A Network Access Control Solution Combining OrBAC and SDN

Rafael Aschoff
Pernambuco Federal Institute of Education, Science, and Technology (IFPE) - Pernambuco, Brazil
rafael.roque@palmares.ifpe.edu.br

Daniel Rosendo, Marcos Machado
Alexandre Santos, and Djamel Sadok
Universidade Federal de Pernambuco (UFPE) - Recife, Brazil
Email: {daniel.rosendo, marcos.machado, alexandre.santos, jamel}@gprt.ufpe.br

***Abstract*—Standard Port-based Network Access Control (NAS) with tagged Virtual Local Area Networks (VLANs) systems are useful to authenticate users within an isolated network environment. This approach on its own, however, lacks the flexibility and granularity level that new generation networks based on SDN (Software Defined Networking) can provide. The flow-based access control provides a more appropriate granularity to enforce network policies. In this paper, we propose a novel solution named SDN-based Network Access Control (S-NAC) that provides authentication and authorization of clients and servers based on high-level policies enforced at flow level. The solution has been implemented, deployed and tested over emulated and real networks.**

I. Introduction and Motivation

The security of technology systems information is a key aspect in any organization, especially for its critical infrastructure. Over the past few years, we have observed a staggering increase in the number of cyber-attacks, their sophistication, and damages caused. Access control strategies are able to deny access from unauthorized clients to the system and narrow what legitimated users may effectively access in a given environment, reducing the range of security breaches [16]. A strong NAC process is expected to reduce or even to avoid losses inherent from intentional and accidental security breaches.

Access control encompasses three sub-processes, namely, authentication, authorization, and accounting, frequently referred to by the initials 'AAA'. Networking systems provide AAA services usually by implementing the IEEE 802.1X standard, the most widely used framework for port-based Network Access Control (PNAS). Together with tagged Virtual Local Area Networks (VLANs), the standard provides a good foundation to support AAA for devices wishing to attach to a computer network. On the other hand, the standard PNAS mechanism, and more broadly speaking, the traditional model of computing infrastructures, is strongly based on hardware components. Until recently, this model was presented as sufficient to meet the demands of the market or computer applications in general. However, the drawbacks of models strongly dependent on hardware specifications have started to show to be counterproductive since they present a limiting factor for new business opportunities[10].

In this context, the need for the revitalization of computing infrastructures arises, eliminating the direct dependence on specific devices and enabling the flexibility of computing resources [5]. Software Defined Environments (SDE) allow solutions that were not practical or possible a few years ago. The SDN paradigm and its advantages over traditional networks were already exploited by some NAC solutions. However, they are often in their development initial stage or focus only on specific areas, such as high-level network policy programming languages, SDN applications for security policy enforcement, or authentication processes [11].

It is plausible to imagine a network with many servers, clients, and services, where some clients can access the email service, whereas a set of other clients cannot establish a connection to any host in a sub-network. Tasks like expressing the aforementioned scenario through network policies, managing security rules, as well as handling potential conflicting between them, become frequent and should be taken into account. Thus, it is necessary to create mechanisms to permit network managers to define, manage, and enforce those interaction policies in a way that easily says which operations can be performed by each network entity and decide which entity is permitted or prohibited to talk to each other [1].

As part of our proposed solution, we have used the OrBAC (Organization Based Access Control) model [8], a consolidated and widely used framework that can be applied in many scenarios due to its abstract characteristic. This paper presents a practical and complete approach named S-NAC (SDN-based Network Access Control), a solution that provides authentication and authorization of clients and servers based on high-level policies enforced at flow level. The solution has been implemented, deployed and tested over emulated and real networks.

The rest of this paper is organized as follows. Section II presents a general description of the solution proposed in this paper. Such general description is further developed in Sections III, IV , and V with the processes of the solution, components and interactions between them. Section VI presents the evaluation studies performed on our solution. Section VII performs an analysis of the related work. Finally, Section VIII concludes the paper and presents the next steps.

II. S-NAC Overview

S-NAC is a proposal for fine-grained control of network resources based on organizational policies designed for software-

978-3-901882-89-0 @2017 IFIP

defined computer networks (SDCN). The S-NAC solution is composed of a set of applications that interact within the context of an SDCN to allow the establishment of network flows identified by the requesting hosts and required services, thus controlling access to network resources.

