Noname manuscript No.
(will be inserted by the editor)

AuthFlow: Authentication and Access Control Mechanism for Software Defined Networking

Diogo Menezes Ferrazani Mattos · Otto Carlos Muniz Bandeira Duarte

Received: date / Accepted: date

Abstract Software Defined Networking is being widely adopted by enterprise networks, whereas providing security features in these next generation networks is a challenge. In this article, we present the main security threats in Software Defined Networking and we propose AuthFlow, an authentication and access control mechanism based on host credentials. The main contributions of our proposal are threefold: i) a host authentication mechanism just above the MAC layer in an OpenFlow network, which guarantees a low overhead and ensures a fine-grained access control; ii) a credential-based authentication to perform an access control according to the privilege level of each host, through mapping the host credentials to the set of flows that belongs to the host; iii) a new framework for control applications, enabling Software Defined Network controllers to use the host identity as a new flow field to define forwarding rules. A prototype of the proposed mechanism was implemented on top of POX controller. The results show that AuthFlow denies the access of hosts either without valid credentials or with revoked authorization. Finally, we show that our scheme allows, for each host, different levels of access to network resources according to its credential.

Keywords Access Control · Authentication · Software Defined Networking

This work was sponsored by CNPq, CAPES, and FAPERJ.

D.M.F. Mattos · O.C.M.B. Duarte
Grupo de Teleinformática e Automação
Universidade Federal do Rio de Janeiro (COPPE/UFRJ)
Rio de Janeiro – RJ – Brasil
Tel.: +55(21)3938-8635
E-mail: {diogo, otto}@gta.ufrj.br

D.M.F. Mattos
Laboratoire d'Informatique de Paris 6
Sorbonne Universities, UPMC Univ Paris 06
Paris, France
E-mail: Diogo.Mattos@lip6.fr

1 Introduction

Providing security is a fundamental need for enterprise networks, multi-tenant data center networks, and critical-infrastructure networks, such as smart grids [16,9]. The main difficulties to ensure a high level of security in networks are the variety of networking equipment, such as switches, routers, and middleboxes, as well as, end host vulnerabilities. Thus, the deployment of security policies requires specific network settings, according to standards and features of each network equipment, which operators have to set manually [6].

Software Defined Networking (SDN) paradigm decouples network control from data forwarding, offering high programmability of control plane and a global view of the network. The adoption of this paradigm promotes the development of logically centralized and integrated security policies, which simplifies the solution of complex network security problems [13]. OpenFlow [10] is an Application Programming Interface (API) that is the most successful implementation of Software Defined Networking[1]. The OpenFlow controller is a software module that logically centralizes the network control plane. The centralization provided by this new paradigm, however, must be carefully protected because of network security concerns. A malicious or misused behavior of a unique component 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 mandatory for SDN security. Both authentication and the privilege level assigned to each host are essential to ensure the proper SDN operation.

In this article, we propose AuthFlow, an authentication and access control mechanism for Software Defined Networking. Our proposed mechanism provides the following contributions. Provision of a host authentication mechanism at a low layer, for instance, immediately above the Medium Access Control (MAC) layer, guarantees a low overhead and a fast access control procedure. A fine-grained access control facility based on mapping authentication credentials of the supplicant host into the flow set belonging to each host. A control framework feature that allows control applications to define flow rules according to the authentication credentials of supplicant hosts. The authentication procedure occurs just above the MAC layer and complies with IEEE 801.X standard. The IEEE 801.X assures that authentication information exchanging is standardized between the supplicant host and the authenticator. Thus, it does not require any change into current hosts. As AuthFlow authenticates at lower layer level, it introduces lower overhead when compared to authentication processes 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.

[1] Considering a three-layer SDN model, we identify three API levels: southbound API, east/westbound API, and northbound API. In this model, OpenFlow is an example of southbound API.

Current proposals for providing security to Software Defined Networking seek to develop security modules in the controller that facilitate the development of new secure applications [20,22]. Other proposals give hosts a temporary IP address to access a Web site, where they present the credentials [18,3, 23]. These proposals, however, are prone to address spoofing attacks, and they introduce greater control overhead when compared to AuthFlow. A prototype of AuthFlow mechanism was developed and evaluated in the experimental environment Future Internet Testbed with Security (FITS)[2] [17]. The results show that AuthFlow is effective to authenticate and to control access of Software Defined Networking, while it keeps a lower overload of control data when compared to a standard network authentication approach.

