View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Technical Disclosure Common

Technical Disclosure Commons

Defensive Publications Series

January 2020

AUTHENTICATION OF APPLICATION FLOWS IN SOFTWARE DEFINED NETWORK DEPLOYMENTS USING A TRANSACTION MODEL

Niranjan M. M

Nagaraj Kenchaiah

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation

M, Niranjan M. and Kenchaiah, Nagaraj, "AUTHENTICATION OF APPLICATION FLOWS IN SOFTWARE DEFINED NETWORK DEPLOYMENTS USING A TRANSACTION MODEL", Technical Disclosure Commons, (January 08, 2020)
https://www.tdcommons.org/dpubs_series/2850

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

AUTHENTICATION OF APPLICATION FLOWS IN SOFTWARE DEFINED NETWORK DEPLOYMENTS USING A TRANSACTION MODEL

AUTHORS:

Niranjan M M
Nagaraj Kenchaiah

ABSTRACT

As Software Defined Networking (SDN) enables third party applications to be integrated into the architecture, a malicious application could have as much of a detrimental effect on the network as a compromised controller. In order to avoid the deployment of malicious/compromised applications, controllers and applications should establish a trusted connection and authenticate the identity of applications and their flows before exchanging control messages. Application flows may be considered network configurations sent by applications that are managed by controllers, which install network configurations into switches. Without authentication, applications may inject malicious configurations into network devices at will, which could reduce network availability, reliability, and/or even lead to a network breakdown. Presented herein are techniques involving a Transaction model that can be utilized to authenticate applications and their flows and further provide trust establishment between a controller and a switch in multi-provider SDN deployment.

DETAILED DESCRIPTION

The nature of high programmability and configurability on network devices (e.g., switches) in SDN environments (e.g., OpenFlow® environments) forces heightened attention to security issues involving the application plane. Applications (e.g., traffic engineering) may provide a variety of management services for a network by configuring application flows but may also introduce new threats for the network. For example, a malicious or compromised application may inject malicious application flows into network devices thereby potentially leading to dramatic consequences.

In other instances, through utilization of an application an attacker can impersonate (masquerade) as a controller and modify information such as configuration information in order to prepare for other types of attacks in the future. Such application plane attacks may include, for example, service chain attacks, Northbound Application Programming Interface (API) abuse, impersonating a controller, resource exhaustion, etc.

Consider one type of application plane attack such as a service chain attack, as illustrated in Figure 1, below.

