Securing Distributed Control of Software Defined Networks

Othman OTHMAN M.M[†] and Koji OKAMURA[††],

[†]Department of Advanced Information Technology, Graduate school of Information Science and Electrical Engineering, Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan.

[††]Research Institute for Information Technology, Kyushu University. 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan.

Summary

This paper shows the method designed to secure and authenticate distributed behavior of the proposed hybrid control model of Software Defined Networks (SDNs) in [1]. Where, the SDN is an emerging topic that tracts attention due to its paradigm, that splits the control plane form data forwarding plane. According to [2], SDN defines OpenFlow [3] as "*key enabler for software-defined networks*". However, there have been some debates regarding the scalability of the OpenFlow's controller; this is due to the design of OpenFlow as it depends on a centralized controller to control flows. Manny efforts have been put to solve this issue, one of them is the hybrid control model proposed in [1]; where the original centralized model is preserved, while adding a distributed control model for some specific cases, in order to increase the network's efficiency. And based on the well-established fact, that all computer systems and networks should be secure; in this work, we propose a security method for such distributed control model of SDNs. In which, we aim to secure the flow installation using the distributed control, in addition to enabling safe usage of the distributed control without any chance of malicious use of the distributed control.

Key words:

Software Defined Networks, OpenFlow, Network Security, Control Model.

1. Introduction

Software Defined Networking (SDN) is an emerging topic that tracts attention due to its paradigm, that splits the control plane form data forwarding plane. According to which, the control plane is realized as the network operating system, which is responsible for controlling maintaining the state of the whole network. And the data plane is realized as the network equipment or devices that carry out instructions form the control plane and forwards the data packets. Where, SDN in [2] have defined OpenFlow as "key enabler for software-defined networks and currently is the only standardized SDN protocol that allows direct manipulation of the forwarding plane of network devices". Thanks to the flexibility provided by OpenFlow and SDN, many researchers embarked on providing new smart applications like; a virtualized network infrastructure in [4], detection of DDoS attack detection [5], measurement-aware routing [6], supporting QoS [7], run-time programming for network to support big data applications [8], and many others. It is believed that large number of new applications will be proposed to enhance the operation of current technologies and to provide even new applications.

On 2008, OpenFlow [3] was first introduced. OpenFlow is a part of Stanford University's clean slate project. OpenFlow provides a specially designed way to control flows on the network equipment by the OpenFlow controller (control plane) through using the OpenFlow Protocol, and splits that form the data plane (network equipment). According to OpenFlow; decision making can be done and modified freely by the OpenFlow controller according to layer 2, 3, VLAN, and layer 4 headers while the forwarding or routing is still done by routers or switches, in addition to, their original functionality. Moreover, OpenFlow defines actions to be performed on flows that can be either collection of statistics or usage data, forwarding packets, dropping packets, or manipulating packet's headers. This freedom, flexibility -due to the split of decision making and forwarding-, and the wide range of actions performed on packets enables OpenFlow to play a crucial role in developing the future Internet along with its main target which is running researchers' experiments on production networks.

However, despite this great flexibility of OpenFlow, there have been many concerns about the scalability of OpenFlow due to the way that the OpenFlow controller controls the OpenFlow network equipment, which forces a tight coupling between the controller and the network equipment. This would mean that the controller can be one of the bottlenecks in the system. There have been many efforts to solve this problem, as in [9], which aims provide a distributed event-based control plane for OpenFlow. Among those efforts, in our previous work [1], we have proposed a hybrid control model; that allows the regular centralized control model to be used as the main control model, in addition to allowing the distributed control model to be used in cases as a fail-safe mechanism. For example, this hybrid control model can be useful in cases where the controller is under heavy loads and is required

Manuscript received September 5, 2013
Manuscript revised September 20, 2013

to install a large number of flow entries into the OpenFlow network equipment, while some network equipment are also heavily loaded (load on network equipment can be thought of in more than one prospective, e.g. the usage of flow table entries). In such cases, the hybrid control model can be used to relieve the controller form doing any further processing to relocate the flows, and enabling it to install those flows as they are; and relying on the distributed control of the network equipment to solve any issues of network equipment overloading. And thus, the hybrid control model enables the controller to work with more ease in cases of overloading.

Moreover, providing security for the hybrid control model of SDNs is quite important; based on the well-established fact, and the lessons learned by designing routing without enough emphasis on security, and the threats it exposed the network to; leading to designing of methods to secure those routing protocols. And thus, in this paper we propose methods to secure the distributed control model of the SDNs, in order to provide a secure method to transfer flows without exposing the network to any threat or exposing any information related to the operation of the network to any eavesdropper. Where this target; of securing the distributed control, goes along with the design of the original centralized control that is secured by means of Transport Layer Security (TLS [10]).