The rest of the paper is organized as follows. Section 2 discusses the security limitations of the Software Defined Networking paradigm. We describe AuthFlow mechanism on Section 3. Section 4 presents our experimental results and evaluates the proposed mechanism. Section 5 presents related works. Section 6 concludes the paper.

2 Security Threats of SDN

The global and logically centralized network view on SDN allows the control logic of security applications to be more complete and integrated than the ones of legacy networks [22,5] and, thus, simplifies handling complex network security problems. Security applications rely on the centralized global network view for implementing flow definition policies based on states, and also flow based security, such as intrusion and malfunction detection algorithms [19]. Nevertheless, the creation of security applications on SDN is a challenge, because the security of the SDN itself is still questionable [12,11,14]. Therefore, we propose the classification of the security threads on Software Defined Networking into three categories: Denial of Service (DoS), lack of trust between components and vulnerabilities of components.

Denial of Service (DoS) 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 against the control plane can be achieved in two different locations of the network: The controller and the communication channel between controller and switches. An attacker may exhaust the processing capacity of the network controller by sending a lot of packets with different headers, which do not match any already defined flow, and, thus, are sent to the controller. Likewise, Denial of Service can be achieved when the communication channel between switches and controller is intentionally jammed. A

[2] 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/.

malicious node can generate enough traffic to overload the channel and prevent communication between the controller and the switches. Authentication of end hosts and switches, using Secure Socket Layer (SSL) and Public Key Infrastructure (PKI), is able to avoid these threats. Authentication is important because only authorized nodes access the network and, in case of malicious behavior, the node authentication can be revoked and the node may be isolated from the network.

Lack of Trust between Network Components hampers the SDN, as applications running on top of the controller can behave maliciously. Controller must identify which applications are trustworthy and which are malicious. Thus, possible measures to increase security are applying mechanisms for certifying applications and for establishing chains of trust and attestation. FortNOX and FRESCO create a safe execution core at the SDN controller, which manages and limits applications actions on the network [20, 22]. Besides, lack of trust affects logging, since untrustworthy applications may log fake records. Hence, applications should sign their own logs, and developers should be certified. Moreover, certified developers could securely certify their applications.

Vulnerability of Components is a security challenge not restricted only to SDN, but it becomes more critical since a vulnerability of a controller lets 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, to attack 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 to run a new application for controlling the network. Vulnerabilities in a management host allow an attacker to incorrectly setup control plane policies. To prevent these 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 Networking, we also highlight three main required features: scalability, responsiveness, availability, and reliability [18]. A step forward for providing these features is to solve the challenge of placing controllers in the network [8, 21, 1].Moreover, authentication, authorization and access control are essential primitives for Software Defined Networking. These primitives, together with the attestation and distribution of controllers, are the basis of a secure network, in which malicious components can be detected, identified and isolated [18, 22, 19].

3 AuthFlow Mechanism

The main idea of AuthFlow is performing authentication using Layer 2 protocols and mapping the identity of a host into the flows created by itself on

the network. The proposed mechanism applies the IEEE 802.1X standard and Extensible Authentication Protocol (EAP). EAP encapsulates authentication message exchanged between the supplicant host[3] and RADIUS authentication server. The translation of IEEE 802.1X messages into RADIUS packets is performed by the Authenticator. AuthFlow Authenticator is a software module that also communicates with our AuthFlow application running on top of the controller. The application allows or denies network traffic from a supplicant host depending on the result of authentication between the supplicant and authenticator.

AuthFlow applies the IEEE 802.1X because, while it specifies the authentication just above the MAC Layer, it is a widely adopted standard. IEEE 802.1X requires no changes on end hosts for network authentication, because IEEE 802.1X is trivially support by common Operating Systems. When a host, compatible with the IEEE 802.1X, 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 Ethernet type `0x888E`. Therefore, authentication procedure neither depends on any host's prior knowledge about the network, nor on a translation of an IP address into a MAC address. The AuthFlow authentication limits new host accesses just to the authentication system, which prevents a host of receiving a temporary IP only for authentication.