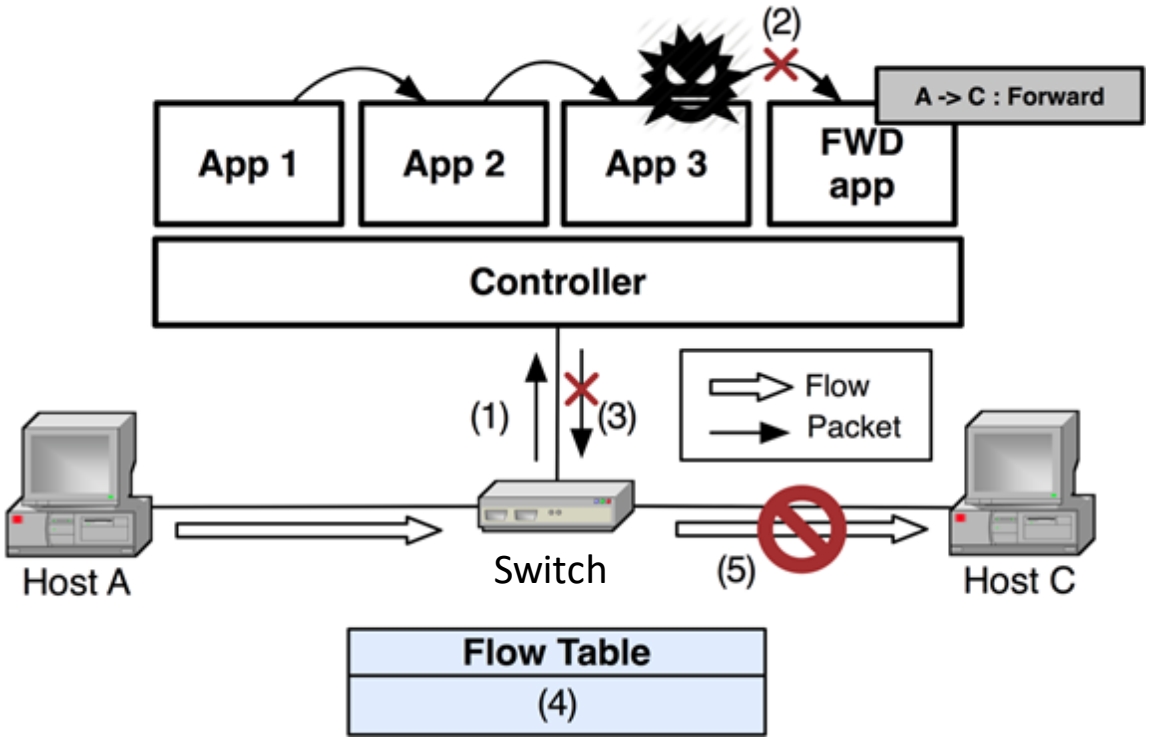

Figure 1

As illustrated in Figure 1, a service chain attack may involve a control message drop as follows:

1. A Packet-In is received by a controller and is passed to applications (Apps) including App 1, App 2, App 3 as per a given service chain.

2. App 3 (which may be a malicious application for this example) drops the Packet-In without passing it to a forwarding (FWD) application.

3. The forwarding application does not reply to Packet-In.

4. No flow rule would be installed in the switch.

5. Because no flow rule is installed in the switch, Host A cannot communicate with Host C.

Another example of a service chain attack can include an infinite loop attack, which may involve App 3 being programmed to fall into an infinite loop, thereby leading the controller instance to freeze.

Consider another type of application plane attack such as Northbound API abuse, which can include application eviction, as illustrated in Figure 2, below.

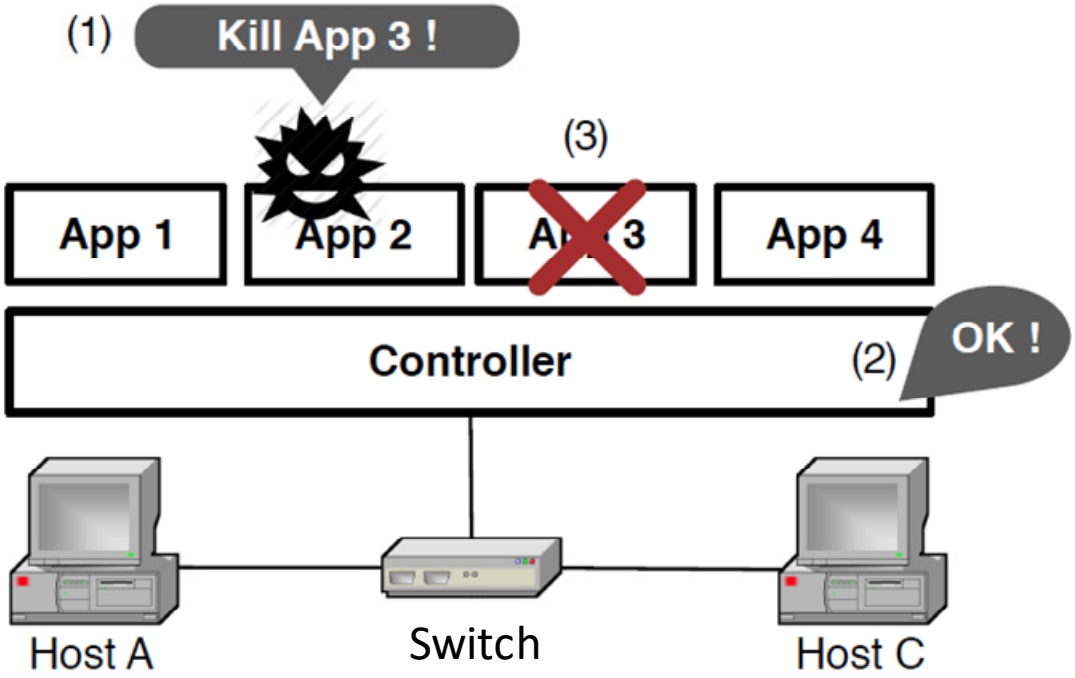

Figure 2

As illustrated in Figure 2, application eviction may be caused as follows:

1. App 2 (which may be a malicious application for this example) calls a function to terminate App 3 via a Northbound API.