The remainder of this paper is organized as follows. We first introduce a brief overview about the distributed behavior of SDN, along with the design of the protocol of distributed control of SDN in Section 2. Next in Section 3, the main principles governing the security methods are described; this includes the threat model under which the methods must work is explained in Subsection 3.1, while the requirements of the proposed security methods in Subsection 3.2. We then explain the design of the security methods of the distributed behavior of SDN in Section 4. Furthermore, Section 5 shows how the proposed security methods are effective against major classes of attacks. We then discuss about the evaluation of the proposed methods in Section 6, and finally conclude in Section 7.

2. Distributed Control Behavior of SDN Reviewed

According to the design of current SDN's leading technology, OpenFlow [11]; flows can be programmed (installed) by the controller. This means that the controller is the only entity that is responsible for installing and maintaining flows on the network equipment. For simplification let's call this type of flow installation the "controller to equipment flow installation". The controller to equipment flow installation has many advantages like having tight control over all of the equipment by the controller.

However, the advantages of the controller to equipment flow installation come with some cost. First is the probability that the controller would be a source of bottle neck in the whole system. This can be confirmed by, Michael Jarschel et al. who concluded in [12] that "*When using OpenFlow in high speed networks with 10 Gbps links, today's controller implementations are not able to handle the huge number of new flows.*". Second, by limiting the flow manipulation to "*the controller to equipment*" installation method OpenFlow can miss some opportunities that the "*network equipment to network equipment*" can provide.

For the previously mentioned reasons, we proposed in a previous work [1], a new method for installing flows, that is, the "network equipment to equipment flow installation" (Ne-NeFI) method. Through using this method, the controller does not have to program (install) flows to each one of network equipment one by one; instead it can ask the equipment to spread this flow to other equipment on behalf of the controller, this can be useful in cases where the controller needs to program non critical-start up time flows. And thus relieving some load off the controller. Also, the network equipment to equipment flow installation method can be used to make the OpenFlow network more self-aware by having the network equipment cooperate and carry loads for each other upon the need and traffic situation by having the overloaded equipment delegating some of its flows to another network equipment.

2.1. Protocol of the Distributed Control of SDNs

In order to enable distributed control of SDNs, represented by the network equipment to equipment flow installation to be adopted to the OpenFlow Protocol, three new packets have to be introduced (see Fig. 1). First one is, equipment to equipment (e-e) flow installation request, abbreviated as "e-e request". While the second is; the e-e flow installation reply, abbreviated as "e-e reply". The third is the e-e flow installation acknowledgement or negative acknowledgement, abbreviated as "e-e ACK/NACK".

The first packet is the e-e flow installation request. This packet holds an OpenFlow header, list of flows to be programmed, address of the equipment that sent the request and an identification value, address and identification of the originator of the request (who requested for the flows to be programmed in the first place, i.e. controller or an equipment), Level of Flow Installation, the Time To Live (TTL) of the IP protocol, and a temporary identifier for this request. Where, the Level of Flow Installation (LFI) is somewhat similar to the TTL field in the IP protocol. Where, the TTL in IP protocol indicates how many hops a packet can travel, and it will be decremented when passing a network equipment and the packet will be discarded when the value reaches 0. While, the LFI, it is decremented each time an equipment relays the e-e request (sends further the broadcast). And thus it is very similar to the TTL value except that the LFI is part of the proposed protocol of the distributed control of the SDNs, while TTL is part of the underlying IP protocol. LFI and TTL are used together to control the propagation of the e-e flow installation request.

The second packet is the e-e flow installation reply. It contains an OpenFlow protocol header, the identification of the e-e flow installation request, the reply to the request which can be either an acceptance or a rejection, address and self-identification of the equipment that sent the reply.

And the third packet is the e-e Acknowledgement or the e-e Negative Acknowledgement (e-e ACK/NACK). Its purpose is clear; to confirm to the equipment that sent the reply that its reply has been received, and accepted or rejected.

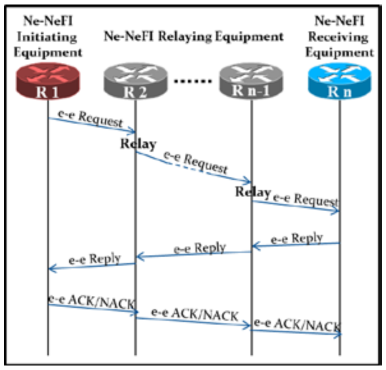

Fig. 1. Distributed SDN control protocol packets.

3. Security Principles of SDN's Distributed Control Behavior

This Section describes in Subsection 3.2 the goals and requirements that the proposed distributed control behavior security method must provide in order to enable a safe use of the distributed control behavior for SDNs. And since the proposed security method aims to provide robust security, it must be able to secure the distributed control behavior under the threat model described in Subsection 3.1.

3.1.Threat Model

The proposed distributed control security methods seeks to provide robust protection against both insider threats (authenticated network equipment), and outsider threats (end hosts or an unauthenticated network equipment). We also assume that an attacker might be able to use different points within the same network to charge his attack.