Following, we discuss the architecture of AuthFlow. We consider a use case in which AuthFlow authenticates virtual routers on a network infrastructure of a hybrid pluralistic Xen and OpenFlow virtualization platform [16,17,4]. Our proposal, however, is not limited to this use case and AuthFlow can be used, without any change, in the authentication of end hosts, switches or routers on an OpenFlow network. In the considered use case, supplicant hosts, that compose the OpenFlow network, are virtual machines that behave either as end hosts or as routers.

In our considered use case, the architecture of AuthFlow consists of physical machines running OpenFlow 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 performed by a software switch compatible with OpenFlow API, the Open vSwitch. The adopted virtualization model is the hybrid Xen and OpenFlow model used in Future Internet Testbed with Security (FITS) [17,16]. The POX controller runs our AuthFlow application that handles packet forwarding, in particular, packets[4] of the IEEE 802.1X. IEEE 802.1X packets are forwarded directly to the Authenticator. Authenticator is a RADIUS client that implements IEEE 802.1X and forwards the content of EAP messages to RADIUS. The authenticator was developed as an adapted version of `hostapd`[5], an Au-

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

[4] For the sake of generality, we call packets for datagrams of all layers.

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

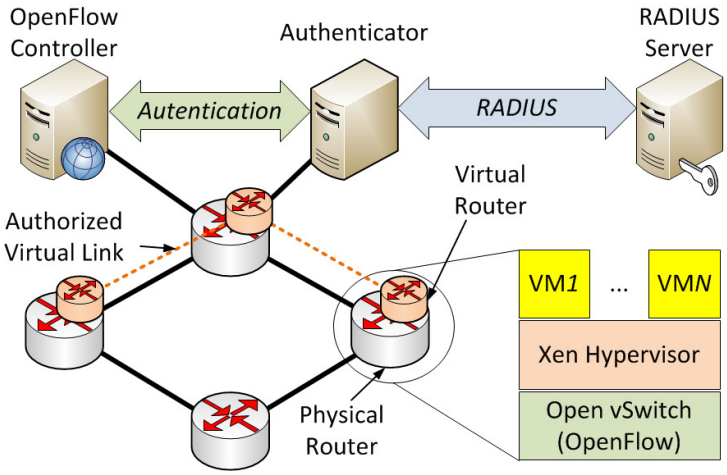


Fig. 1 The architecture of AuthFlow is composed of three main nodes: OpenFlow controller, the Authenticator and the RADIUS server. OpenFlow controller runs the AuthFlow application. To be authenticated on the network, a virtual machine starts EAP authentication according to IEE 802.1X standard. Authenticator decapsulates authentication messages from EAP packets, performs authentication of the virtual machine against RADIUS server, and informs OpenFlow controller of the result.

thenticator that is originally used to provide wireless networks. The `hostapd` was modified to inform our AuthFlow application, on top of POX, about the authentication of virtual networks. Our `hostapd` sends a confirmation message of authentication success for POX over a secure, encrypted and authenticated channel using SSL 3.0 (Secure Socket Layer) and Public Key Infrastructure (PKI). The authentication server is a RADIUS server that extracts the information encapsulated into EAP 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 MS-CHAP v2 (Microsoft Challenge-Handshake Authentication Protocol), which authenticates virtual router against a database using username and password as credentials. We deployed a Lightweight Directory Access Protocol (LDAP) database to store username and password pairs for each router. LDAP is also able to store other parameters that can define the privileges of the router on the network access.

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. 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 the authentication procedure succeeds, the Authenticator sends a `success` message for Supplicant host and sends an authorization and confirmation message for POX through a SSL channel. This message identifies the Supplicant by its MAC address, confirms the success of the authentication, and also informs the identity of the Supplicant host. After authentication,

our AuthFlow application running on top of POX allows the Supplicant host to access network resources. In case of revocation of the authentication of the Supplicant host, the Authenticator communicates AuthFlow application, which immediately denies the host access to the network, erasing and blocking all flow entries from and to the banned host.