2. The controller accepts the App 3 termination request.

3. Innocent App 3 is terminated.

Consider yet another type of application plane attack such impersonating a controller, which can result in other types of attacks. For example, by turning off the ports of a switch, an attacker can cause the ports to fail to provide a traffic forwarding service (e.g., denial of service). In another example, an attacker may modify a resource allocation

policy in a switch to achieve better service than may be defined by the attacker's service level agreement (SLA).

Along with modification of configuration information, the attacker may obtain the SLA or service chaining of an application in order to use it for a future attack. Further, the attacker may report incorrect statistics to an application, which can cause the application to make incorrect decisions (e.g., for traffic engineering decisions).

In an another example, an attacker can also change vital information on a switch such as the security policy and/or credentials of the controller in order to spoof the controller or prepare tampering/eavesdropping attacks on future communications. In yet another example, an attacker can change the reachability of a switch (e.g., changing an Internet Protocol (IP) address and/or port number used to connect to a controller) in order to shut down current connection to the switch and induce the switch to reconnect to an untrusted/fake controller.

These example attacks are more prominent in cases in which third-party SDN applications are integrated or when using cloud services provided by another SDN in which an attacker may obtain information about the controller and applications more easily. For cases in which SDN applications belong to the same provider, such as the SDN controller, it is typically more difficult for the attacker to get information from the inside SDN network. However, for cases in which third-party SDN applications (i.e., developed and published by third party) are integrated or using a cloud service provided by another SDN provider, then an attacker can obtain information about the controller and applications more easily. If an attacker is a third-party application itself, then the controller and applications may be subject to attacks even more easily.

Thus, what is needed is an authentication mechanism to authenticate application flows from legitimate applications and verify the authenticity of application flows that are significant for configurable SDN devices before acting upon such flows. Further, to handle cases involving masquerading (impersonation) as a controller, trustworthiness of a controller should be considered (i.e., a switch should verify the trustworthiness of a controller before applying the flows or configurations sent by the controller).

Some solutions may involve setting permissions and providing an isolation mechanism to enforce the permissions at an API entry. Such solutions typically apply

minimum privilege on applications protecting the network from control-plane attacks. A limitation of such solutions, however, is that such solutions do not implement permission checks prior to authorizing application commands. Other solutions may involve role-based authentication for determining the security authorization of each application. A limitation of this approach, however, is determination of an appropriate security authorization level.

As SDN enables third-party applications to be integrated into a networking architecture, a malicious application could have as much of a detrimental effect on a network as a compromised controller. Similarly, a poorly designed or buggy application could unintentionally introduce vulnerabilities to the system.

In order to avoid the deployment of malicious/compromised applications, controllers and applications should establish a trusted connection and authenticate the identity of applications and their flows before exchanging control messages. Without authentication, applications may inject malicious configurations into network devices at will, which could reduce network availability, reliability, and/or even lead to a network breakdown. For techniques presented herein, application flows may be considered network configurations sent by applications that are managed by controllers, which install network configurations into switches.

This proposal provides techniques that involve utilizing a Transaction model of blockchain to facilitate the authentication of application flows from all legitimate applications and verify the authenticity of such flows before they are applied in a network. Along with authentication, techniques of this proposal also provide for traceability and accountability of application flows.

There may be instances in which an attacker, through a malicious application, may impersonate a controller and attempt to inject malicious flows in a network. Thus, it may not be sufficient to provide only for the authentication of applications and their flows. Instead, trust may need to be established between a controller and switches so that only a trusted controller may push application flows to the switches.

Trustworthiness between a controller and a switch can be provided via a Trusted Platform Module (TPM) utilizing Remote Attestation techniques. For establishing trustworthiness, a TPM on the controller may populate a measurement list (using a set of registers, referred to herein as Platform Configuration Registers (PCRs)). The controller

can send this measurement list along with a signature (signed using a private Attestation Identity Key (AIK) of the local TPM) to a remote attester. The remote attester can validate the signature to identify the controller as a trusted controller and can verify the PCR values against boot event logs to verify the integrity state of the controller (e.g., to verify whether trusted firmware, a trusted boot loader, or a trusted Operating System (OS) kernel are running on the controller).

