AuthFlow: Authentication and Access Control Mechanism for Software Defined Networking

Diogo Menezes Ferrazani Mattos, Lyno Henrique Gonçalves Ferraz,
Otto Carlos Muniz Bandeira Duarte
Grupo de Teleinformática e Automação
Universidade Federal do Rio de Janeiro (COPPE/UFRJ)
Rio de Janeiro – RJ – Brasil
Email:{menezes,otto}@gta.ufrj.br

***Abstract*—Software Defined Networks are being widely adopted by enterprise networks. Providing security features in these next generation networks, however, is a challenge. In this paper, we present the main security threats in Software Defined Networks and we propose AuthFlow, an authentication and access control mechanism based on host credentials. The main contributions of the proposed mechanism are twofold: i) AuthFlow authenticates hosts directly at the data link layer in an OpenFlow network, which introduces a low overhead and ensures a fine-grained access control; ii) AuthFlow uses the authentication credential to perform access control according to the privilege level of each host, through pairing the host credentials and the set of flows that belongs to the host. A prototype of the proposed mechanism was implemented over POX, an OpenFlow controller. The results show that the proposed mechanism denies access from unauthorized hosts, even in the scenario where a host has its access authorization revoked. Finally, we show that each host can have different levels of access to network resources according to their authentication credential.**

I. Introduction

Providing network security is a growing need for enterprise networks, the data center networks for cloud computing and networks that constitute critical infrastructure such as smart grids. The main difficulties to ensure a high level of security in networks are the variety of networking equipment, such as switches, routers, middleboxes, among others, and the fact that the end hosts that connect to the network may be not dependable and are able to present several vulnerabilities. Thus, the deployment of security policies requires that operator manually set network settings according to standards and to features of each piece of equipment. In addition, end hosts must also be certified to ensure network access only to hosts which have valid and authorized credentials.

The Software Defined Networking (SDN) paradigm decouples control from data forwarding, offering high programmability of control plane and a global view of the network. The adoption of this paradigm allows developing logically centralized and integrated security policies [1]. Thus, SDN facilitates the solution of complex problems in network security. The Application Programming Interface (API) OpenFlow [2] is the most successful implementation of Software Defined Networking. The OpenFlow controller is a piece of software that centralizes the network control plane. The forwarding plane is performed by high-performance switches compatible with OpenFlow. This new paradigm, however, presents some limitations to network security, as a component with malicious behavior can compromise the proper operation of the entire network, for example, performing a denial of service against the network controller. Therefore, an access control mechanism is essential to ensure security of SDN. Both authentication and the privilege level assigned to each host are also essential to ensure the proper operation of a Software Defined Network.

In this paper, we propose AuthFlow, an authentication and access control mechanism for Software Defined Networking. AuthFlow's contribution is two-fold: (i) Authenticating end hosts directly on Data Link Layer; and (ii) paring access credentials of end host with the flow set belonging to each end host. Authentication of end hosts on link layer is performed by the IEEE 801.X standard. It assures that authentication information exchanging is standardized between end hosts and the Authenticator. Thus, it does not require any change into current end hosts. The authentication mechanism encapsulates exchanged information into Extensible Authentication Protocol (EAP), which allows the use of different authentication methods. As AuthFlow authentication mechanism is direct at the data link layer, it takes the advantage of providing a low overhead when compared to an authentication application at the network layer or at the application layer. Authentication at higher layers depends on the assignment of an IP address to the end host and, also, depends on exchanging information on the authentication application. Another feature of the proposed mechanism is provisioning a fine-grained access control, since AuthFlow allows defining flow-based access policies for each host according to the credential of the end host. Hence, the control of which services a host can access shall be performed according to their credentials and, no more, according to their IP or MAC addresses. AuthFlow mechanism consists of an OpenFlow application running over POX OpenFlow controller, and two other components: Authenticator and RADIUS server. Authenticator receives messages of IEEE 802.1X standard and validates the credentials against the RADIUS server.

The main proposals of providing security to Software Defined Networks seek to develop security modules in the controller that facilitate the development of new secure applications [3], [4]. On the other hand, other proposals for authentication of end hosts on SDN consider that authentication should be done only after the host has received a temporary IP address and has been redirected to a Web site, where it should present its credentials [5], [6]. These proposals, however, are prone to address spoofing attacks, and they introduce greater control overhead when compared to AuthFlow. Other proposals describe some threats to SDN and indicate possible directions for addressing these threats [7], [8]. Whereas SDN is prone to security threats, a prototype of AuthFlow mechanism was developed and evaluated in the experimental environment Future Internet Testbed with Security (FITS)[1] [9]. The efficiency of the proposed access control mechanism is evidenced by experiments that show that a host is denied to access the network, both in case a host is unauthenticated and in case a host has withdrawn its authentication. The results of the prototype evaluation show that end hosts have different views of the network, allowing or denying access to each service, depending on the level of privilege that each end host has according to its access credentials.