For the case of outsider attacks, they are prevented from initiating any attack to the SDN network, because they are not registered in the Trust Manager (refer to Subsection 4.1) and thus any attempt to send distributed control will be blocked and their packets dropped. While for the case of the insider attacks, further details are provided in Section 5 on how the proposed methods disables them.

3.2. Security Requirements/Goals

The main goals and requirements of the proposed security methods are as follows:

- Allowing the transfer of flow table entries form one network equipment to another, in a way that prevents any malicious user form obtaining any information related to that flow entry or disclosing its contents. And thus preventing any malicious user from obtaining any knowledge about the network or its operation or control.
- Enabling a smooth operation of the distributed control of SDN. This requires, the security methods to be able to protect the distributed control's protocol, so that no attack could be charged to jeopardize the operation of the SDN's distributed behavior.
- Protecting the whole SDN network from any attack that might use the distributed control to affect the normal operation of the SDN. The importance of this requirement is obvious, since the original design of the centralized (central controller to any equipment) according to OpenFlow [11] is secured by using Transport Layer Security (TLS) [10]. And thus, any propose to extend the centralized control model must be able to maintain the security of the whole network.

4. Design of the Security Method

In order to designing a successful security method for the distributed control of the SDN, special care must be taken enable the designed algorithm to achieve the security requirements (described in Subsection 3.2) while being able to operate under the threat model described in Subsection 3.1.

To shed more light on the design of our proposed security method for the distributed control of SDNs; Subsection 4.1 shows the main components of the security method. While Subsection 4.3 describes in details the algorithms used to compose the desired security scheme. Finally, Subsection 4.4 shows the how the proposed security method can be applied, using a scenario based example.

4.1. Building Blocks

The main building blocks that the security method of the distributed behavior of SDN, are explained as follows:

1) Trust Manager

   Is the entity; that is responsible for assuring a secure binding of the unique ID of each network equipment with the public key of that network equipment. Along with the binding, the second responsibility of the trust manager is to periodically distribute a certificate list to all of the network equipment in its domain. Where this certificate list; contains the digital certificates of all of the equipment in the domain of the trust manager. The third responsibility of the trust manager is to manage the list of trusted network equipment within its domain by listening to threat warning reports sent by the network equipment within its domain in case of a suspected attack or a confirmed one (as will be explained in the next Subsection). And then, responding to the threat warnings by suspending the certificate of the attacker network equipment, and sending an updated certificate list – that does not contain the certificate of that attacker – to the network equipment within its domain.

   Through our design of the proposed security method, we assume that the public key signature algorithms used in the trust manager and the network equipment are secure and no malicious user is able to fraud a valid signature nor he is able to recover the secret key of any component of the scheme. We also assume that the communication between the network equipment and the trust manager is also secure.

2) Network Equipment

   Network equipment are same as the regular SDN equipment that are either routers or switches that supports OpenFlow or any other SDN technology. And in addition to their regular tasks those network equipment have to perform additional operations in order to use a secure distributed control of SDNs. Those additional operations are; digitally signing e-e requests, verifying digital signature of the distributed control, receive and store the certificate list distributed by the trust manager, and reporting any threats (activity or message exchange the is expected to be of a malicious attacker) to the trust manager.

4.2. Proposed Security Mechanism

This Subsection shows the conditions forming the main methods of security. And thus, they are the main essence of how attacks are prevented:

1) Using Trust Manager: it will send and update a list of trusted devices. e.g. list of (device ID, public key).
2) Using start time and end time for each e-e request.
3) Having every device that relays or receives the e-e request broadcast to check the signature of device that originated the e-e request, and thus make sure that the e-e request received is exactly same as it left the device that originated it.
4) Having every device that relays the e-e request broadcast, to check the signature of the previous device that relayed the e-e request, knowing that it must be a direct neighbor.
5) Having every device that relays the broadcast to sign the e-e request.
6) The flow table entries will be encrypted when they are transferred form one network equipment to the other, so that no eavesdropper can expose their contents.
7) Each network equipment will hold counters, one for each network equipment in the certificate list. Where this counter counts the number of e-e requests that does not pass the conditions of the main methods of security shown in this chapter.

4.3.Algorithms

Section 4.1 explained about the main components of the security method of the distributed control of SDNs. While, this Section; explains the details of the algorithms of the security method, and discusses their steps in details.

We first start by showing the list of variable and primitive functions used in the algorithms of the security method in Table 1. Then, we start by explaining the details of the algorithm used by the originator of the e-e request to create and send the request, as shown in

SEND_REQUEST algorithm. After that, we will explain the details of the algorithms used for by the network equipment that receives the e-e request; that are: RECEIVE_BROADCAST algorithm, which calls the CHECK_INCOMING_REQUEST algorithm and the RELAY_BROADCAST algorithm. After that, the later algorism will be explained respectively. Finally, we will explain the RECEIVE_REPLY algorithm, which shows the steps taken by the originator of the e-e request upon receiving a reply to his request.

Table 1. List of Notations used in the algorithms following in this paper (variables, and operations).