AuthFlow application controls access of end hosts by denying or allowing virtual machines to access virtual links and services. The main idea is to block or to allow the creation of new flow entries that uses the virtual links. Thus, when starting an OpenFlow network that employs AuthFlow, all links are initially denied for any communication, including links that interconnect OpenFlow switches, i.e., links in the network core. Hence, 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`, an arbitrary number that is only used once, to avoid replay or spoofing attacks. This procedure is only possible because switches are assumed to be trustworthy entities; they are already authenticated into the controller through the OpenFlow secure channel, i.e., the SSL tunnel that is established between controller and all switches on the network. The trust of the switches is also assumed because switches run only the control measures deployed by the controller. In case of switches run other control measures than which controller deploys, it is not possible to assume the trustworthy of all switches.

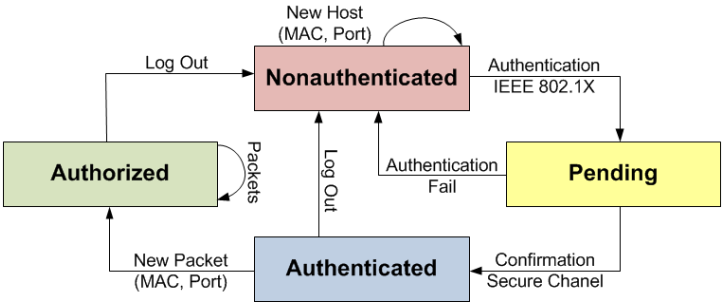


Fig. 2 AuthFlow access control is a four-state mechanism. i) Nonauthenticated, hosts that have not yet initiated the authentication process; ii) Pending, while the authentication process occurs; iii) Authenticated, when the host has already successfully completed authentication; iv) Authorized; when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

Figure 2 presents 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` defines the port of the

ingress switch where the host is connected. A host joining the network is initially in `Nonauthenticated` state and all traffic generated or addressed to it is dropped, except for Ethernet traffic with type `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 dropped, but host is awaiting the confirmation from the Authenticator to POX of the success of its authentication and what were the used credentials. As authentication is successfully confirmed, POX moves the host to the `Authenticated` state. In this state, our AuthFlow application on top of POX releases access to the network resources that the host is allowed to, according to its identity. Thus, when there is traffic to the host, POX checks whether traffic is in accordance with the policies related to the host credential. 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.

It is noteworthy that the tuple `(MAC, port)` is defined on the ingress switch of host in the network. Therefore, the tuple authenticates a host to access the network by a single port on a defined switch. Besides, controller authenticates each flow-entry on every switch because of the global network view, as the controller knows which host is authenticated on each port and on which switch.

Considering the proposed access control, allowing 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 a flow of a given host and its identity. If a flow has 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 the host are assigned to this flow. The forwarding decision about this flow takes also into consideration the credential of the host. Thus, access control is fine-grained as it is done according to the credential of the flow and not only according to the OpenFlow fields. The process is similar to flows which have destination MAC address (`dl_dst`) and output port (`output`) equal to those in the authentication tuple. AuthFlow access control policies define both outgoing and incoming rules to end hosts according to their identities.

4 Experimental Results

AuthFlow prototype is deployed on an island of the Future Internet Testbed with Security (FITS) [17]. The prototype runs Xen hypervisor 4.1.4 and OpenFlow network is realized by a software switch Open vSwitch 1.2.2. Open

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

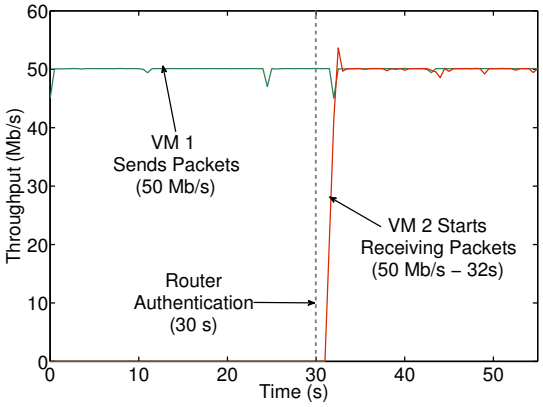
vSwitch is set to be controlled by POX[7]. The AuthFlow application for forwarding and for access controlling was developed in Python and runs on top of POX. Our prototype employs a modified version of `hostapd` as Authenticator that creates a secure channel from the Authenticator to POX to inform the controller whether there is a new authentication, or a loss of authentication of an end host. AuthFlow prototype adopts FreeRADIUS v2.1.12 as its RADIUS server. As proof of concept, the authentication method of the prototype was EAP-MS-CHAP v2, in which `(username, password)` is checked against a LDAP database.