The rest of the paper is organized as follows. Section II discusses the security limitations of the Software Defined Networking paradigm. We describe AuthFlow mechanism on Section III. Section IV presents our experimental results and evaluates the proposed mechanism. Section V presents related work. Section VI concludes the paper.

II. Security Threats of Software Defined Networking

The global and centralized view of SDN allows the control logic of the security applications to be more complete and integrated than the current ones [4] and, thus, simplifies handling complex network security problems. Security applications rely on the centralized network controller for implementing flow definition policies based on states, and also flow based security, for example, intrusion detection algorithms or malfunction detection. Nevertheless, the creation of security applications on SDN is a challenge, because the security of the SDN itself is still being questionable [7]. Security challenges of Software Defined Networks fall into three categories: Denial of service, lack of trust between components and vulnerabilities of components.

Denial of Service can occur in both data and control planes. In data plane, a malicious host that generates false flows can exhaust both bandwidth resources and memory resources, or flow table entries, of switches on the network. Denial of service in the control plane can be achieved in two different points of the network: The controller and the communication channel between controller and switches. An attacker may exhaust the processing capacity of the network controller when sending a lot of packets with different headers. This happens because every packet is analyzed by the switch and a packet, whose header does not match any flow already set, is sent to the network controller. Thus, in a scenario where a switch sends an unusual amount of new packet headers to the controller, this may exhaust controller processing resource and controller will not be able to respond to requests of new flows in a timely manner. Likewise, denial of service can be achieved when the communication channel between switches and controller is intentionally jammed. If there is not enough bandwidth or redundancy in the communication channel, a malicious switch can generate enough traffic to overload the channel and, hence, prevent communication of the controller with other switches. Authentication of end hosts and switches, using Secure Socket Layer (SSL) and Public Key Infrastructure (PKI), is able to avoid this threat. Authentication is important because only authorized nodes access the network and, in case of identifying malicious behavior, the authentication of a node can be revoked and the node may be kicked off the network.

Lack of Trust between Network Components hampers the SDN, because applications running over the controller can behave maliciously. In this case, the controller must be able to identify which applications are trustworthy and which are malicious. Thus, a possible measure to increase security on applications running on a SDN is applying mechanisms for certifying applications and for establishing chains of trust and attestation. Some proposals to provide security on SDN consider deploying a security core on controller itself to ensure the safe execution of applications, without interfering in other application or performing prohibited actions on the network [3], [4]. On the other hand, the lack of trust also affects logging actions, since log, when it is done, presents no assurance that the action has really occurred and if the application, that has registered, has been behaving properly. One possible solution is adopting log for each application, signed by each application. The application, in its turn, would be certified and signed by developers.

Vulnerability of Components is not a security challenge of only this new network paradigm, but it becomes more critical in SDN because vulnerability in a controller node makes the entire network vulnerable. Thus, there are three possible sources of vulnerabilities: switches, controller and management hosts. Vulnerabilities at a switch may allow an attacker, who gains access to a switch, performs an attack on the control plane, such as forging messages from other switches to exhaust the resources of the controller. Vulnerabilities at the controller allow an attacker to alter the control plane or even run a new application to control the network. Vulnerabilities in a management host allow an attacker to incorrectly setup control plane. To prevent this type of attacks, there are some measures, such as, attestation of the control applications, use of dual certification protocols between applications and management hosts and, finally, replication of control applications for fault and intrusion tolerance.

Among the major challenges of securing Software Defined Networks, we should also highlight that these networks required three main features: scalability, responsiveness and availability [5]. In order to provide such features, we should solve the challenge of placing controllers in the network [8]. In this sense, placement of controllers and the number of replicated network controllers should comply with the safety requirements, such as, scalability, availability and response time of the network.

Authentication, authorization and access control are essential primitives for Software Defined Networks. These primitives, together with the attestation and replication of controllers, are the basis of a secure network, in which malicious components, i.e., vulnerable or bad behaved nodes, can be identified and isolated [5], [4].

III. AuthFlow Mechanism

The main idea of AuthFlow is performing authentication using Data Link Layer protocols and pairing the identity of

[1]FITS is an inter-university testbed network which was developed through a partnership between Brazilian and European institutions. More information on http://www.gta.ufrj.br/fits/.

a host with the flows created by itself on the network. Thus, the proposed mechanism applies the IEEE 802.1X standard and Extensible Authentication Protocol (EAP). EAP encapsulates authentication message exchanged between the supplicant host[2] and RADIUS authentication server. The authenticator is employed in AuthFlow as a process that communicates with an OpenFlow application, running over the POX controller. The application allows or denies network traffic from a supplicant host depending on the result of authentication between the supplicant and authenticator.