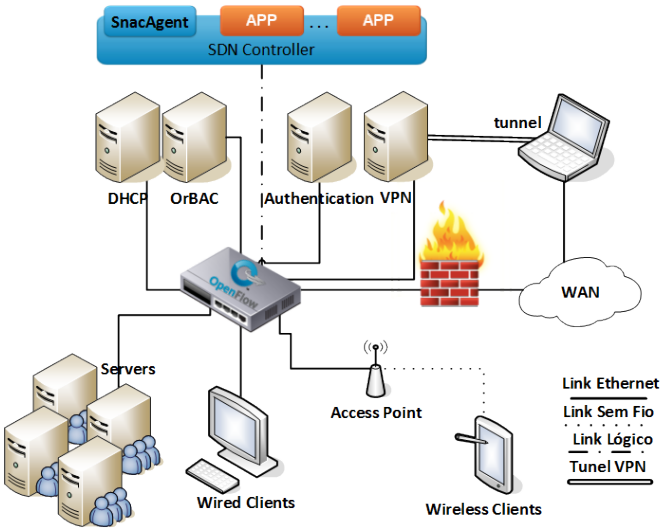

Fig. 1. S-NAC Overview

Figure 1 shows the general idea behind the access control used in the S-NAC. At the center of the solution, we have one or more OpenFlow switches which are managed by an SDN Controller. There are different classes of entities for which S-NAC may potentially provide control through OpenFlow switches, which act as Policy Enforcement Points (PEP). External hosts, local wired and wireless hosts, as well as servers, need to be authenticated to have access to the controlled network.

Authentication is performed using a server with functions similar to those from RADIUS. IEEE 802.1X packets exchanged between supplicants and the authentication server are monitored by SnacAgent running in the controller. In the case of external customers, authentication is done through a VPN server and do not follow the 802.1x process. In both cases, hosts receive a unique ID that is used in the authorization process.

OpenFlow switches have multiple tables where flow entries are installed. Our SnacAgent manipulates such table's flows through the SDN controller. In other words, the SnacAgent is able to add, update or remove entries in any flow table of switches managed by the SDN controller where our SnacAgent is deployed. The controlled switches use these set of flow rules to establish the actions to be performed upon receiving a packet.

Authorization is controlled in our solution by such entries and related actions (block or deny, for instance), which are translated from a set of abstract policies defined by the network administrator. These policies are specified using OrBAC model, which permits modeling user, roles, and components, such as servers or specific services.

Overall, our solution forces both client and servers to authenticate themselves through the SnacAgent and controls access to network resources by dynamically enforcing policies previously defined for these clients and servers.

The next sections detail the specific processes of authentication, authorization, and accounting provided by S-NAC.

III. Authentication

As previously mentioned, S-NAC provides authentication via a mechanism similar to the IEEE 802.1X standard. From the point of view of a peer (client or supplicant), it is exactly the same process. This simplifies the mechanics of deployment of our solution since most current operating systems already implement the required tools and mechanism to support PNAC as specified in the IEEE 802.1X standard.

The usual process, however, requires a number of functions performed by legacy switches, access points or routers. Our solution extracts such functions from the network devices and provides them using our SnacAgent in addition to a special authentication server.

The authentication server chosen for our prototype was the hostapd [14]. The hostapd does the heavy lifting for us in terms of the authentication process, however, we still require the interception of the exchanged packets (performed by our SnacAgent) in order to enforce the authentication (into OpenFlow Switches). Shortly after a successful authentication process, our SnacAgent allows DHCP traffic for the peer in question. Information regarding the peer is collected throughout the entire authentication process and subsequent DHCP negotiation, including MAC address, physically connected port, 802.1X identifier, and IP address. These values, together with the authentication status provided by the authentication server, give us the information we need to start the authorization process.

In order to reduce the potential damages caused by hackers, clients have to re-authenticate and can also be directly disconnected from the network. The re-authentication timer is configured in our SnacAgent. To deny access to a client regardless of the timer, an administrator would have to use our graphical interfaces to directly disconnect it and also update the set of policies (in case one wants to prevent or alter the privileges of such client).