Notation	***Description***
D_i	ID of Device i.
Sig_i	Signature of Device i for the corresponding packet.
sigList	List of pairs of (Device ID and its signature of that packet), for all the devices that the packet traversed along its path.
Req	e-e request that includes the LFI, start time, end time.
create_req ()	Create a request based on the current needs of this network equipment.
create_signature_list ()	Create an empty signature list (sigList).
can_Accept_Request (req)	Indicates whether the device can accept the e-e request or not.
Rep	e-e reply.
create_Reply (req)	Create the proper e-e reply for the in hand e-e request.
sign(PrK , DATA)	Create digital signature for DATA using the private key PrK.
send (DST, $Data_0$, … , $Data_n$)	Send one or more pieces of data, to the destination address represented in DST.
get_Src (packet)	Get the source address of the packet in hand.
is_Not_Neighbor(D_i)	Check if device D_i is a neighbor or not.
is_Not_In_Device_List (D_i)	Check if D_i is found in sigList or not.
drop (req)	Drop or discard the e-e request in hand.
check_Signature_If_Not_Valid (Data , (D_i ,Sig_i))	Check if the digital signature Sig_i is a valid signature of device D_i for the data represented in Data argument.
check_Signature_If_Not_Valid (Data , Sig_i)	Same as the previous bur D_i is that of the sender of Data.
drop_Counter_per_Period(D_i)	Counter of the Dropped packets that came from Device i in the current period.
Tolerate_Limit	The number of redirection requests per period, which can be accepted per device.
report_To_Trust_Manager(D_i)	Sending a threat warning to the trust manager.
Append ($Data_1$, $Data_2$, … , $Data_n$)	Append pieces of data to gather to produce a single piece of data. In this case append $Data_1$ through $Data_n$.
Append (sigList , (D_i , Sig_i))	Append the pair (D_i, Sig_i) to the sigList in hand.
Broadcast (req, sigList)	Broadcast the e-e request along with the sigList to all of the ports except the egress port.
originalReq	Cashed copy of the original e-e request that was sent by this equipment earlier.
is_req_satisfied (req)	Check if the request represented by req has been served before or not
create_neg_ack (req)	Creates a negative acknowledgement for the request in hand (req)
create_ack (req)	Creates a positive acknowledgement for the request in hand (req)

```
1:  Function  SEND_REQUEST ()

2:      Req = create_req ()
3:      self_sign = sign (private$_{self}$ , Req)
4:      sigList = create_signature_list ()
5:      Append ( sigList , (D$_{self}$ , self_sign) )
6:      Broadcast (Req, sigList)
```

Fig. 1. SEND_REQUEST Algorithm.

The SEND_REQUEST algorithm (shown in Fig. 1), is a simple algorithm, which is followed by the originator of the e-e request, to create the e-e request, as shown in line 2. After that, the originator will digitally sign the e-e request and add that as the first signature to the sign list of the e-e request, shown in lines 3 through 5. The Final step in this algorithm is for the originator to broadcast the e-e request, as shown in line 6.

```
1:  Function  RECEIVE_BROADCAST (req, sigList)
    // req = the e-e request
    // sigList = {(D$_1$ ,Sig$_1$), …., (D$_n$ ,Sig$_n$)}

2:      CHECK_INCOMING_REQUEST (req, sigList)
3:      if (can_Accept_Request (req))
4:          rep = create_Reply (req)
            // rep = e-e reply
5:          repSig = sign(private$_{self}$ , rep)
6:          reqSig = sign (private$_{self}$ , req)
7:          rep = Append (rep, repSig, reqSig)
8:          send (get_Src (req) , rep)
9:      end if

10:     decrement (LFI)
11:     if (LFI > 0)
12:         RELAY_BROADCAST (req, sigList)
13:     end if
```

Fig. 2. RECEIVE_BROADCAST Algorithm.

The main algorithm for handling the security method within the network equipment receiving e-e request is the RECEIVE_BROADCAST algorithm (shown in Fig. 2). This algorithm is called after receiving a e-e request, and it starts by checking if the incoming e-e request matches point 2) through 5) of the methods show in Subsection 4.2 by calling in line 2 the CHECK_INCOMING_REQUEST (shown in Fig. 3, and explained next). After that, if the received e-e request is found to be correct then the algorithm will check in lines 3 to 9, if equipment in which it runs is capable of serving this e-e request (that is; the equipment is capable to receive new flow table entries form the sender of the e-e request) . And if it can receive the new flow table entries then the algorithm will generate an e-e reply (line 4) that will be signed (line 5). Also, if the equipment is willing to accept the request it has to generate a signature of the received e-e request (line 6); so that the originator of the e-e request can verify that the sender of the reply did receive the original e-e request that the originator did send. The final step of accepting to serve an e-e request is to send the e-e reply, its signature, and the signature of the e-e request; to the originator of the e-e reply (lines 7, 8). And in case of the network equipment is not capable of serving the e-e reply by itself then it will decrement the lifetime (LFI) of the e-e reply by one, sign it, and broadcast it again as shown through lines 8 to 11.

```
1:   Function CHECK_INCOMING_REQUEST (req, sigList)
     // req = the e-e request
     // sigList = {(D_1 ,Sig_1), …., (D_n ,Sig_n)}

2:       if (is_Not_Neighbor(D_n) OR
         is_Not_In_Device_List (D_n) )
3:           drop (req)
4:       end if

5:       if (check_Signature_If_Not_Valid (req, (D_1 ,Sig_1) ) )
6:           drop (req)
7:       end if

8:       if (check_Signature_If_Not_Valid (req, (D_n ,Sig_n) ) )
9:           drop (req)
10:          increment ( drop_Counter_per_Period (D_n ) )
11:          if (drop_Counter_per_Period (D_n ) > Tolerate_Limit)
12:              report_To_Trust_Manager (D_n )
13:          end if
14:      end if
```