AuthFlow mechanism adopts the IEEE 802.1X standard because, while it specifies the authentication directly at the data link layer, it is a widely adopted standard. IEEE 802.1X requires no changes on end hosts for network authentication. When a host compatible with the IEEE 802.1X standard starts, it also starts the authentication phase by sending a `start` message to a reserved multicast MAC address (`01:80:C2:00:00:03`), with the Ethernet type set to `0x888E`. Therefore, authentication procedure depends on neither any prior knowledge of the host about the network, nor a translation of an IP address into a MAC address. This procedure prevents a host of receiving a temporary IP for only receiving a definitive IP depending on the authentication result. The use of the IEEE 802.1X standard greatly simplifies the authentication process.

Following, we discuss the architecture and operation of AuthFlow. We consider a use case where AuthFlow authenticates virtual routers on a network infrastructure of hybrid pluralistic Xen and OpenFlow virtualization platform [9]. Our proposal, however, is not limited to this use case and AuthFlow can be used without any change in the authentication of end hosts in an OpenFlow network. In the considered use case, hosts, that compose the OpenFlow network, are virtual machines that behave either as end hosts or as routers.

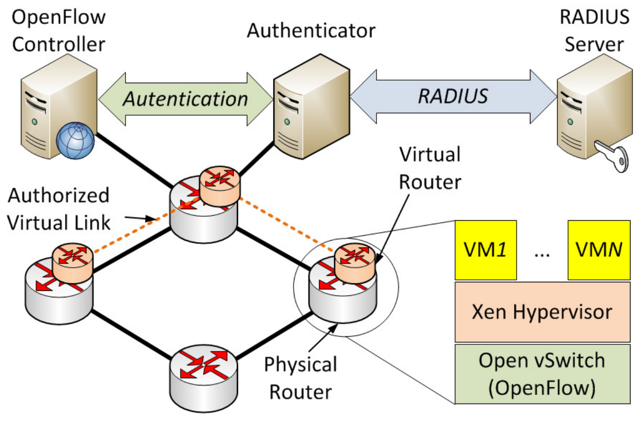


Figure 1. The architecture of AuthFlow is composed of three main nodes: OpenFlow controller, the Authenticator and the RADIUS server. A virtual machine, which authenticates on the network, starts EAP authentication according to IEE 802.1X standard. Authenticator decapsulates authentication messages from EAP packets, performs authentication of the virtual router against RADIUS server, and informs OpenFlow controller of the result.

In our considered use case, the architecture of AuthFlow consists of physical machines running OpenFlow software switches, a POX controller, an Authenticator and a RADIUS authentication server, as shown in Figure 1. Physical and virtual machines act as routers and, then, are called respectively physical routers and virtual routers. Physical routers are nodes with Xen virtualization system and host virtual routers. Packet forwarding between physical and virtual routers is done by a software switch compatible with OpenFlow API. Our architecture places an Open vSwitch[3], in each physical router, as the OpenFlow forwarding engine. The adoped virtualization model is the a hybrid model used in Future Internet Testbed with Security (FITS) [10], [9]. The POX controller runs an application to handle packet forwarding, in particular, packets[4] of the IEEE 802.1X standard. IEEE 802.1X packets are forwarded directly to the Authenticator. Authenticator is a RADIUS client that implements the IEEE 802.1X standard and forwards the content of EAP messages to RADIUS. The authenticator was developed as an adaptated version of `hostapd`[5], an Authenticator that is primary used in wireless networks. The `hostapd` was modified to inform our POX application about the authentication of virtual networks. Thus, when performing authentication of a virtual network, `hostapd` sends a confirmation message of authentication success for POX over a secure, encrypted and authenticated channel using SSL 3.0 (Secure Socket Layer) standard and Public Key Infrastructure (PKI). The authentication server is a RADIUS server that extracts the information encapsulated in EAP authentication and validates the credentials presented by virtual routers against a database. As EAP allows the use of several different authentication methods, the method adopted was MSCHAP v2 [11], which authenticates virtual router against a database using credentials like username and password. We deployed a Lightweight Directory Access Protocol (LDAP) database to store username and password pairs. Nonetheless, the authentication method and the database are not essential components to describe the proposed mechanism, as they do not interact directly with the OpenFlow network.

The AuthFlow authentication mechanism works as follows. A virtual router sends an authentication request, standardized by IEEE 802.1X, and POX controller redirects it to the Authenticator. Then, the Authenticator responds it and the Supplicant host sends its credentials. The Authenticator checks the credentials of the Supplicant against RADIUS server, running the authentication method defined in EAP. If credentials are correct, the Authenticator sends a `success` message for Supplicant host and sends an authorization and confirmation message for POX via a SSL secure channel. This message identifies the Supplicant by its MAC address, confirms the success of the authentication, and also informs the identity of the Supplicant. After authentication, our application running over POX allows the Supplicant host to access network resources. In case of revocation of the credentials of the Supplicant host, the Authenticator communicates POX, which immediately denies the host access to the network.

