Security Analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN Controllers

Ramachandra Kamath Arbettu
Department of Computer Science
TU Darmstadt, Germany
Email: ramachandra.kamatharbettu@stud.tu-darmstadt.de

Rahamatullah Khondoker, Kpatcha Bayarou, Frank Weber
Fraunhofer Institute for Secure Information Technology (SIT)
Darmstadt, Germany
Email: firstname.lastname@sit.fraunhofer.de

***Abstract*—There is an immense expectation on Software-Defined Networking (SDN) in industry as a novel approach towards potentially replacing conventional network management and control. However, SDN is not immune to security vulnerabilities which currently exist in the legacy systems or which may newly arise due to change in the network design. Since the beginning of SDN development, primary focus of research was on separation of control plane from data plane by keeping performance and operational flexibility unchanged. In the due course of achieving this, security aspects of an SDN have taken a back seat. Even though separation of control plane from a data plane is a great step towards simplification of network management, it subjects the network into a potential two way target for intruders to gain control. Due to the centralized design of SDN, compromising security of a controller will be as good as compromising the security of a whole network. Enterprises which are moving towards adapting SDN are concerned about security issues and the resulting problems. In this paper, we analyze the security issues of few of the widely used controllers. We found that the OpenDaylight controller is the most secure one compared to the other controllers. In addition, this paper also provides a snapshot of current development in security aspect of SDN controllers such that it may help SDN controller developers to identify the issues and rectify the same in future releases.**

***Index Terms*—Software Defined Networking (SDN), Controller Security, STRIDE, Network Security, Security Analysis**

I. Introduction

Software Defined Networking (SDN) has opened up new opportunities for network enthusiasts to experiment and deploy innovative ways of network management and to dynamically take control of packet forwarding in their network. However, what has not changed is the need to understand the network topology and stay informed about how network elements perform across the topology. This is still a key factor in achieving performance and security goals which are probably the reasons why network security and monitoring receive immense importance in the networking community. With the greater control of network in a centralized fashion comes a greater responsibility for the SDN controller to monitor and prevent any attacks on the network.

The SDN architecture could be broadly decomposed into three main layers (Fig. 1). The data plane, the bottom most layer of an SDN architecture, is responsible for data-path implementation. The data plane consists of switches which receive flow rules from higher layers as an instruction which will be persisted in the switches flow table. If the received packet does not match any entry in the flow table, the switch forwards that packet to the controller for a decision. The control plane is an implementation of the control-path of a legacy network. This layer is the critical layer of an SDN architecture which undertakes the task of traffic engineering, traffic shaping, and network management. In this paper we mainly focus on this layer. Customization of packet forwarding, policy management, user management, and Quality of Service (QoS) are handled by the application layer with the help of a controller. All of the network functions and monitoring tools are usually parts of the application layer in an SDN architecture. The interaction between different layers have been carried out with the help of widely used protocols. A North Bound Interface (NBI) is an interface between the controller and the application layer which supports most of the application layer protocols for interactions namely HTTPS, SFTP etc. The south Bound Interface (SBI) is responsible for the interaction between the controller and the underneath switches, OpenFlow [1] is being widely used as a SBI protocol. SBI integration is also supported by Simple Network Management Protocol (SNMP) [2], Command Line Interface (CLI), Telnet (TN) [3], Secure Shell (SH) [4] etc.

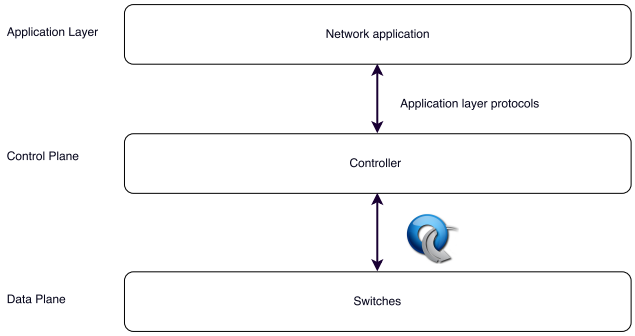

Fig. 1. SDN Architecture Diagram

An SDN controller provides multiple levels of access to network users, applications, and devices based on the roles. A controller remains as a potential high value target of the attackers because of multiple reasons: firstly, it is the "brain" [5] of an SDN which is logically centralized by design, secondly, there are multiple interfaces that are open to external application usage which an intruder can exploit and gain

978-1-4673-8991-4/16/$31.00 ©2016 IEEE

control over the network.

In addition to data flow, there are multiple critical data structures or data stores comprised by the controller. These data stores may include but are not limited to topology information, user details, access control, and policy information. Integrity of this data is critical due to the fact that the controller is the single entity that manages the network by using data available in the local data store.

In this paper, several controllers have been selected and analyzed how each of the above mentioned interfaces, processes, and internal data of a controller are secured. For each of the controllers, an individual STRIDE threat matrix have been created to summarize the maturity of controller against known attack categories.