Fig. 3. CHECK_INCOMING_REQUEST Algorithm.

After explaining about the main algorithm related to receiving the e-e request that is the RECEIVE_BROADCAST. We will continue to explain one of the algorithms that the main algorithm calls; that is the CHECK_INCOMING_REQUEST algorithm as shown in Fig. 3. Where, in this algorithm, it first starts in line 2 to check if it received the e-e request from a direct neighbor, and if it received the e-e request form a network equipment that was included in the certificate list – that was received from the trust manager – and in case that any of those conditions is not satisfied then the packet will be dropped as shown in line 3. After the e-e request passes the test in line 2, the validity will be tested of both the signature it of its originator (line 5), and its last signature in its signature list (line 8). If both are correct, then the CHECK_INCOMING_REQUEST algorithm finishes. Elsewise, if the signature of the originator of the e-e request is found to be invalid (line 6), then the request will be dropped. Same thing will happen in case that the last signature in the signature list (Sig_n of sigList) is found to be invalid (line 9). Furthermore, in that case, the counter of the dropped e-e requests related to the network equipment that broadcasted the false e-e request will be incremented (line 10) (as explained in point 7) of Subsection 4.2). And in case that this counter increases over a predefined threshold that is the Tolerate_Limit (line 11), the algorithm will generate a threat warning report and send that to the trust manager (line 12). Where in its turn the trust manager will count for a number of threat warning reports before removing the network equipment form the certificate list and thus preventing the suspected malicious equipment form doing further malicious activities. It should be clarified here that choosing the value of the Tolerate_Limit, or the threshold upon which the trust manager removes an equipment form the certificate list; is out of the scope of this paper and it is left up to the operator to decide.

```
1:   Function RELAY_BROADCAST (req, sigList)
     // req = the e-e request
     // sigList = {(D_1 ,Sig_1), …., (D_n ,Sig_n)}

2:       self_sign = sign (private_self , req)
3:       Append ( sigList , (D_self , self_sign) )
4:       Broadcast (req, sigList)
```

Fig. 4. RELAY_BROADCAST Algorithm.

The second algorithm called by RECEIVE_BROADCAST is the RELAY_BROADCAST shown in Fig. 4. The RELAY_BROADCAST algorithm is relatively simple. It starts by signing the e-e request using the current network equipment's public key. After that the signature is appended to the signature list (sigList) in the e-e request, and finally the e-e request is broadcasted again.

Finally, the RECEIVE_REPLY algorithm, shown in Fig. 1; explains the steps followed by the originator of the e-e request upon receiving an e-e reply to his request. It starts by checking the validity of both the signature of the e-e reply and the signature of its corresponding e-e request, if they were signed by the sender of the e-e reply or not (line 2). If any of those two signatures fails, then the received reply will be dropped (line 3). In case the two signatures are valid, the next step will be to check if the e-e request has been already satisfied or not. If it was satisfied (lines 6 to 11); then a negative acknowledgement (NACK) will be sent, after signing it to the sender of the reply. On the other hand, if the request has not been satisfied (lines 12 to 18);

then a positive acknowledgement (ACK) will be sent to the e-e reply sender, after singing it.

```
Function  RECEIVE_REPLY (rep, repSig, reqSig)
// rep = the e-e reply
// repSig = the signature of the reply, by its sender.
// reqSig =  the signature of the received request, by the sender
          of the reply.

    if (check_Signature_If_Not_Valid (rep, repSig ) OR
    check_Signature_If_Not_Valid (originalReq, reqSig ) )
        drop (rep)
    end if
    else
    // if both signatures are verified correctly
        if ( is_req_satisfied (originalReq) )
            NACK = create_neg_ack(rep)
            nackSig = sign (private_self , NACK)
            NACK = Append (NACK, nackSig)
            send (get_Src (rep) , NACK)
        end if
        else
        // the request has not been accepted before
            ACK = create_ack(rep)
            ackSig = sign (private_self , ACK)
            ACK = Append (ACK, ackSig)
            send (get_Src (rep) , ACK)

        end else
    end else
```

Fig. 1. RECEIVE_REPLY Algorithm.