Access control on AuthFlow works by denying or allowing access of virtual machines to virtual links. Thus, when starting an OpenFlow network that employs the AuthFlow, all network

[2]The nomenclature for supplicant, authenticator and authentication server is defined by the IEEE 802.1X standard.

[3]http://www.openvswitch.org/.

[4]We call packet datagrams of whatever layer because packet is more general than frames and OpenFlow API access all layers from Ethernet link layer to the transport layer.

[5]http://hostap.epitest.fi/hostapd/.

links are initially blocked for any communication, including links that interconnect the OpenFlow switches, i.e., links belonging to the network core. If so, these links do not need to perform authentication process to allow their traffic. Thus, AuthFlow mechanism performs the topology discovery of the network core via Link Layer Discovery Protocol (LLDP). LLDP packets are forwarded link to link and, as the controller generates and verifies each LLDP packet transmitted in the network core, the controller is able to identify which links are between OpenFlow switches and which links are connecting end hosts. LLDP packets generated by the controller are tagged with a nonce to avoid replay or spoofing attacks.

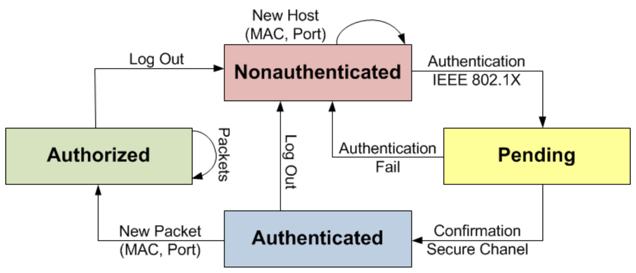


Figure 2. AuthFlow access control is a four-step mechanism. I) Nonauthenticated, hosts that have not initiate yet the authentication process; ii) Pending, while the authentication process occurs; iii) Authenticated, when the host has already completed successful authentication; iv) Authorized; when a host is authorized to access the network and the host owns a set of authorized flows already defined on the network.

Figure 2 stands for the state diagram of the access control mechanism of AuthFlow. In Figure 2, a host is always represented by a tuple `(MAC ,port)`, where MAC is the MAC address of the host and port is where the host is connected to the ingoing switch. The figure summarizes the authentication process. A host joining the network is initially in `Nonauthenticated` state and all traffic generated or addressed to this host is dropped, except for Ethernet traffic with type set to `0x888E` (IEEE 802.1X). IEEE 802.1X traffic is forwarded from host to the Authenticator as a *multicast* flow and, in the opposite direction, it is forwarded as *unicast* flows, as the Authenticator learns the MAC address of the Supplicant host after receiving the first packet of IEEE 802.1X. As soon as the host starts, it also starts the authentication procedure by sending the `start` message. A state change of the host to `Pending` ensues. In `Pending` state, all traffic of the host is still being dropped, but host is awaiting confirmation from the Authenticator to POX of the success of its authentication and what credentials were used for authentication. As authentication is successfully confirmed, POX moves the host to the `Authenticated` state. In this state, our application running over POX releases access to network resources that the host is allowed to, according to its credentials. Nevertheless, when there is traffic to the host, POX checks whether traffic is in accordance with the policies related to host credentials. If policies are consistent with the use of the network, the host is moved to `Authorized` state and accesses network resources according to its privileges and policies.

Considering the proposed access control, releasing or denying end hosts traffic is performed depending on the credentials presented by the host while it authenticates. The authentication tuple `(MAC , port)` is paired with the identity of the host. Thus, it is possible to correlate the flow of a given host and its identity. Flows and identiy correlation occurs as follows. If an OpenFlow flow has among its fields the source MAC address (`dl_src`) and the incoming port on the switch (`in_port`) equal to those in the authentication tuple, the authentication credentials and identity[6] of host are assigned to this flow. Thus, the forwarding decision about this flow may take into consideration also the credential of the host. Therefore, access control is further refined as it is done according to the credential of the flow and not only according to the OpenFlow fields. Similarly, if an OpenFlow flow has among its fields the destination MAC address (`dl_dst`) and output port (`output`) equal to those in the authentication tuple, the authentication credential of the host is also attributed to the flow. Hence, AuthFlow mechanism also controls flows addressed for a host in accordance with host identity. Therefore, AuthFlow access control policies may define both outgoing and incoming rules to end hosts according to their identities.