We evaluated the performance of the prototype with `Iperf`, `nmap`, and `tcpdump` tools. For the sake of simplicity, but without losing generality, four personal computers running AuthFlow compose our experimental scenario, and each personal computer hosts a virtual machine that acts as a router or as a host in a virtual network. All computers are 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 set to operate at 100 Mb/s, to homogenize the experimental setup. 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).

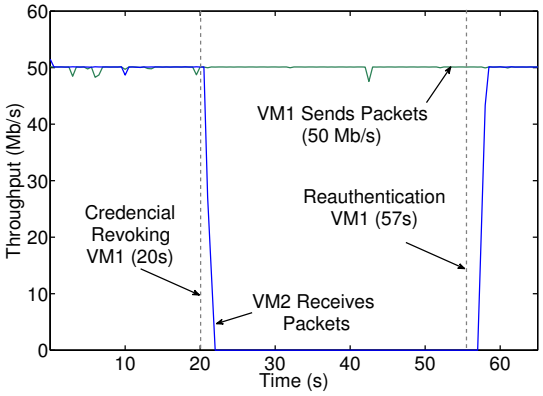
The first experiment evaluates the effectiveness of AuthFlow into dropping unauthorized traffic. The experimental scenario is simple: Virtual Machine 1 (VM1) sends packets to Virtual Machine 2 (VM2), and a virtual router between VM1 and VM2 forwards the packets. It is assumed that Virtual Machines 1 and 2 were previously authenticated and the virtual router is in `Nonauthenticated` state. 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 is authenticated, 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 till 2.5 s between starting router authentication and the effective releasing of network access. This delay is due to mainly the IEEE 802.1X authentication process plus an insignificant time of definition of new OpenFlow flow entry. This delay only occurs when a node joins the network. After the host have been authenticated, any new flow will be delayed just by the time of a new flow installation, which is up to 30 ms [16]. The following packets match on an already installed flow, and, thus, have no extra delay due to OpenFlow operation.

The second experiment shows the effectiveness of AuthFlow 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, and VM2 access to the network is blocked for both sending

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



(a) After 30 s, the virtual router between Virtual Machines 1 and 2 (VM1 and VM2) is authenticated and, thus, traffic flows from VM1 to VM2.

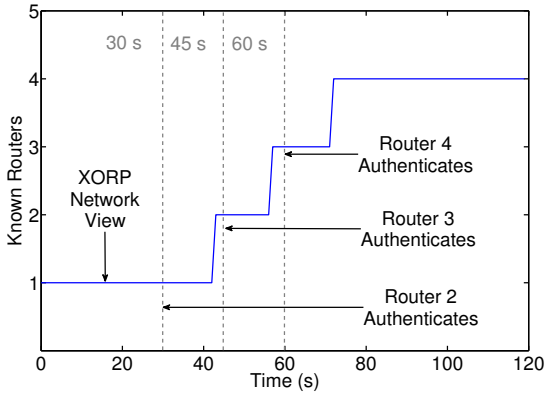


(b) At 20 s, VM2 authentication is revoked and, thus, network traffic is blocked. At the end of the experiment, the authentication is restored and the traffic of VM2 is allowed again.

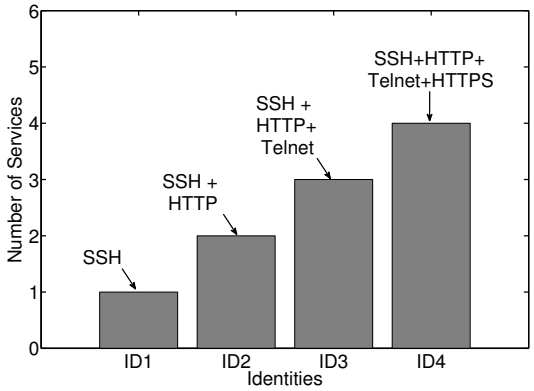
Fig. 3 Traffic is dropped before authentication and when credentials are revoked. Virtual Machine 1 (VM1) sends packets to the Virtual Machine 2 (VM2). (a) Authentication of virtual router between VM1 and VM2. (b) Revocation of VM1 authentication.

