March 2022

SECURE PROVENANCE METHOD FOR TRACEABILITY AND TROUBLESHOOTING IN SDN DEPLOYMENTS

NIRANJAN M M

SECURE PROVENANCE METHOD FOR TRACEABILITY AND TROUBLESHOOTING IN SDN DEPLOYMENTS

AUTHOR:
NIRANJAN M M

ABSTRACT

SDN deployments being large, finding faults in such large system is hard. Debugging in complex SDN deployments possess great challenges as the state of the system continuously changes due to the occurrence of the events, addition/deletion of application flows, network issues etc., Conventional logging method just logs all the generated messages locally and/or to the remote syslog and it is difficult to debug because of its volume. There are techniques which provides signed logging system, which just enhance authentication and trustworthiness of the logged messages but does not improve the traceability and troubleshooting any issue. The techniques presented herein, propose method to maintain meta provenance which includes events, application flows and state change of controllers and switches, in the form of Transaction in the Hyper Ledger of the private Blockchain so that it would improve traceability, debug-ability, accountability and troubleshooting in complex SDN deployments.

DETAILED DESCRIPTION

SDN deployments being large, finding faults in such large system is hard. Debugging in complex SDN deployments possess great challenges as the state of the system continuously changes due to the occurrence of the events, addition/deletion of application flows, network issues etc., Sometimes they are of distributed nature and identifying faults in those deployments adds complexity. These faults are often partial, irregular and may result in abnormal behaviour rather than system failure. So, diagnosing a problem in such systems require collecting relevant information from many different SDN entities (such as SDN Controllers, Switches, etc.,) and correlating those with the problem.

Conventional logging method just logs all the generated messages locally and/or to the remote syslog and it is difficult to debug because of its volume. It will be more difficult when same syslog server is used for multiple entities. And it is difficult to bifurcate specific information about the state and event of the entities from these logs. And it does not carry information to troubleshoot abnormalities/issues occurred due to missing events (the events which did not occur at-all). Moreover, it does not carry information required for tracing and troubleshooting any incident result due to the attacks such as repudiation attack (this attack is used to modify the authoring information by attacker to log wrong data to log files. If this attack takes place, the data stored on log files can be considered invalid or misleading). And hence it lacks traceability and troubleshooting, but these are very much required for the complex SDN deployments especially when it is of distributed nature.

There are techniques which provides signed logging system using TPM methods, which just enhance authentication and trustworthiness of the logged messages but does not improve the traceability and troubleshooting any issue (in other words, these methods does not simplify the debug-ability). Just to re-iterate, these signing methods also does not in-corporate any extra information, which can be used for troubleshooting issue occurred especially due to missing events.

Hence it is necessary to have methods to collect state and event information about the SDN Controllers and switches along with network events (such as link or/and port state changes) so that we can diagnose any problem in the SDN deployments. And we need a method to enhance troubleshooting any abnormalities/issues/attacks occurred due to missing events. Without integrity and validity of information, it may mislead and give an unwanted result. Hence, we also need to ensure integrity and validity of the information as well as how information has been manipulated up to its current state.

To minimise the debug-ability, troubleshooting abnormalities/issues occurred, overcome the attacks such as repudiation attack etc., we need a method to securely capture state, application flow and event information, which we can use to reconstruct/recreate State Machine and Event Diagrams (UML) which we can directly map to the code. There are plenty of tools exists which converts State Machines and Event Diagrams to generate code and vice-versa. These generated state machine and event diagrams would help to map abnormalities/issues due to missing events/flows. Provenance is one of the techniques where state and event information related to the whole system is cached and used to track the issues whenever problem occurred in the deployment. It is used for tracing and analysing problems in complex systems and getting reason behind errors and unwanted behaviour. Provenance technique would improve traceability and troubleshooting by caching metadata of state changes, application flows and events of the SDN entities such as controllers, switches etc., This metadata is maintained as transaction in the Hyper Ledger of the private blockchain, hence all authenticated SDN entities has access to this Hyper Ledger for tracing and troubleshooting network issues at any point of time and on any SDN entity.