IV. Experimental Results

We developed an AuthFlow prototype on an island of the Future Internet Testbed with Security (FITS) [10], [9]. The prototype runs Xen hypervisor 4.1.4 to provide virtual domains that act as end hosts. OpenFlow network is realized by a software switch Open vSwitch 1.2.2. Open vSwitch [12] is set to be controlled by POX[7]. The forwarding and the access control applications were developed in Python and runs over POX. Our prototype employs a modified version of `hostapd` as Authenticator. Our version of `hostapd` creates a secure channel from the Authenticator to POX to inform the controller whether there is a new authentication or a loss of authentication of an end host. AuthFlow prototype adopts FreeRADIUS v2.1.12[8] as its RADIUS server. As proof of concept, the authentication method tested in the prototype was EAP-MSCHAP v2 [11], in which a pair `(username, password)` is checked against a LDAP database.

We evaluated the performance of the prototype with `Iperf`[9], `nmap`[10] and `tcpdump`[11] tools. Four personal computers compose our experimental scenario. All computers run AuthFlow prototype. In each computer was instantiated a virtual machines that act as router, sends or receives packets, depending on each experiment. Summing up, we instantiated four virtual machines at all. All personal computers were equipped with Intel Core 2 Quad 2.4 GHz processor, 3 GB of RAM and run Debian Linux 3.2.0-4-amd64. Each computer has at least two network interfaces and all of them are configured to operate at 100 Mb/s to ensure homogeneity, since there were also 1 Gb/s network interfaces. Virtual machines are configured with one virtual CPU, 128 MB of RAM and run Debian Linux 3.2.0-4-amd64. Virtual machines run routing protocols over eXtensible Open Router Platform (XORP) [13], a software-based routing platform.

[6]We consider that credential is the proof of the identity of a host.

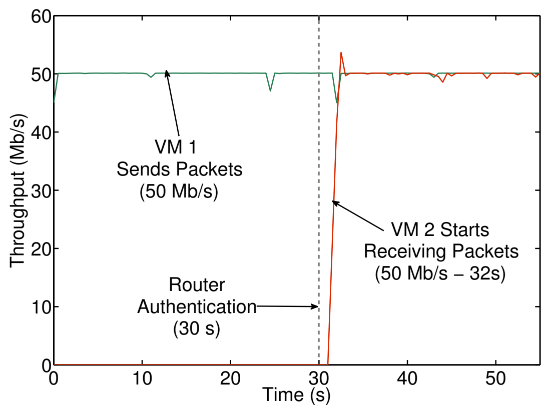
[7]The POX controller used in our prototype is a development branch of the controller used in FITS, to support AuthFlow mechanism.

[8]http://freeradius.org/.

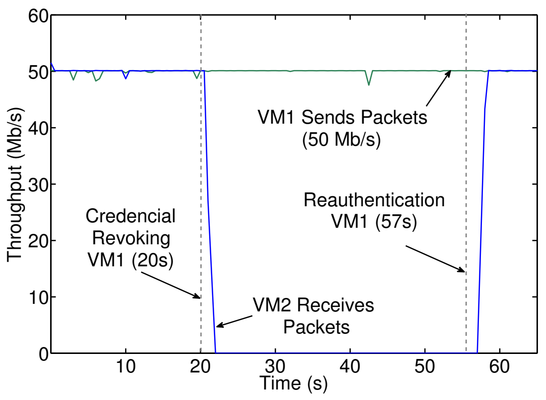
[9]http://iperf.sourceforge.com/.

[10]http://www.nmap.org/.

[11]http://www.tcpdump.org/.



(a) After 30 s, the virtual router, between virtual machines 1 and 2 (VM1 and VM2), authenticates and, thus, allows the network traffic from VM1 to VM2.



(b) At 20 s, VM2 authentication is revoked and, thus, network traffic is blocked. At the end of the experiment, the authentication is restored and the traffic of VM2 is allowed again.

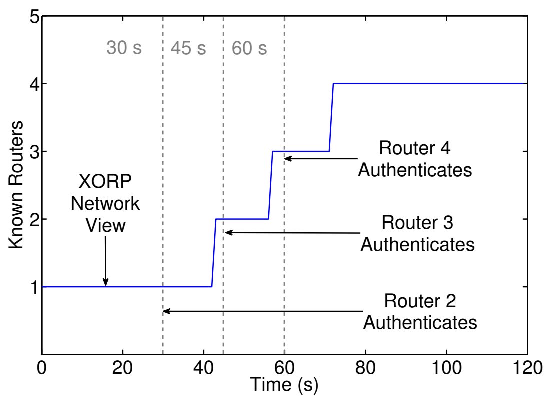
Figure 3. Traffic forwarding blocks before authentication and when credentials are revoked. Virtual Machine 1 (VM1) sends packets to the Virtual Machine 2 (VM2). (a) Router between VM1 and VM2 authenticates itself. (b) Revocation of VM1 authentication.

