate 256 rules. Furthermore, the whole process required 1.1 seconds in average.

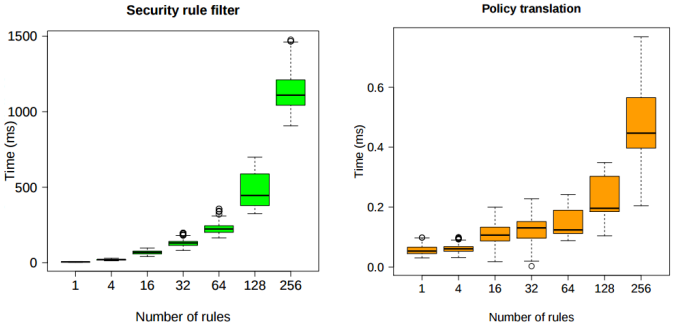

Figure 5: Policy inference steps: **(a)** security rule filter and **(b)** OrBAC to OpenFlow policy translation.

B. Comparison Against Existing Solutions

1) ***Framework Features****:* Next, we compare the management tasks provided by HACFlow, Frenetic [7], FRESCO [8], and OpenSec [9] based on the requirements and problems to be solved motivated in this work. Table I shows the results.

- *F1*) *High-level security policy definition*: allows operators to define policies in a high-level way. Operators do not worry about how these security policies will be implemented.
- *F2*) *Hierarchical policies*: permits operators to organize the network entities hierarchically to simplify management.
- *F3*) *Conflict detection and resolution*: provides mechanisms to detect and solve conflicts.
- *F4*) *Definition of dynamic security policies*: allows operators to define policies according to some circumstances.
- *F5*) *Reaction against network changes and events*: provides mechanisms to automatically react to network changes and events according to operator's needs.
- *F6*) *Security policy delegation*: permits operators to delegate rights between network entities.

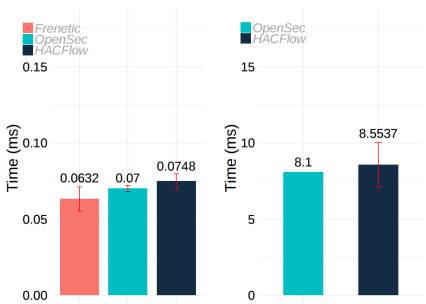

Figure 6: **a)** Rule translation and **b)** Event reaction comparison.

- *F7*) *High-level to low-level policy translation*: provide mechanisms to translate high-level policies into OpenFlow.

Table I: Features Implemented by Different SDN Solutions.

Features	Frenetic	FRESCO	OpenSec	HACFlow
F1)	✔	✔	✔	✔
F2)			✔	✔
F3)	✔	✔	✔	✔
F4)		✔		✔
F5)	✔	✔	✔	✔
F6)				✔
F7)	✔	✔	✔	✔

We highlight that HACFlow is the only one that allows operators to delegate roles to network entities. Besides, HACFlow and OpenSec allow the definition of hierarchical policies.

2) ***Policy Translation****:* We compare HACFlow against Frenetic and OpenSec frameworks regarding the required time to translate a high-level security policy into low-level OpenFlow flow rule. Differently of OpenSec, we can get access to the Frenetic source code project. We run Frenetic in a virtual machine with the same configuration. The experiment was executed 256 times and the results are the mean and standard deviation. From Figure 6a, we point that HACFlow, Frenetic, and OpenSec require similar times to translate a single security rule. But, Frenetic required a lower time.

3) ***Event Reaction Delay****:* We compare HACFlow against OpenSec regarding how long each one requires to react to a network state change and event. The reaction time includes the moment that the framework receives the alert until it returns the OpenFlow flow rules to reconfigure the network. The experiment was run 256 times and our results represent the mean and standard deviation. From Figure 6b, OpenSec required a lower time, 8.1ms against 8.5ms by HACFlow.

V. Conclusions

This paper presented HACFlow, a novel SDN framework that aims to simplify and automate the security policy management in OpenFlow networks.We analyzed the performance of HACFlow and the results showed that it significantly reduce the effort (HACFlow is faster and less prone to errors than manual per-device configurations) to implement a variety of network configurations. We also compared HACFlow against related approaches and the results showed that HACFlow offers more management features (based on the points motivated).

REFERENCES

[1] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in internet of things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.

[2] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[3] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[4] ONF, "Software-defined networking: The new norm for networks, https://www.opennetworking.org," *White Paper*, 2014.

[5] J. A. Wickboldt, W. P. De Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, "Software-defined networking: management requirements and challenges," *IEEE Communications Magazine*, vol. 53, no. 1, pp. 278–285, 2015.

[6] O. TR-516, "Framework for SDN: Scope and Requirements," `https://www.opennetworking.org`, 2015, version 1.0. Last access: December, 2016.

[7] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *ACM Sigplan Notices*, vol. 46, no. 9. ACM, 2011, pp. 279–291.

[8] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks." in *NDSS*, 2013.

[9] A. Lara and B. Ramamurthy, "Opensec: Policy-based security using software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 30–42, 2016.