As one may notice, impersonating the authentication server can jeopardize the authentication process. In order to mitigate such issue, we decided to use a prioritized list of authentication servers that must be directly registered on our SnacAgent.

Apart from monitoring the 802.1x packets, the SnacAgent is also responsible for informing the controlled switches where to forward these packets in order for them to reach the desired authentication server. Considering that we have control over where to forward 802.1x packets and from where flows are considered valid, compromising our solution by impersonating a registered authentication server becomes more difficult.

IV. Authorization

Abstract organizational policies provide a very friendly way to specify the expected behavior of today's complex computer

networks. In our solution, we employ the OrBAC framework due to its flexibility in specifying highly abstracted policies as well as its power to infer conflict free and concrete rules.

These concrete rules represent the inferred permissions and prohibitions of all specified clients, servers, services in a server, and more generic rules that contemplate networks. It is important to note, however, that while these are concrete rules, they are still not fully specified flow rules. For instance, these rules do not include network parameters such as switch port or client IP address.

In fact, it is not a trivial task to align organizational level policies with network-level policies. In any case, a translation mechanism is necessary. In S-NAC, such translation is performed by an OrBAC module inside the SnacAgent, alongside an external application named OrBAC Server.

The OrBAC Server keeps a database with the abstract policies stored as well as provides concrete rules by demand through a REST API. Upon request of the authentication module, the OrBAC module queries all concrete rules that affect the peer in question, translates them into network rules, that are OpenFlow table entries, and installs the entries throughout the required controlled switches accordingly.

1) Policy Definition: In the OrBAC model, the first step to create a security policy is to define an *Organization* entity. This organization represents the domain in which the abstract policies will be defined. The *Role*, *Activity*, and *View* abstract predicates also need to be defined for this organization.

The second step involves the specification of concrete entities of the network domain, including users, hosts, services, etc. After defining the concrete entities, a link between them and an organization is necessary. In order to do so, the relationships *Empower*, *Consider* and *Use* must be used to link *Subjects*, *Actions*, and *Objects* respectively.

Finally, the relationship *Define* specify *Contexts* which are situations or requirements that can be achieved or not, reflecting the actual state of an abstract rule and, consequently, its inferred concrete rules. The result of all these steps is a 5-nary predicate (*organization*, *role*, *activity*, *view* and *context*) of the OrBAC model. Abstract *permissions*, *prohibitions*, as well as *obligations* rules, can be specified following this guideline.

Organization: As we said, an *Organization* is the first entity that must be defined in order to create an abstract rule. We modeled the *Organization* as a network environment. The remained predicates of the 5-nary model must be created and assigned to this *Organization* (represented as our network topology) in order to create abstract security rules for the network environment.

Role and subject: A *Role* is a set of one or more *subjects*. Plenty of subjects could be grouped by a role, this allows reducing the number of security rules and consequently simplifying the policy management. The same is applied to the *Activity* and *View* predicates. Here, as an example, we defined the *AdminUser* and *StandardUser* as our roles and with them we empower the respective users of our network environment. The relationship *Empower* creates an association between subject, role and organization, as defined below.

Empower(Topology, Bob, StandardUser)

Activity and action: The *Action* entity represents actions between subjects and objects, that means, it represents which actions can be performed by a subject over an object. We defined the *oneWay* and *twoWay* actions of activity *Access*. Here, oneWay and twoWay access determine the flow of the communication, unidirectional or bidirectional respectively. The link between action, activity, and organization is created by the *Consider* relationship.

Consider(Topology, twoWay, Access)

View and object: Objects can be represented as inactive entities of the network, for example, network services. Here we represented objects as services such as WebMail, FTP, SSH, etc. The relationship presented bellow includes an example for the WebMail service.

Use(Topology, webMail, Services)

Context definition: In this work, we do not cover the definition of dynamic policies. Here, we use the default context of the OrBAC model, meaning that the state of all rules defined will not be changed and its state will always be active, resulting in static network security rules. The definition of dynamic policies is an ongoing work that we plan to cover in future works.

Class definition: A class definition can be defined with many attributes that may be assigned to *concrete* entities, allowing them to obtain valid network parameters and characteristics. We defined the following five classes (Client, Host, Network, Resource, and Any), as depicted in Figure 2. It is important to note that those attributes were created based on the list of required match fields of the OpenFlow protocol version 1.4.0 [15].