The first experiment evaluates the effectiveness of AuthFlow mechanism into dropping unauthorized traffic. Packets forwarding of a flow is only released after authentication, otherwise, packets are dropped. The experimental scenario is simple: Virtual Machine 1 (VM1) sends packets which are addressed to Virtual Machine 2 (VM2). Packets are forwarded by a virtual router between VM1 and VM2. It is assumed that Virtual Machines 1 and 2 were previously authenticated on the network and the virtual router that interconnects them is not yet authenticated. The VM1 generates a UDP flow of 1472 B packet size at a constant rate of 50 Mb/s. As the router is not authenticated, the flow does not reach the VM2. After 30 s, the router authenticates into the network, as shown in Figure 3(a), and the UDP flow reaches VM2. Figure 3(a) shows that there is a delay of the order of 2 s to 2.5 s between starting router authentication and the effective releasing of network access. This delay is due to the IEEE 802.1X authentication process plus the time of definition of new OpenFlow flow. This delay only occurs when the router joins the network.

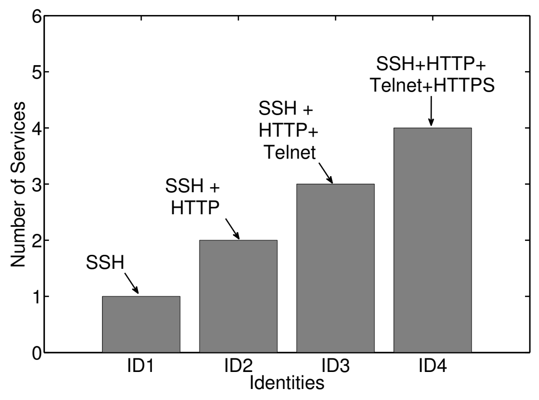
The second experiment shows the effectiveness of AuthFlow mechanism when experiencing a revocation of a credential, as shown in Figure 3(b). The scenario consists of a virtual machine VM1, which communicates directly with another virtual machine, VM2, without a router between them. Again, we assume that initially both virtual machines are authenticated. After 20 s, VM2 authentication is revoked, then VM2 access to the network is blocked for both sending and receiving packets. We observed that the delay of blocking the network traffic of VM2 is less than 1 s. After 57 s, VM2 authentication is restored back and VM2 restarts to receive packets. Restoring authentication delays approximately 2 s, as well as the authentication of a new host. We should emphasize the effectiveness of the proposed mechanism into releasing and blocking traffic through forwarding or dropping packets, respectively, associated with authentication and revocation of credentials procedures. Therefore, AuthFlow constitutes a strong ally in defending Software Defined Networks against denial of service attacks due to its effectiveness of dropping unauthorized traffic.

The next experiments show the network view from the perspective of authenticated hosts, i.e., which hosts an authenticated host reaches and which services that host can access on the network. Figure 4(a) shows the network view based on a virtual router running a link state routing protocol, OSPF (Open Shortest Path First). We consider a ring topology, connecting all four virtual routers. The experiment consists of timely verifying the OSPF database of an already authenticated router and, then, identifying how many neighboring the observed virtual router already knows. At the beginning of the experiment, the virtual router is the only one that is already authenticated. After 30 s, the second virtual router authenticates itself. At 45 s, the third one authenticates itself, and finally at 60 s, the fourth router authenticates itself. It is noteworthy that the delay between authentication and discovery of each new virtual router, shown in Figure 4(a), is due to the handling broadcast/multicast packets adopted by our OpenFlow network. To avoid overloading the network, each flood packet is matched against a flow rule to drop packets with the same defining fields during the next 5 s.

The fourth experiment intends to prove one of the main advantages offered by AuthFlow, which consists in defining forwarding rules based on the authentication credential. The key idea is to assure that authentication provides an “ identification” of flows corresponding to services that are authorized for a host. Thus, authentication by AuthFlow enables releasing flows corresponding to services that have been released to that credential, and blocking all other flows. The experiment consists of a authenticated requester host, with one of four possible identities (ID1, ID2, ID3 and ID4), access a service provider host on the network. Each identity allows access to a set of network services (one, two, three or four services, respectively). Therefore, the requester host performs a port scan (`nmap`) on the provider host. In our scenario, the requester host is the same for the four identities, keeping the same IP and MAC addresses during the entire experiment. The only modification between each test scenario is the authentication of the requester host with another identity. Figure 4(b) shows the number of services that the requester host access on the



(a) Number of neighbors that a virtual router discovers at the network according to the number of authenticated routers.



(b) Number of services provided by the network in accordance with the credentials which the end host presented to authenticate on the network.