The techniques presented herein, propose method to maintain meta provenance which includes events, application flows and state change of controllers and switches, in the form of Transaction in the Hyper Ledger of the private Blockchain so that it would improve traceability, debug-ability, accountability and troubleshooting in complex SDN deployments. Hyper Ledger of private Blockchain is incorporated in our method for providing authenticated ledger which is mandatory requirement for enterprise SDN deployments. As we know Blockchain maintains tamper proof transactions (once it is created on the Blockchain, it cannot be modifiable) and hence can be used for troubleshooting abnormalities due to attacks such as repudiation attack etc., Note here, that only state, application flows and event changes are maintained as Transaction in the Hyper Ledger and not the whole syslog. For syslog, existing signed logging system could be used.

The information collected in this method would be the first one to look for troubleshooting abnormalities/issues in the network using states and events before debugging further using syslog. Because of huge volume of data and logs getting generated by multiple features, debugging using syslog would require in-depth understanding of all SDN entities and their applications. Both syslog or signed logging methods may not provide any information to debug issues occurred due to missing events and does not help to find attacks such as repudiation attack etc.,

SDN deployment comprises of controllers, switches, and other network devices. These entities may encounter resource and network failures. Hence analysing provenance would be an important mechanism to detect network failures and at the same time monitor resource malfunctions. Provenance helps to explain the series of actions which led to the change of an object (it could be due to data, state, or event of an entity in SDN deployment) to its current state from that of its origin. So, tracking and then analysing preceding events helps to diagnose actual reason behind system failure or security breaches. This is analogous to positive provenance.

The absence of events like loss of packet data or unavailable events can also be explained by using counterfactual reasoning to identify conditions under which these events could have occurred. To find a missing or negative event in a SDN entity, we need to consider all possible positive events from the SDN entity and its neighbouring SDN entities that would have resulted in the missing event and from that information the reason can be deduced. In this case, a large chain of events needs to be considered. This is analogous to negative provenance.

However, there may be situations where a tampered SDN entity (controller or switch) may give wrong information in the absence of TPM (attestation/trustworthiness) method. In such cases, provenance can be used to track manipulation or tamper-evident properties from neighbouring to assist operator/administrator to detect compromised SDN entity. So, provenance information of events provides useful information to analyse, optimise and secure any system. The metadata for the network events and application flows can be customisable by the configuration based on the requirement (priority, size of the deployment, etc) and hence minimising the number of transactions maintained in the Hyper Ledger.

The technique presented herein works as explained below:

SDN deployments comprises of controllers, Switches and other SDN entities along with network events generated due to link and port state changes etc., For simplicity, only Controllers and Switches are considered, but doesn't limit to these and can be extended to other entities such as, virtual and Cloud entities etc., Hence we need traceability, debug-ability and accountability to troubleshoot whole network once the system breaks down or suffers from abnormal behaviours.

This method carries provenance information (state change, events etc.,) as transaction of the private Blockchain in Hyper Ledger for all the entities of SDN deployments. Transaction model enhances traceability and debug-ability by recording all state changes, network events among controllers and switches, which would help in capturing all network behaviour. Within a running network, if the network suffers abnormal attacks, the attack process is also logged as a transaction. With these logged transactions of attack trajectories, the future attacks lunched on the network can be identified using attack pattern recognition.

For enhancing traceability and debug-ability across controllers and switches, create transaction for the SDN entities with states (T_STATE), events (T_EVENT) as below and add to the Hyper Ledger of private Blockchain.

i. Transaction for the States:

Capture required major state changes of the controllers starting from the boot-up as "state provenance" in hyper ledger as Transaction.

State of the controller and its Transaction are defined as

STATE_controller = (ID_controller, ID_state, old_state, new_state, SK_controller, PK_controller)

Transaction_STATE_controller = (ID_controller, ID_state, STATE_controller, SIGNATURE_controller)

SIGNATURE_controller = DS.Signature (SK_controller, ID_controller, ID_state)

Here,

ID_controller = Identity of the controller

ID_state = Identity of the state which would be maintained as Transaction

old_state, new_state = Previous State and Current State respectively.

SK_controller, PK_controller = Private Key and Public Key of the controller (used for authenticating the information)

SIGNATURE_controller = Transaction for the state change is signed by the controller using its Private Key

Capture required major state changes of the Switch starting from the boot-up as "state provenance" in hyper ledger as Transaction.