Fig. 2. Network attributes in class definition

Particular attention must be given to the ID attribute, presented in both Host and Client classes. As previously mentioned, our authentication module keeps records of the credential of the authenticated peers, such as their IDs. When a client or host authenticates on the network, it is through this ID attribute that the match occurs, allowing the enforcement of particular rules. Additional attributes are obtained as a result of the authentication process, such as IP address, switch, MAC address, and switch port.

2) Policy Repository: In our proposed solution, the S-NAC Agent get policies from an external OrBAC Application. This application uses an XML file to store only the configured network abstract policies. When this file is loaded by our application the OrBAC engine automatically infers the rules to be applied to the network.

3) Policy Decision Point: Here, S-NAC checks for abstract conflicting policies. The conflict of policies is a common problem, mainly when defining policies to high and complex network environments. This problem can be solved by setting priorities for each security policy. Therefore, we specify a priority scheme based on the aforementioned five classes (Resource, Client, Host, Any, and Network), where the first three have higher priorities and the others lower priorities. Figure 3 depicts this scheme.

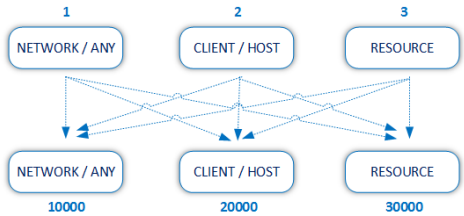

Fig. 3. S-NAC priority scheme

As a primary example of conflicting policies, suppose that the network policy manager defines the following abstract permission and prohibition:

Permission(Topology, StandardUser, twoWayAccess, HostA, default_context)

Prohibition(Topology, StandardUser, oneWayAccess, Network A, default_context)

Assuming that the subject Bob is empowered in StandardUser role, as well as, the HostA is inside NetworkA, the inferred concrete policies says that: "In Topology, Bob is permitted to access HostA", and "In Topology, Bob is prohibited to access "NetworkA", resulting in a conflict. Therefore, the enforcement of these two rules on a switch will result in two entries in a flow table, each one with its assigned flow priority, 40000 and 20000 respectively, based on our priority scheme. This way, incoming packets will first match flow entries with high priorities, in that case, the permission flow rule.

4) Policy Translation: Once we have specified our network environment in the OrBAC model and created our security policies, it is time to make the inferred OrBAC concrete rules understood by the OpenFlow switches in order to make those rules effective and usable. This means that the OrBAC concrete rules must be translated into OpenFlow flow rules, so that them can be write into the OpenFlow network switches in the topology. In Section VI we made experiments about the required time to execute this conversion. The following example show how an OrBAC rule will look like once translated to OpenFlow flow rule.

Abstract rule: *Permission(Topology, AdminUser, twoWayAccess, sshService, default_context)*

Concrete rule: *Is_permitted(Alice, twoWay, sshService)*

OpenFlow flow rule (ovs-ofctl): *cookie=0x2969600000000000, duration=11.582s, table=0, n_packets=0, n_bytes=0, hard_timeout=180, idle_age=11, priority=60000,tcp,nw_src =10.0.3.1,nw_dst=10.0.3.4,tp_dst=22 actions=NORMAL*

In this example, we are assuming that Alice is an AdminUser in a machine with Ip address 10.0.3.1, and the sshService is a service in a machine with Ip 10.0.3.4 on port 22. The resulting OpenFlow rule means that all packets with source Ip 10.0.3.1 and destination port 22 will be forwarded to machine 10.0.3.4.

5) Policy Enforcement Point: The security policy enforcement is the last step of our policy based network management process. It consists of deploying all the high level OrBAC security rules defined by the network manager.

In our implementation, we used different strategies to enforce the policies, each strategy depends on the OrBAC concrete rule entities (*Subject*, *Action*, and *Object*) and its associated class definition (Resource, Client, Any, Network, and Host). Figure 4 shows the enforcement of the following three security policy examples.

Policy 1: *Permission(Topology, Alice, twoWayAccess, HostA, default_context)*. In this example, we have a client-to-host permission rule. This means that, this flow will be enforced in a path from Alice's switch to HostA's switch, and due to the *twoWayAccess* activity each switch must have two flow rules, the input and output flows.