Figure 4. Network view from a virtual router (a) and network view from an end host (b). Each credential is associated with an identity and accesses a set of services on the network.

service provider host. Thus, we observe that a host can only access the services released to that identity with which it is authenticated. It is noteworthy that the port scan returns the “filtered” state for these ports that has no service allowed. It shows that blocking other services is performed by discarding `SYN` packets, which, in fact, occurs because of rules installed by POX controller as it verifies that a host has not the proper privilege level to access a service.

V. Related Work

The security of Software Defined Networks, particularly the security of OpenFlow networks, is a subject fully argued currently. There are proposals for developing security applications on OpenFlow network infrastructure, as there are others that seek to ensure the security of the infrastructure itself. Nevertheless, secure authentication, access control, scalability, responsiveness, availability and confidentiality continue to be challenges on SDN [7].

Kreutz *et al.* classify the main vectors of attacks on Software Defined Networking and present possible countermeasures to protect the network against these attacks [7]. Kreutz *et al.* are restricted to attacks on the resilience and on the trustworthy of the network. on the other hand, to ensure the confidentiality and availability of SDNs, FITS, Future Internet Security Testbed, bases on a hybrid virtualization system over Xen and OpenFlow [9]. The main idea of FITS virtualization platform is to provide isolation of communication and resources between virtual networks on a SDN infrastructure. FITS adopts a forwarding scheme based on packet queuing to ensure bandwidth allocation for each virtual network. FITS tags packets of each network with a VLAN tag in order to multiplex the virtual network to which a packet belongs. In FITS, however, there are no mechanisms for authentication or access control between virtual machines and network infrastructure. Therefore, AuthFlow is complementary to FITS in the sense of the former ensures secure access control to the latter.

The UPV/EHU network [14], an European OpenFlow-testbed network, also adopts a proposal for authentication based on IEEE 802.1X standard. This proposal, however, does not consider the use of authentication credentials of a node for controlling the access of flow definition. The main differential of AuthFlow proposal is to perform the paring of authentication credentials and set of defined flows. Hence, at any time, it is possible to identify which node has generated or is receiving a flow on a given switch. Depending on the node behavior, if necessary, AuthFlow allows revoke a node authentication. The proposal provides to the network controller a primitive of defining forwarding rules according to the identity of each host. This primitive is an advantage of AuthFlow in comparison with other proposals.

Guenane *et al.* propose an authentication mechanism for virtual networks using EAP-TLS. The mechanism is implemented on smart cards [15]. The proposal focuses on guaranteeing access for virtual machines and for customers of virtual networks to smart cards, which implement the TLS protocol and encapsulate messages on EAP. EAP messages are sent to a RADIUS server that authenticates the components and customers of the virtual network through mutual authentication provided by exchanging certificates signed by a Certification Authority. This proposal does not define how the mechanism should control the access of network nodes and how authentication may be performed to authorize customer accesses network resources. AuthFlow is complementary to this proposal for authentication with smart cards, since the authentication method is encapsulated in EAP. Thus, AuthFlow may control access of virtual routers to network resources using EAP-TLS authentication method.

Resonance [5] and Ethane [6] are other proposals to authenticate nodes in a Software Defined Network. Both argue that node authentication must be done through a web site, in which the user must submit their credentials. These approaches present a basic restriction that is the need for a node has a browser installed to accesses web content. This requirement is quite limiting when considering virtual network environments composed of extremely lightweight virtual machines that do not have graphical interface or web browsers. Moreover, another disadvantage of these proposals is to limit authentication to username and password method, while AuthFlow adopts authentication based on EAP encapsulation, thus authentication method may be whatever, since it is compatible with EAP. As aforementioned, even robust authentication methods based on secure microcontrollers are possible to be applied with AuthFlow mechanism. Another advantage of AuthFlow, when compared with these proposals, is that authentication is performed directly at Layer 2. Therefore, there is no need for a node to acquire an IP address before logging in, as it happens in Resonance or Ethane. In AuthFlow, when a host joins the network, it starts its authentication according to IEEE 802.1X standard, directly at the Data Link Layer, authenticating the pair of MAC address and the switch port that the host is connected. This procedure prevents a node of using a spoofed MAC address, unlike proposals that are not intended for preventing forgery of addresses.

The proposals FRESCO [4] and FortNOX [3] define a set of security primitives for OpenFlow networks. The FortNOX proposal advocates the creation of a secure execution core for applications over an OpenFlow-network controller. The secure core prevents an application to perform actions that interfere in control rules of another application. The proposal supports network slicing between applications on the same controller, which provides a finer control of privileges and better limits the control domain for each application than done by FlowVisor [16]. FlowVisor, in its turn, slices the network between multiple controllers; however, it does not provide a secure policy between controllers, in order to the actions of a controller do not affect the others. Following the main idea of implementing a secure core for running applications, FRESCO defines a set of primitives and a modular language for the development of secure applications for OpenFlow networks. These proposals are related to the AuthFlow. Thus, the proposal of this paper can be employed as a FRESCO secure module, for example, to allow the use of a new security primitive, the authentication primitive.