and receiving packets. The delay of blocking the network traffic of VM2 is less than 1 s. After 57 s, VM2 authentication is restored and VM2 restarts to receive packets. Restoring authentication delays approximately 2 s, as well as the authentication of a new host. AuthFlow benefits Software Defined Networking solutions for Denial of Service protection due to the unambiguous identification mechanism and fast responsiveness.

The next experiments show the network view from the perspective of authenticated hosts, i.e., which hosts and services an authenticated host reaches.



(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.

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

Figure 4(a) shows the network view of a virtual router running a link state routing protocol, OSPF (Open Shortest Path First). We consider a ring topology, connecting all four virtual routers. We verify the OSPF database of an already authenticated router each second and, then, identify how many neighboring the observed virtual router already knows. At the beginning of the experiment, the observed virtual router is the only one that is already authenticated. After 30 s, the second virtual router authenticates. At 45 s, the third, and finally at 60 s, the fourth router authenticates. 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. Each flood packet is matched against a rule to drop packets with the same headers for the next 5 s to avoid overloading the network.

The fourth experiment defines 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. The experiment consists of a supplicant host, authenticated against one of the four possible identities (ID1, ID2, ID3 and ID4), accessing a service provider. Each identity allows access to a set of services (one, two, three or four services, respectively). Therefore, the supplicant host performs a port scan (`nmap`) on the provider. The supplicant 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 supplicant host against another identity. Figure 4(b) shows the number of services that the supplicant host access on the service provider. The host can only access the services released to that identity with which it is authenticated. Port scan returns the "filtered" state for these ports that has no service allowed. It shows that blocking other services is performed by dropping `SYN` packets.

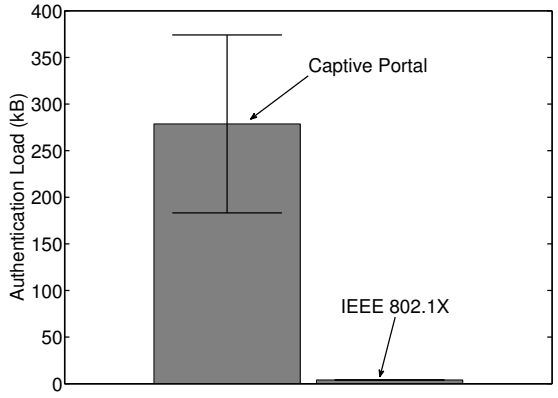


Fig. 5 Comparison between the authentication overload of a captive portal and the AuthFlow authentication through the IEEE 801.X standard.

The last experiment compares the authentication overload on the host side between two authentication mechanisms: A captive portal, and the IEEE 802.1X used on AuthFlow. The experiment runs ten authentication on the network, both on captive portal and on the IEEE 802.1X. The results are shown as average of the ten runs and with a 95% of confidence interval. Figure 5 shows that a simple captive portal web page (`Captive Portal`)[8], as proposed by Resonance [18] and Ethane [3], may introduce up to 300 kB of authentication

[8] For the sake of simplicity, we evaluate a standard page of NoCat captive portal. Available at http://nocat.net.

overload. The large variation of the overhead on the captive portal approach is due to cache on the web browser. The IEEE 802.1X (`IEEE 802.1X`), as proposed by AuthFlow, introduces a small overhead of 4 kB. Hence, AuthFlow authentication method introduces fewer overheads than other approaches of authentication on SDN.

5 Related Works

The security of Software Defined Networking, particularly the security of OpenFlow networks, is a subject fully debated currently. There are proposals for developing security applications on OpenFlow network infrastructure, as there are also others that seek to ensure the security of the infrastructure itself. Nevertheless, secure authentication, access control, scalability, responsiveness, availability and confidentiality are still challenging on SDN [11]. Main security proposals for SDN, however, focus on developing auxiliary system for enhancing security, instead of embedding security on the network infrastructure itself.