State of the switch and its Transaction are defined as

STATE_switch = (ID_switch, ID_state, old_state, new_state, SK_switch, PK_switch)

Transaction_STATE_switch = (ID_switch, ID_state, STATE_switch, SIGNATURE_switch)

SIGNATURE_switch = DS.Signature (SK_switch, ID_switch, ID_state)

Here,

ID_switch = Identity of the Switch

SK_switch, PK_switch = Private key and Public Key of the Switch (used for authenticating the information)

SIGNATURE_switch = Transaction of the state change is signed by the switch using its Private Key

ii. Transaction for the Events:

Capture major events (customisation by configuration) occurred on controllers and Switches starting from the first negotiation as "event provenance" in hyper ledger as Transaction.

Event occurred on controller and Switch and its Transaction are defined as

EVENT_on_controller = (ID_controller, ID_event, event, SK_controller, PK_controller)

Transaction_EVENT_on_controller = (ID_controller, ID_event, EVENT_on_controller, SIGNATURE_controller)

Here,

ID_event = Identity of the event which would be maintained as Transaction

SIGNATURE_controller = Transaction of the event on controller is signed by the controller using its Private Key.

Similarly, transaction would be created for events on Switch, but signed by switch using its Private Key.

iii. Transaction for the Application Flows:

Application flows would be generated from the applications running on the Controller to configure policies or executing commands on the switch. As we know SDN supports integration of third-party applications (e.g., traffic engineering etc.,), and may inject malicious flows and can attack the network, hence we need even applications to authenticate itself for every flow generated by them. From traceability, capture all application flows as "data provenance" in the blockchain as Transaction.

Application and their flows are defined as

APP_FLOW = (ID_controller, ID_switch, ID_APP, ID_FLOW, SK_APP, PK_APP)

Transaction_APP_FLOW = (ID_controller, ID_switch, ID_APP, ID_FLOW, SIGNATURE_APP)

SIGNATURE_APP = DS.Signature (SK_APP, ID_Controller || ID_switch || ID_APP || ID_FLOW)

Here,

ID_APP = Identity of the application

ID_FLOW = Identity of the Flow generated by the Application

SK_APP, PK_APP = Private Key and Public Key of the application (used for authenticating the information)

SIGNATURE_APP = Transaction of the application flow is signed by the Application generating the flow using its Private Key

Once Transactions are created for the States, Events and application flows as above, these will be added to the private blockchain and hence to the Hyper Ledger. Any of the issues or abnormalities can be troubleshooted using this transactional information. In short, captured

state, application flow and event information can be used to reconstruct/recreate State Machine and Event Diagrams (UML) which can be directly mapped to the code. There are plenty of tools exists which converts State Machines and Event Diagrams to generate code and vice-versa. These generated state machine and event diagrams would help to map abnormalities/issues due to missing events/flows. To ensure integrity and validity, Signatures would be added to each transaction by the entity who adds to the private blockchain. Other entities would verify the Signature using corresponding Verification function.
For example, for the flow validation, verification function at the switch would be as below:

```
VER_APP=DS.Verification (PK_APP,SIGNATURE_APP)
```

Similarly, verification function for validating controllers and switch identity would be as below:

```
VER_controller = DS.Verification (PK_controller, SIGNATURE_controller)
VER_switch = DS.Verification (PK_switch, SIGNATURE_switch)
```

Along with Traceability and troubleshooting, these transactions would help in Accountability by listing out how many flows generated between controllers and Switches.

The state provenance carry, state changes of the SDN entities such as Controller, Switches etc., starting from the bootup. State of the SDN entity just does not limit to states such as Booting up, Running, Active, Standby, etc., but also includes the start/stop of the application services running, security services etc., The event provenance carry, events occurred on the SDN entities starting from the bootup. Events would include link up/down, port up/down, stoppage of applications, restart of applications etc., Administrator would have an option to configure level of information (criticality of state change, priority of events, priority of application flows etc.,) required as per the serviceability required.

In summary, the techniques presented herein provides periodic auditing for network behaviours (network events associated with resulting network states), which helps to enforce the stability and robustness of the SDN deployment. It improves the traceability and troubleshooting for network issues. Moreover, it provides resistance against future attacks, by incorporating attack pattern recognition using provenance data and events. In cases where SDN deployments are in a suspicious state and if there are faulty or misbehaving entities, provenance information can play a vital role to debug and identify them. It can also be used to assess the damages that faulty entity might have caused to the system. Provenance information can be used to answer audit questions, to predict future workloads. Based on such predictions, performance of the system, as well as the availability of service, can be identified.