VI. Conclusion

The security of enterprise networks depends on efficient mechanisms for access control and authentication of hosts. As Software Defined Networking (SDN) is being widely adopted by enterprise networks, the challenge of providing security to SDN has become even more critical. In this paper, we propose the AuthFlow, a mechanism for authentication and access control to the infrastructure of an OpenFlow Software Defined Network. The mechanism is based on IEEE 802.1X standard and on a RADIUS authentication server. We implement the AuthFlow mechanism as a RADIUS authentication against a LDAP database. The proposal, however, is extensible to other authentication methods, such as EAP-TLS which authenticates hosts based on certificate exchanging. We developed and evaluated a prototype and our results show that the proposed authentication mechanism prevents unauthorized hosts from accessing network resources, even when hosts are already authenticated and, after some time, they lose their privileges. Results also show that AuthFlow is more efficient than other proposals, as it introduces lower overhead of control data, and allows the definition of policies for flow access control according to the access credentials of each host.

As future work, we intend to deploy AuthFlow in Future Internet Testbed (FITS) as its authentication mechanism and its default access control. We also intend to extend AuthFlow for new authentication methods, such as EAP-TLS, allowing the use of signed certificates as access credentials.

Acknowledgments

The authors would like to thank FINEP, FUNTTEL, CNPq, CAPES, FAPERJ and UOL for their financial support.

References

[1] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, “Logically centralized?: state distribution trade-offs in software defined networks,” in *Proceedings of the First workshop on Hot topics in software defined networks*, ser. HotSDN ’12. Helsinki, Finland: ACM, 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev., 2008*, Mar. 2008.

[3] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, “A security enforcement kernel for openflow networks,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN ’12. New York, NY, USA: ACM, 2012, pp. 121–126.

[4] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, “Fresco: Modular composable security services for software-defined networks,” in *Proceedings of Network and Distributed Security Symposium*, 2013.

[5] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, “Resonance: Dynamic access control for enterprise networks,” in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, ser. WREN ’09. New York, NY, USA: ACM, 2009, pp. 11–18.

[6] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, “Ethane: Taking control of the enterprise,” *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 1–12, 2007.

[7] D. Kreutz, F. M. Ramos, and P. Verissimo, “Towards secure and dependable software-defined networks,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’13. New York, NY, USA: ACM, 2013, pp. 55–60.

[8] B. Heller, R. Sherwood, and N. McKeown, “The controller placement problem,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN ’12. New York, NY, USA: ACM, 2012, pp. 7–12.

[9] I. M. Moraes, D. M. Mattos, L. H. G. Ferraz, M. E. M. Campista, M. G. Rubinstein, L. H. M. Costa, M. D. de Amorim, P. B. Velloso, O. C. M. Duarte, and G. Pujolle, “Fits: A flexible virtual network testbed architecture,” *Computer Networks*, no. 0, pp. –, 2014.

[10] P. H. Guimarães, L. Ferraz, J. V. Torres, D. Mattos, A. Murillo, M. A. Lopez, I. Alvarenga, C. Rodrigues, and O. C. M. B. Duarte, “Experimenting Content-Centric networks in the future internet testbed environment,” in *IEEE International Conference on Communications 2013: IEEE ICC’13 - Workshop on Cloud Convergence: challenges for future infrastructures and services (WCC 2013) (ICC’13 - IEEE ICC’13 - Workshop WCC)*, Budapest, Hungary, Jun. 2013, pp. 1398–1402.

[11] G. Zorn, “Microsoft PPP CHAP Extensions, Version 2,” RFC 2759 (Informational), Internet Engineering Task Force, Jan. 2000.

[12] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, “Extending networking into the virtualization layer,” *Proc. HotNets*, Oct. 2009.

[13] M. Handley, O. Hodson, and E. Kohler, “XORP: An open platform for network research,” *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 53–57, 2003.

[14] J. Matias, E. Jacob, N. Toledo, and J. Astorga, “Towards neutrality in access networks: A nando deployment with openflow,” in *ACCESS 2011, The Second International Conference on Access Networks*, Luxembourg City, Luxembourg, Jun. 2011, pp. 7–12.

[15] F. Guenane, N. Samet, G. Pujolle, and P. Urien, “A strong authentication for virtual networks using eap-tls smart cards,” in *Global Information Infrastructure and Networking Symposium (GIIS), 2012*, 2012, pp. 1–6.

[16] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, “Flowvisor: A network virtualization layer,” Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium, Tech. Rep., 2009.