Kreutz *et al.* classify the main attack vectors on Software Defined Networking and present possible countermeasures to protect the network against these attacks [12]. Their work is, meanwhile, restricted to attacks on the resilience and on the trustworthy of the network.

FITS, Future Internet Testbed with Security, is based on a hybrid virtualization system over Xen and OpenFlow [17,2]. FITS ensures the confidentiality and availability of SDNs. The main idea of FITS 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 virtual network 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.

The UPV/EHU network [15], 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 difference between the UPV/EHU and AuthFlow is that the latter maps authentication credentials into the set of flows belonging to a host. Hence, at any time, it is possible to identify which host has generated or is receiving a flow on a switch.

Guenane *et al.* propose an authentication mechanism for virtual networks using EAP-TLS. The mechanism is implemented on smart cards [7]. 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. Nevertheless, Guenane *et al.* do not define how the authen-

tication and access control may be performed to authorize customer to access network resources.

Resonance [18] and Ethane [3] are other proposals to authenticate nodes in Software Defined Networking. 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 to have a browser installed to accesses web content. It 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. In AuthFlow, when a host joins the network, it starts its authentication according to IEEE 802.1X standard, just above the MAC Layer. It authenticates the MAC address only at 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 network addresses.

The proposals FortNOX [20] and FRESCO [22] define a set of security primitives for OpenFlow networks. FortNOX 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 other applications. FortNOX slices the network between applications on the same controller, which provides a finer control of privileges and better limits the control domain for each application than FlowVisor [10]. FlowVisor, in its turn, slices the network between multiple controllers; however, it does not provide a secure policy between controllers, to isolate actions of a controller from others. Besides the definition of a new set of primitives, FRESCO defines a modular language for the development of secure applications for OpenFlow networks. As AuthFlow provides a new mapping scheme of host credentials into the set of flows, these proposals can be extended to use AuthFlow to create a new primitive, the host authentication.

6 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 article, we proposed the AuthFlow, a mechanism for authentication and access control to the infrastructure of an OpenFlow Software Defined Networking. The mechanism authenticates hosts just above the MAC Layer using IEEE 802.1X standard and RADIUS authentication server. We implement the AuthFlow mechanism as a RADIUS authentication against a LDAP database, however, the proposal is extensible to other authentication methods, such as EAP-TLS which authen-

ticates 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. 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 credential of the 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.

References

1. Canini, M., Kuznetsov, P., Levin, D., Schmid, S.: A distributed and robust SDN control plane for transactional network updates. In: 2015 IEEE Conference on Computer Communications (INFOCOM), pp. 190–198 (2015). DOI 10.1109/INFOCOM.2015.7218382
2. Cardoso, L.P., Mattos, D.M.F., Ferraz, L.H.G., Duarte, O.C.M.B., Pujolle, G.: An efficient energy-aware mechanism for virtual machine migration. In: Global Information Infrastructure and Networking Symposium (GIIS'15), 2015, pp. 1–6. IEEE, Guadalajara, JA, Mexico (2015)
3. Casado, M., Freedman, M., Pettit, J., Luo, J., McKeown, N., Shenker, S.: Ethane: Taking control of the enterprise. ACM SIGCOMM Computer Communication Review **37**(4), 1–12 (2007)
4. Fernandes, N.C., Moreira, M.D.D., Moraes, I.M., Ferraz, L.H.G., Couto, R.S., Carvalho, H.E.T., Campista, M.E.M., Costa, L.H.M.K., Duarte, O.C.M.B.: Virtual networks: isolation, performance, and trends. "Ann. Telecommun." **66**(5-6), 339–355 (2011). DOI 10.1007/s12243-010-0208-9
5. Ferraz, L.H.G., Mattos, D.M.F., Duarte, O.C.M.B.: A two-phase multipathing scheme with genetic algorithm for data center network. In: 2014 IEEE Global Communications Conference (GLOBECOM). Austin, Texas, USA (2014)
6. Filasiak, R., Grzenda, M., Luckner, M., Zawistowski, P.: On the testing of network cyber threat detection methods on spam example. "Ann. Telecommun." **69**(7-8), 363–377 (2014). DOI 10.1007/s12243-013-0412-5
7. Guenane, F., Samet, N., Pujolle, G., Urien, P.: A strong authentication for virtual networks using EAP-TLS smart cards. In: Global Information Infrastructure and Networking Symposium (GIIS'12), 2012, pp. 1–6. IEEE (2012)
8. Heller, B., Sherwood, R., McKeown, N.: The controller placement problem. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12, pp. 7–12. ACM, New York, NY, USA (2012)
9. Hudson, D.L., Cohen, M.E.: Intelligent agents in home healthcare. "Ann. Telecommun." **65**(9-10), 593–600 (2010). DOI 10.1007/s12243-010-0170-6
10. Kobayashi, M., Seetharaman, S., Parulkar, G., Appenzeller, G., Little, J., van Reijendam, J., Weissmann, P., McKeown, N.: Maturing of OpenFlow and software-defined networking through deployments. Computer Networks **61**(0), 151 – 175 (2014). DOI http://dx.doi.org/10.1016/j.bjp.2013.10.011. Special issue on Future Internet Testbeds {Part I}
11. Kreutz, D., Ramos, F., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., Uhlig, S.: Software-defined networking: A comprehensive survey. Proceedings of the IEEE **103**(1), 14–76 (2015)
12. Kreutz, D., Ramos, F.M., Verissimo, P.: Towards secure and dependable software-defined networks. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN'13, pp. 55–60. ACM, New York, NY, USA (2013)