4.4.Usage Scenario

To further explain how the proposed security method works, the following scenario follows the steps of the distributed control along with the proposed security method. This scenario is show as a series of steps shown by figures Fig. 2 through Fig. 5. In those figures, each router is identified by its color. The medal shape represents the digital signature, where the color of the medal's ribbon represents the router that signed it by having that router's color. And the red tick symbol represents either a verified digital signature, or verifying that the e-e request was sent by a direct neighboring network equipment. Also, in this scenario we assume that the certificate list has been already distributed to the whole network, where this is represented by showing the symbol of the certificate list over the whole network. While the numbers in circles represents the step number in the explanation of each figure.

The scenario first starts by having network equipment named R1 – shown in red color – initiating a distributed control behaviour that is the Ne-NeFI. This initiation shown in step 1 of Fig. 2, where R1 broadcasts a e-e request after signing it.

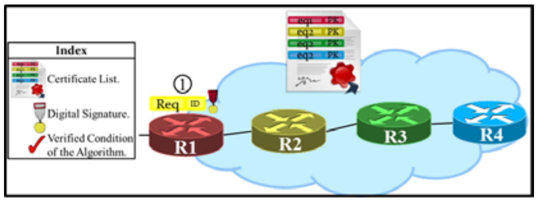

Fig. 2. Request sending.

After that, in Fig. 3, network equipment R2 receives the e-e request, checks that it was received from its neighbour, and that the signature is an authentic one of R1; as shown in step 1. However, for the sake of illustration, both of R2 and R3 are not capable of serving the e-e request. And Thus R2 will broadcast the e-e request after signing it. Next, in step 2, network eqiupment R3 will does the same steps followed by R2 in step 1, and will end broadcasting the e-e request.

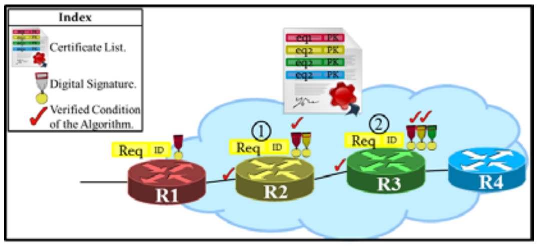

Fig. 3. Relaying Request.

Next, in Fig. 4 step 1, R4 receives the e-e request, and verifies that it was sent by a neigbour equipment, and that it was properly signed. Then, R4 desides to serve this e-e request. And thus, it will send an e-e reply after signing it to the initiator of the Ne-NeFI's e-e request, that is R1. As shown in step 2.

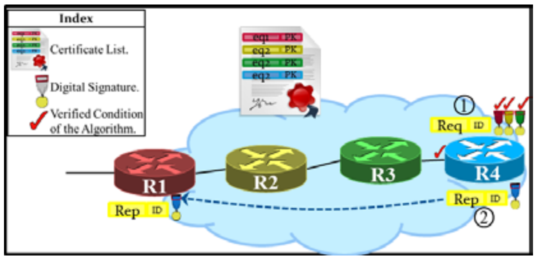

Fig. 4. Replying to request.

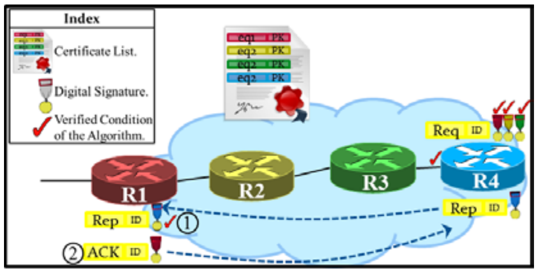

Fig. 5. Accepting reply.

Finally, Fig. 5 shows in step 1, that R1 – the initiator of the Ne-NeFI – receives the e-e reply sent by R4, and checks that it was properly signed by R4. After that, in step 2, R1 will send an e-e ACK to R4 after signing it, so that R4 can verify that the acknowledgement was sent by R1.

5. Attack Resistance

Based on the threat model presented in Subsection 3.1, and the requirements illustrated in Subsection 3.2; the proposed security methods are designed to provide robust security, resisting many security threats. In the rest of this Section we considered various classes of security threats and showed how the proposed security methods provide protection against them.

- **Man in the Middle Attacks**

  This class of attacks can occur in different variations. Firs, is the case where an eavesdropper whether internal or external, captures legitimate packet from a distributed control exchange. In this case, the effect of such attack is very limited, since the attacker will not be able to alter any contents of the distributed control, since it is digitally signed. And based on the assumption that the public key signature algorithms are secure, then it will be inapplicable for the attacker to resign the distributed control packets since the attacker does not possess the private key of the previous equipment that have relayed the packet, nor that of the initiator.