Policy 2: *Permission(Topology, Any, twoWayAccess, WebMail, default_context)*. Here, we have an any-to-resource permission rule. This means that, this security rule will be enforced in all switches in the topology and the *twoWayAccess* activity means that each switch must have the input and output flow.

Policy 3: *Prohibition(Topology, Student, oneWayAccess, Network_Professor, default_context)*. This is a client-to-network prohibition rule. In that case, only the switches in which a Student is connected to must have this single flow rule, once this is an *oneWayAccess* policy. We highlight that just one OpenFlow flow rule is enough to block the communication.

V. Accounting

Although not a traditional accounting process per si, S-NAC provides a way to monitor both past and current controlled traffic. Statistics are generated for every active flow entry installed in all controlled switches. Whenever a flow expires

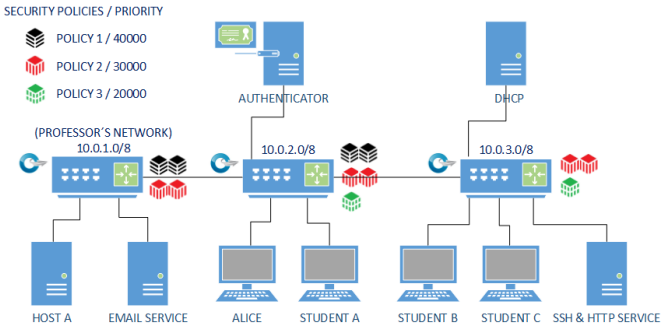

Fig. 4. S-NAC security policy enforcement.

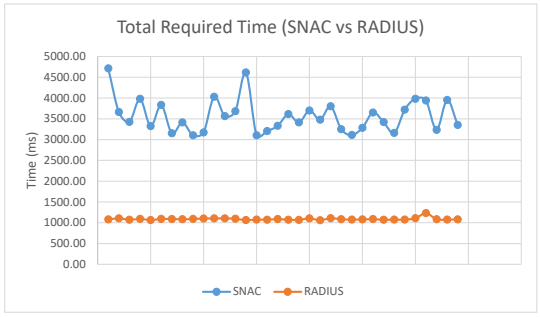

Fig. 5. Time to connect to the network and access a resource.

or is directly removed, its statistics are saved for future analysis. Statistics include creation time, expired time, number of matches, set of actions, switch id, priority, table id, number of packets, number of bytes, idle time out, hard time out, cookie and flow duration in second and nanoseconds.

Since the flow entries are directly related to the available policies and set of active clients, the provided statistics give a way to analyze the conformance or attempted deviation from the expected behavior of the controlled network. It is important to notice that the main focus of the S-NAC solution is the authentication and authorization steps. A full network accounting solution for SDN networks with the knowledge of highly abstract policies is still an open question and might pose as an opportunity for future enhancements of the S-NAC.

VI. Experiments and Results

To analyze S-NAC, we first compare the S-NAC with legacy networks by measuring how long would it take to access a simple resource on the network considering one is still not connected. Then, we evaluate the required time to convert a high-level policy defined in S-NAC to a low-level OpenFlow flow rule. In order to setup our SDN network we used a Lanner Network Device model FW-8760 with eight physical Ethernet ports running the Open vSwitch, a number of Desktop machines, and a set of U2 Servers. An additional legacy switch with support for RADIUS was also part of our testbed.

The first evaluation refers to the total required time to access a network resource for an unauthenticated client. In the S-NAC scenario, this time involves the following steps: (1) client authenticates into the network using WPA_Supplicant; (2) client negotiates an IP address with the DHCP server; (3) policies from the client are retrieved from the OrBAC Application; (4) the OrBAC module translates policies into flow rules; (5) flow rules are enforced in the network; finally, (6) the client uses the cURL command to request a web page and the process finishes when it receives an OK HTTP status. Steps (3) and (4) are not present for the legacy network and step (5) is much simpler, consisting only of the liberation of a physical switch port and VLAN assignment.