Once controller is verified as trusted, the controller will be added to the blockchain (and hence to the distributed ledger) of trusted controllers. Optionally, the blockchain can be incorporated into the switches as well. Utilizing the distributed ledger, all the blockchain enabled switches can determine the trustworthiness of a controller before applying application flows and/or configurations sent by the controller.

For simplicity, techniques of this proposal are divided into four parts:

i. Identity creation for applications and their flows
ii. Transaction creation on the blockchain
iii. Functions to authenticate and verify applications and their flows
iv. Traceability and Accountability

i. Identity creation for applications and their flows

To introduce the Transaction model for authentication of application flows from all legitimate applications (APPs), various information may be utilized (per-application) such as application identity (ID_app), a public key (PK_app), a private key (SK_app), and an application type (Type_app).

An application may be defined as APP = (ID_app, PK_app, SK_app, Type_app). For the application definition, ID_app is a unique identifier that is used to represent the identity of an application. In some instances, an application can use a unique application package name as its identity. Further for the application definition, an asymmetric key generation algorithm "KeyGeneration()->(PK, SK)" is used to create a key pair including a public key PK_app and a private key SK_app for an application. Finally, the category of an application can be identified by Type_app.

In SDN, an application cannot be realized without a controller and a switch, each of which can be identified by their unique IP address or Media Access Control (MAC)

address. Thus, a controller can be identified via an identifier (ID_controller) and a switch can be identified via an identifier (ID_switch).

A signature of an application can utilized for the controller to authenticate the application and its flows. For this proposal, a signature for an application can be defined as SIGN_APP = DS.Signature(SK_app, ID_app || Type_app || ID_controller).

Similar to definitions provided for an application, identities for flows of an application can be defined utilizing a flow identity (ID_flow), content, the public key of an application (PK_app), a controller identity (ID_controller), and a switch identity (ID_switch). In short, an application flow can be defined as APP_FLOW = (ID_flow, content, PK_app, ID_controller, ID_switch). This tuple for an application flow can be used indicate from where the application flow originates and to where the application flow contributes.

When an application flow is sent by an application, the application attaches the application flow with its signature for the flow by its private key SK_app. Thus, a signature for an application flow can be defined as SIGN_APP_FLOW = DS.Signature(SK_app, ID_flow || ID_controller || content)).

ii. Transaction creation on the blockchain

In Part(i), above, identities for applications and their flows were created. In Part(ii), transactions for applications and their flows are created on the blockchain.

Whenever an application connects with the controller to provide network flows for switches that are connected to the controller, application information along with the relationship between the application and the controller will be recorded as transactions T_APP (transaction of the application) and T_APP-CONTROLLER (transaction of the controller) on the blockchain as:

T_APP = (ID_T_app , ID_app, PK_app, Type_app, ID_controller, SIGN_APP)
T_APP-CONTROLLER = (ID_T_app-controller, ID_T_app, ID_T_controller)

From the T_APP transaction, only legitimate applications owning the public key PK_app can succeed in verifying the signature of an application that was previously accepted by

the controller. Thus, signature verification for an application can be defined as VER_APP = DS.Verification(PK_app, DS.Signature(SK_app, ID_app || Type_app || ID_controller)). By utilizing this transaction model, an application identity can be verified.

Next, consider a transaction for an application flow represented by T_flow-afore and T_flow-after, which include the identity of the flow, the flow content, the identifier of the targeted controller, the public key of the flow-stemming application, and a signature signed by the flow-stemming application with its private key.

Further consider that T_flow-afore and T_flow-after may be a transaction associated with an application flow that is injected into the network by a specific application, passed by a related controller, and ultimately installed on a specific switch, which can be represented as follows:

T_flow-afore = (ID_T_flow-afore, ID_flow, content, ID_controller, PK_app, SIGN_APP_FLOW)

T_flow-after = (ID_T_flow-after, ID_flow, ID_controller, ID_switch, state)

T_flow = (ID_T_flow, ID_T_flow-afore, ID_T_flow-after)

Here, T_flow-afore records the process from some specific application to a specific controller and T_flow-after records the process from the specific controller to a specific switch.