  Furthermore, even if the attacker was an internal one and did legitimately sign the packet – because he was listed in the certificate list – after tampering with its original contents. This can be carried out only in the case of the e-e request being relayed, because in the case of e-e reply and e-e ACK this will not be possible since only the sender will sign; and thus, it will not be possible for the attacker to resign the packet. Similarly, in case of the e-e request, each network equipment that relays, or accepts the e-e request; will check the signature of the originator of the e-e request in addition to the last signature in the signature list (sigList). And so, if any malicious user tampers with the original e-e requst, this will cause the e-e request to be dropped. Thus, any tampering with the contents of the distributed control protocol will be detected, and renders any sbusequent attacks to be impossible.

- **Resource Exhaustion Attacks**

  This attack also can occur in different variations. The first one can be the case that an attacker being either an internal or an external eavesdropper will copy legitimate e-e request and resend them to a distant part of the network for the purpose of consuming the flow table entries of many equipment all over the network. However, according to the proposed security methods, such attacks are rendered impossible since that the algorithm for any relaying network equipment will make sure that the e-e request was received form a direct neighbor, if not then the e-e request will be discarded. And thus this type of resource exhaustion will not be possible.

  Another variation of the resource exhaustion attack is the simple case of having an internal equipment sending a large number of e-e requests for the purpose of exhausting the available space of the flow tables of other equipment within the network. In this case, according to the proposed security methods, neighbor network equipment will send threat warnings to the trust manager, who in turn will remove the malicious sender form the certificate list, and thus will stop the attack and prevent any further attacks to be done by that malicious equipment.

  While another more sophisticated variation, where an attacker might have control over one or more internal network equipment. In such case, an attacker can copy a legitimate e-e request, encapsulate it, and send it to another network equipment – under control of the attacker – in a more distant part of the network, in order to exhaust the resources of network equipment in that distant part of the network. In such attacks, the attacker can maintain the e-e request in its original form without tampering it or its signature; after that the attacker can legitimately add the signature of the distant attacker equipment to the signature list. Thus, if a network equipment receives the request form the distant attacker equipment; and the receiver, will find an authentic e-e request with its original signature, and will find that it received this e-e request by a direct neighbor. However, such attacks are made unfeasible, since, the e-e request will be more likely to be served/accepted by another network equipment that is nearer to the originator of the e-e request, and in this case the originator will send a negative acknowledgement to cancel the installation to the distant sender of the e-e reply. And thus such attacks are infeasible since they require big efforts of the attacker to charge such attacks, while their effect will be minor.

6. Discussion and Evaluation

In this section we discuss the efficiency of the proposed security methods, in terms of their ability to achieve their goals within a reasonable time frame.

Furthermore, it is important to mention that in the design we did not mention about the type of encryption and digital signature used in our methods. Because we believe our methods should be written in a generic way so that they can be implemented with any public key signature and encryption, that the implementer find suitable. However, there are many attractive public key signature algorithms that are very suitable to use in such security methods. For example the elliptic-curve signature in [13] provides a compact size of the public key, private key, the signature; that are respectively 64 bytes, 32 byes, and 64 bytes. Such compact signature size is very attractive, since it will add little extra size to e-e request. Moreover, this signature algorithm [13] can sign and verify data in short times. According to [14], on a widely known CPU – like Intel Core i5-4570S; 4 x 2900MHz – the algorithm of [13] can perform signature in 0.023644556 milliseconds, and verification in 0.071357913 milliseconds. While, for the public key encryption; there are many attractive algorithms. For example, the well-known RSA1024 (implemented in [15]) can achieve encryption in 0.015677039 milliseconds, and decryption in 0.444308778 milliseconds.

Quantifying the required number of digital signature, verification, encryption, and decryption required for the proposed security methods, will lead us to have better understanding and assessment of their efficiency. And thus, we denote to digital signature with **"s"**, signature verification with **"v"**, encryption with **"e"**, decryption with **"d"**, and the number of relaying network equipment with **"R"**. It can be calculated that the total time needed to complete a secure distributed control will be " **Total_time= (4s + 5v) + ((2v + s)*R)** " where the first part of the equation represents the static part of the distributed control; that is, 1 signature by the originator of the e-e request. And, 2 verifications of the request and 2 signatures of the e-e reply; for the equipment willing to server the request. Corresponding to the later, the originator will do 2 verifications for the e-e reply, and 1 signature of the acknowledgement. Finally the equipment accepting the request will verify the acknowledgement. And for the purpose of assessing the suitability of using the proposed security method, we consider the case of a data center, as explained in [16], where it is stated that flow installations should be handled within a period of 10 milliseconds. Within this time constraint, the proposed security method can be completed with about 57 relaying equipment on the path of the e-e request. And thus, a large number of equipment can be covered with 57 relays of the broadcast, which we think that it is very likely to that an equipment within this number will be willing to serve the request and thus achieving the target of the distributed behavior.

7. Conclusion

Providing future Internet with technologies that enable it to play its role is extremely important. Because of that, many researchers are studying technologies to be the future Internet enabling technologies. SDN is one of the candidate future Internet technologies, as it provides compelling functionalities that enable smarter applications to be built. However, there have been many concerns regarding its scalability; as well as of its key enabler OpenFlow, especially, due to its dependence on a central controller. And thus, many efforts were done to overcome this problem. One of them was proposed in [1], which proposed to do that by providing a hybrid control model that combines both the centralized control with some distributed control behavior. Following the well-established fact; that all sensitive computer networking operations must be secure, in this work we propose methods for securing the distributed control behavior of the SDNs, that was proposed in [1]. In order to get a fully secure hybrid control, since the centralized control is already secure by means of TLS.