Figure 5 shows a time series comparison between our S-NAC solution with the legacy networks scenario. As one can observe, our solution authentication time is greater than the legacy network, which was expected since our manageable, flexible, dynamic, and fine-grained solution needs to translate high-level rules into OpenFlow network rules and enforce the generated rules into the switches. The total time for the SNAC scenario varied between 3.2 and 4.1 seconds and in about 90% of the cases such time were less than 3.9 seconds. It is important to note, however, that this is our worst case scenario and only happens once. The total time includes a series of processes such as authentication, IP allocation, authorization, and service request and response.

Our next experiment, show in Figure 6, refers to the translation of different high-level security policies based on the defined five classes (Client, Host, Network, Resource, and Any) to low-level OpenFlow flow rules. This experiment was conducted in a Mininet [12] emulated environment, with a client node, a Web server resource, both connected to an OpenFlow switch, and the HP VAN SDN[7] controller. We chose the Mininet network emulator due to its flexibility in scenarios and scripts deployment, which helped during the experiments.

The results show a linear increase of the translation time for the various types of policies considered. Moreover, policies with a greater number of OpenFlow fields requires a slightly larger time to translate (see Figure 2). This is due to the additional computations need to retrieve the information and create the flow rule. Finally, the required time to convert a single policy varied between 1.62ms to 2.19ms, while for 32 policies we needed between 8.04ms to 25.06ms. In order words, even a large set (32) of very specific policies that deal with clients and resources directly (client-to-resource) only takes a few milliseconds.

VII. Related Work

Overall, solutions based on the standard computing infrastructure model share some common drawbacks related to the lack of flexibility. Solutions that employ non-controlled switches, for instance, are strongly attached to the physical port of such switches and changes made on the policies base will not take effect until the user authenticates again. Moreover, it is usually not possible to properly enforce the access control when more than one client is attached to the same physical port. On top of that, most of these solutions tend to use VLANs as the underlying mechanism for policy

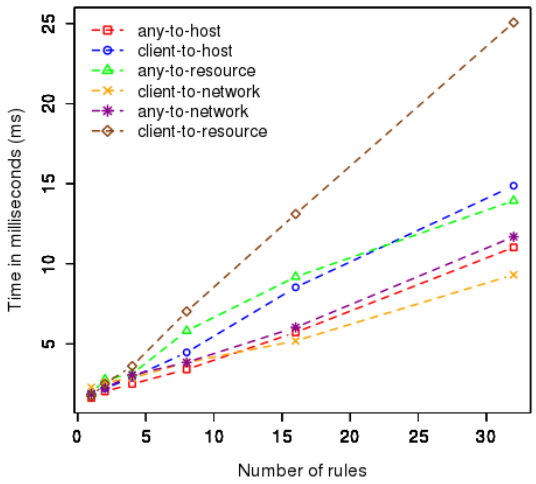

Fig. 6. Time to translate S-NAC security policy to OpenFlow rules.

enforcement, however, the maximum number of 4096 VLANs may be a real limitation in some scenarios.

Our SnacAgent does not suffer from the aforementioned limitations. Since it is able to update, remove and create flow rules on-the-fly and dynamically update the controlled switches, hence changes in policies do not have to wait for any user activity. Physical ports are just one of the parameters among many used to identify a client. Thus different roles or policies can be enforced for different clients connected to the same port. Finally, while our approach can benefit from the use of VLANs as a resource, its use is by no means a requirement.

Some approaches for NAC have already taken into account the SDN paradigm. Recently some works have done attention regrading the definition of network security policies using a high-level language in order to govern rights to network entities. Those works consists in using a high-level language to specify network policies that will be later, translated to SDN OpenFlow flow rules, and then enforced on underlying network topology (data plane) by the controller.

Batista et al.[4] proposed a framework to manage an OpenFlow network using the Ponder language. Ponder permits the management and specification of security policies for distributed systems. The main drawback of that work is the absence of a policy conflict resolution mechanism. Also, they did not make experiments in translating the proposed PonderFlow language in OpenFlow rules, in order to validate they approach.

Kim and Feamster[9] proposed Procera, a framework that uses a high-level language to define policies to be applied in SDN networks. The framework also focuses in event-driven networks, a characteristic that requires dynamic network reconfiguration. Scalability (increasing the number of rules) and performance (time required to translate the high-level language to OpenFlow rule) evaluation should be done to validate the framework.