13. Levin, D., Wundsam, A., Heller, B., Handigol, N., Feldmann, A.: Logically centralized?: State distribution trade-offs in software defined networks. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12, pp. 1–6. ACM, New York, NY, USA (2012). DOI 10.1145/2342441.2342443
14. Lopez, M.E.A., Duarte, O.C.M.B.: Providing elasticity to intrusion detection systems in virtualized software defined networks. In: IEEE ICC 2015 - Communication and Information Systems Security Symposium (ICC'15 (11) CISS). London, United Kingdom (2015)
15. Matias, J., Jacob, E., Toledo, N., Astorga, J.: Towards neutrality in access networks: A NANDO deployment with OpenFlow. In: ACCESS 2011, The Second International Conference on Access Networks, pp. 7–12. Luxembourg City, Luxembourg (2011)
16. Mattos, D.M.F., Duarte, O.C.M.B.: XenFlow: Seamless migration primitive and quality of service for virtual networks. In: 2014 IEEE Global Communications Conference (GLOBECOM). Austin, Texas, USA (2014)
17. Moraes, I.M., Mattos, D.M.F., Ferraz, L.H.G., Campista, M.E.M., Rubinstein, M.G., Costa, L.H.M.K., de Amorim, M.D., Velloso, P.B., Duarte, O.C.M.B., Pujolle, G.: FITS: A flexible virtual network testbed architecture. Computer Networks **63**(0), 221 – 237 (2014). DOI http://dx.doi.org/10.1016/j.bjp.2014.01.002. Special issue on Future Internet Testbeds Part {II}
18. Nayak, A.K., Reimers, A., Feamster, N., Clark, R.: Resonance: Dynamic access control for enterprise networks. In: Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, WREN'09, pp. 11–18. ACM, New York, NY, USA (2009)
19. Piedrahita, A.F.M., Rueda, S., Mattos, D.M.F., Duarte, O.C.M.B.: FlowFence: A denial of service defense system for software defined networking. In: Global Information Infrastructure Symposium (GIIS'2015), 2015, pp. 1–6. IEEE, Guadalajara, JA, Mexico (2015)
20. Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., Gu, G.: A security enforcement kernel for OpenFlow networks. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12, pp. 121–126. ACM, New York, NY, USA (2012)
21. Ros, F.J., Ruiz, P.M.: On reliable controller placements in software-defined networks. Computer Communications pp. – (2015). DOI http://dx.doi.org/10.1016/j.comcom.2015.09.008. To be published
22. Shin, S., Porras, P., Yegneswaran, V., Fong, M., Gu, G., Tyson, M.: FRESCO: Modular composable security services for software-defined networks. In: Proceedings of Network and Distributed Security Symposium (2013)
23. Villain, B., Ridoux, J., Rotrou, J., Pujolle, G.: Mutualized OpenFlow architecture for network access management. In: 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), pp. 413–419 (2014)