In order to achieve the desired security for the distributed control, we designed security methods and algorithms. Where the proposed methods require; according to our design, a centralized trust manager to distribute a list of trusted equipment along with their public keys. In addition to the centralized trust manager, the network equipment must be able to perform digital signature, signature verification, and reporting any threat warnings to the trust manager. In more details, the equipment to originate and send the e-e request has to sign it. While each network equipment relying the e-e request must make sure that it received a verified e-e request from direct neighboring equipment, and the e-e request have not been tampered. And thus, we can make user that the genuine e-e request did traverse a trusted path. After that, if the e-e request have reached a network equipment that is willing to accept this request, it will send a signed reply along with a signature the e-e request that it received to the equipment that originated the e-e request. Thanks to which, the originator of the e-e request, upon receiving the e-e reply can make user that it came from an authentic equipment, and that equipment did receive the original genuine e-e request. Finally, the equipment that originated e-e request will reply to the e-e reply, by either an acknowledgement or a negative acknowledgement. And thus by following the previous steps it is possible to secure the distributed control of the hybrid control model of SDNs, thus enjoying the benefits of the hybrid control without jeopardizing the whole network.

References

[1] O. M. Othman and K. Okamura, "Hybrid Control Model for Flow-Based Networks," in the international conference COMPSAC 2013 - The First IEEE International Workshop on Future Internet Technologies, Kyoto, Japan, 2013.

[2] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, 2012.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: enabling innovation in campus networks," in ACM SIGCOMM Computer Communication Review, 2008.

[4] H. Shimonishi and S. Ishii, "Virtualized network infrastructure using OpenFlow," in Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP, 2010.

[5] R. Braga, E. Mota and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in Local Computer Networks (LCN), 2010 IEEE 35th Conference on, 2010.

[6] G. Huang, C. Chuah, S. Raza and S. Seetharaman, "Dynamic measurement-aware routing in practice," Network, IEEE, Vols. 25,3, 29-34.

[7] B. Sonkoly, A. Gulyas, F. Nemeth, J. Czentye, K. Kurucz, B. Novak and G. Vaszkun, "On QoS Support to Ofelia and OpenFlow," in Software Defined Networking (EWSDN), 2012 European Workshop on, 2012.

[8] G. Wang, T. Ng and A. Shaikh, "Programming your network at run-time for big data applications," in Proceedings of the first workshop on Hot topics in software defined networks, 2012.

[9] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in Proceedings of the 2010 internet network management conference on Research on enterprise networking, 2010.

[10] T. Dierks and E. Rescorla, "rfc5246: The Transport Layer Security (TLS) Protocol Version 1.2," The Internet Engineering Task Force, 2008.

[11] "OpenFlow Switch Specification, Version 1.1.0," 2011.

[12] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in Proceedings of the 23rd International Teletraffic Congress, 2011.

[13] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe and B.-Y. Yang, "High-speed high-security signatures," Journal of Cryptographic Engineering, vol. 2, no. 2, pp. 77-89, 2012.

[14] D. J. Bernstein and T. Lange, "eBACS: ECRYPT Benchmarking of Cryptographic Systems.," [Online]. Available: http://bench.cr.yp.to. [Accessed 10 June 2013].

[15] "OpenSSL," [Online]. Available: http://www.openssl.org/.

[16] A. Tavakoli, M. Casado, T. Koponen and S. Shenker, "Applying NOX to the Datacenter," in HotNets, Citeseer, 2009.

Othman OTHMAN M.M. received M.S. Degree in Graduate School of Information Science and Electrical Engineering from Kyushu University, Japan. And, he received B.S. in Faculty of Engineering form An-Najah National University, Palestine. He is a Ph.D. student and belongs to the department of Advanced Information Technology, Graduate school of Information Science and Electrical Engineering, Kyushu University, Japan.

Koji OKAMURA. who is a Professor at Department of Advanced Information Technology and also at Computer Center, Kyushu University, Japan. He received B.S. and M.S. Degree in Computer Science and Communication Engineering and Ph.D. in Graduate School of Information Science and Electrical Engineering from Kyushu University, Japan in 1988, 1990, and 1998, respectively. He has been a researcher of MITSUBISHI Electronic Corporation, Japan for several years and has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan and Computer Center, Kobe University, Japan. He is interested in Internet and Next Generation Internet, Multimedia Communication and Processing, Multicast/IPv6/QoS, Human Communications over Internet and Active Networks. He is a member of WIDE, ITRC, GENKAI, HIJK projects and Key person of Core University Program on Next Generation Internet between Japan and Korea sponsored by JSPS/KOSEF.