Before generating T_flow-afore, the controller is to authenticate all application flows for the application. A malicious flow can be filtered because controller will audit the application creating the flow and verify its identity with the transaction T_app from the blockchain. When the flow is installed in a switch by the controller, the transaction T_flow-after is generated, which can later be used for reply detection. The switch would further verify the trustworthiness of the controller (by looking into distributed ledger) before applying the application flows sent by that controller.

iii. Functions to authenticate and verify applications and their flows

Up to Part(iii), identities for applications and their flows are created (Part(i)) and transactions for applications and their flows are populated on the blockchain (Part(ii)). In

Part(iii), functionalities are defined for authentication and detection of new application flows based on existing transactions on the blockchain.

a. Authentication for Application Flows

For this proposal, an API can be implemented as AuthFlow(ID_flow, PK_app, ID_app) to enable the authentication of application flows. This API can be used to determine whether a given flow is related to a registered application and controller utilizing the records T_APP and T_APP-CONTROLLER on the blockchain.

In other words, the controller verifies the identity of an application creating the given flow based on the transaction T_APP and T_APP-CONTROLLER using another API, IsRightFlow(ID_flow). If the relationship records exists, the controller utilizes the public key of the application PK_app to verify the signature using another API, VerifyFlow(ID_flow, PK_app). If the signature is valid (is signed by the registered application) then the flow content can be considered verified (i.e., the content of the flow has not been altered).

b. Reply Attack Detection for Application Flows

Another API, IsFlowReplyDetection(ID_flow, PK_app, ID_app) can be utilized to detect reply flows by auditing the identifiers of flows based on the transactions T_flow-afore. In at least one implementation, the reply attack detection API can be called by the AuthFlow() API as a sub-function before executing the logic to verify the identity of an application flow.

Thus, the reply attack detection API is used to check ID_flow and determine if it previously existed. If the flow previously existed, the protocol utilizes PK_app to determine whether T_APP exists. If T_APP exists, the protocol decreases the reputation value of the application as punishment and returns a value of TRUE. Otherwise if it is a new flow, a value of FALSE is returned.

c. Legitimate Application Determination

Another API, IsRightFlow(ID_flow), can be implemented to determine whether an application creating a given flow is legitimate based on the transaction T_APP, T_APP_CONTROLLER.

d. Flow verification

Yet another API, VerifyFlow(ID_flow, PK_app), can be implemented to utilize the public key of an application to verify a flow signature in order to determine whether content of the flow is modified or not.

iv. Traceability and Accountability

Traceability and accountability for application flows can assist operators to troubleshoot a network once a network device breaks down and/or suffers from abnormal network behaviors. On the other hand, for scenarios involving flow arbitration, a flow arbitration system that provides for the ability to arbitrate flow conflicts can be utilized to identify which flows may be sent by which applications where traceability and accountability for application flows that may be urgent can be provided.

The Transaction model utilized for techniques of this proposal may also enable the traceability of application flows injected into the network in order to trace a cause of a network event. Thus, logged transactions may not only record the network flows sent among SDN controllers and switches but may also capture all network behavior. Within a running network, if the network suffers abnormal attacks, the attack processes can also be logged as transactions. With these logged transactions of attack trajectories, future attacks launched on the network can be identified (e.g., using attacks pattern recognition).

For enhancing traceability across SDN controllers and switches, a transaction for network events can be created as T_EVENT, which may be similar to transactions created for applications and their flows as follows:

EVENT = (ID_event, event, PK_switch, ID_controller, ID_switch)

T_EVENT = (ID_T_event, ID_event, PK_switch, ID_controller, ID_switch, EVENT)

In summary, techniques are presented herein provide a Transaction model that can be utilized to authenticate applications and their flows and further provide for trust establishment between a controller and a switch in multi-provider SDN deployment. In some implementations, existing mechanisms for blockchain that can scale up to VISA level transactions can be utilized to handle scalability aspects of the techniques of this proposal.