Han et al.[6] proposed a multi-layered policy management framework for SDN. The framework relies on an application, control and data layers of SDN network architecture. The framework weakness is about not reacting to policy updates.

Lara and Ramamurthy [13] proposed OpenSec, a security framework to automate implementation of security policies. They also focus on reacting automatically to network security alerts. In OpenSec, the network operator define security rules that determine for which processing units (DDoS, DPI, and Encrypt) a traffic must be re-routed to. While, in this work we define security policies that determines which operations can be performed by network entities.

VIII. Conclusion and Future Works

In this work, we proposed a NAC solution which combines features of the SDN paradigm with OrBAC model. We believe that our approach complements previous work by providing a complete solution based on commercial applications while improving the flexibility and expressiveness of the authentication and authorization processes.

The S-NAC, however, is not without its limitations. While it is not the focus of this work, legitimate uses could cause some damages. Even though we are able to easily update the policy database and disconnect some users, there is no mechanism in place to detect or prevent malicious behavior of authorized users. We also do not prevent social engineering and other schemes which will indirectly allow non-authorized users to access confidential information or even some network resources.

While our approach enables authentication of more than one user per physical port it is important to ensure the security of such shared channel, otherwise, an attacker could potentially impersonate some client. To address this particular vulnerability, one could use MACSec and the Secure Device Identity, defined in the standards IEEE 802.1AE [2] and IEEE 802.1AR [3] respectively.

Moreover, the work allows dynamic changes of policies which impact current and future clients, but does not take into account dynamic contexts yet, such as time of the day, for instance. As future works, we plan to extend S-NAC to support the definition of dynamic security policies, through the use of Contexts in the OrBAC model.

The main scalability concerns are arguably the time to translate policies and the number of rules on the switches. Our translation process only takes a few milliseconds and our wildcards and on-demand policies help reducing the number of rules. We still have some scalability issues regardless. In our future work we will try to reduce table entries and make use of the high availability functions provided by some SDN controllers.

We also plan to create mechanisms that aggregate the enforcement of security policies to save switch's resources, such as the TCAM memory. With the concept of hierarchical policies in OrBAC model along with others techniques, we can achieve this goal.

REFERENCES

[1] Framework for SDN: Scope and Requirements. Technical Recommendation. Version 1.0. ONF, 2015.

[2] I. S. Association et al. Ieee 802.1 ae. *Media Access Control (MAC) Security*, 2006.

[3] I. S. Association et al. Ieee 802.1 ar secure device identifier, 2009.

[4] B. L. A. Batista and M. P. Fernandez. Ponderflow: A new policy specification language to sdn openflow-based networks. *International Journal on Advances in Networks and Services Volume 7, Number 3 & 4, 2014*, 2014.

[5] Y. Fan, D. Xiao, X. Mei, C. Liu, X. Yan, and C. Hu. A software-defined intelligent method for antenna design. In *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*, pages 470–474. IEEE, 2014.

[6] W. Han, H. Hu, and G.-J. Ahn. Lpm: Layered policy management for software-defined networks. In *Data and Applications Security and Privacy XXVIII*, pages 356–363. Springer, 2014.

[7] HP. Technical report, HP SDN controller architecture. Hewlett-Packard Development Company, L.P., Tech. Rep., September, 2013.

[8] A. A. E. Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin. Organization based access control. In *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*, pages 120–131. IEEE, 2003.

[9] H. Kim and N. Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, 2013.

[10] K. Kirkpatrick. Software-defined networking. *Communications of the ACM*, 56(9):16–19, 2013.

[11] 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*, 103(1):14–76, 2015.

[12] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[13] A. Lara and B. Ramamurthy. Opensec: A framework for implementing security policies using openflow. In *2014 IEEE Global Communications Conference*, pages 781–786. IEEE, 2014.

[14] J. Malinen et al. hostapd: Ieee 802.11 ap, ieee 802.1 x. Technical report, WPA/WPA2/EAP/RADIUS Authenticator. online: https://w1.fi/hostapd/, 2014.

[15] ONF. Openflow switch specification version 1.4.0. Technical report, ONF TS-012, 2014.

[16] R. S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994.