For the analysis, we considered two Java based open source controllers, namely OpenDaylight (ODL) [6] and Open Network Operating system(ONOS) [7]. In addition, we chose a python based controller Ryu which is widely used in research areas and Rosemary [8] which is a proprietary controller software. The STRIDE [9] threat modeling framework is used to analyze the security levels of individual controllers.

The rest of the paper is organized as follows. Section II sheds light on the STRIDE [9] modeling techniques. Section III is dedicated to discuss available controller solutions and their differences. The security analysis of the selected controllers is discussed in Section IV. The paper is summarized and concluded in Section V and VI respectively.

II. STRIDE

STRIDE is a framework developed by Microsoft for threat modeling purpose which has been an integral part of Security Development Lifecycle (SDL). In SDL approach, security of an application is prioritized over other parameters such as performance, availability, and integrity for evaluation during Software Development Lifecycle (SDLC). STRIDE by itself is an acronym for six threat categories such as **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service (DoS) and **E**levation of Privileges.

Before the analysis, it is important to decompose a system into different components and analyze threats that could affect a component by considering its internal behavior and its interaction with other components. Data Flow Diagram (DFD), a graphical representation of inter-process interaction and data exchange information among different entities, plays an important role in STRIDE. It also represents data injection, data manipulation and data output of individual processes in a system. Before starting a threat analysis, it is important to create a correct DFD because an incorrect DFD may lead to failure of the threat modeling process [10]. However, it is important to identify different components in advance to create a DFD. Once all of the processes involved are finalized, one can use different symbols to represent different components as mentioned in Table I.

There are multiple techniques which are used for threat modeling including Petri Nets [11] - a mathematical modeling of a system, Flexible Modeling Framework (FMF) [12] - a network safety modeling technique, SecureUML [13] - a model driven security of distributed systems, and DREAD [14] - system risk assessment tool are few examples.

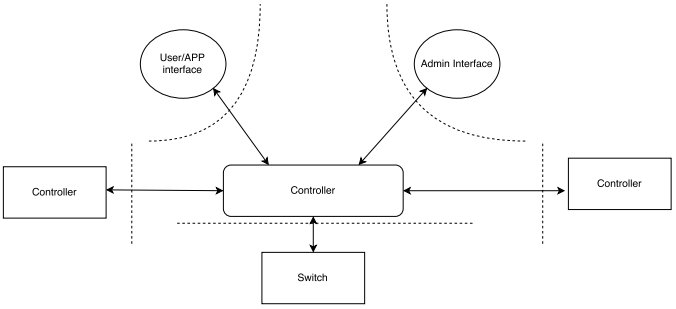

Fig. 2. Data Flow Diagram of an SDN controller

Component	Symbols
Data Flow	Arrow
Process	Circle or Oval
Interactors	Rectangle
Trust Boundary	Dotted line

TABLE I
MEANING OF SYMBOLS IN DFD

The STRIDE threat modeling technique can be applied at an application layer solely based on the components which interacts with external entities and the ways the interaction take place. Since our aim in this paper is to assess security aspects of a controller by considering communication channels and communication entities of SDN without considering the internal details of it, we use STRIDE as it best fits for our analysis.

III. SDN CONTROLLERS

An SDN controller is a realization of control plane of a network in a logically centralized fashion which functions as a facilitator and takes the decision of packet routing based on the global view of the network topology. An application can communicate with the controller through a NBI and injects information such as routing and security policies to be applied. A controller translates and optimizes those instructions to flow rules and forwards those rules to switches by using protocols such as OpenFlow.

A. OpenDaylight

OpenDaylight (ODL) [6] is a Java based open source controller developed and managed by the linux foundation. ODL is based on modular architecture which gives flexibility to a developer to plug-in new applications using north bound APIs. ODL supports OpenFlow and other standard protocols from IETF for south bound communication. In addition to this, a vendor can also plug-in its own protocols as new modules in ODL for north and south bound communication. The east west bound communication of ODL is based only on OpenFlow protocol.

In this study, the ODL controller has been considered for two main reasons, primarily because ODL tends to standardize

the APIs to achieve industry recognition which in turn may prioritize security requirements. Secondly, ODL has a dedicated DoS attack detection and prevention module packaged with recent builds.

B. Open Network Operating System

Open Network Operating System (ONOS) [15] is yet another Java based controller software. Like ODL, ONOS is also modular and developed as a distributed system which makes use of Open Service Gateway initiatives (OSGi) standard to maintain different modules.

Despite its resemblance of architecture with ODL, ONOS has been additionally considered within this study. This is due to the fact that ONOS has already been deployed by a number of major telecom equipment vendors, hence it has been introduced to the enterprise market.

C. Rosemary

Rosemary [8] is an open source controller software with an agenda of having a micro Network Operating System (NOS) architecture along with a set of network applications plugged into it. Each network application has to run inside an instance of the Rosemary controller. By doing so, NOS has complete control over resource utilization and access from user applications.

The Rosemary controller has been considered due to its micro NOS architecture and the importance given by the Rosemary development team to the robustness and security aspects in its implementation objectives.

D. Ryu

Ryu [16] is an open source python implementation of an SDN controller with a component based architecture. It has a set of well-defined APIs for network application development and it has been widely used by research community. Ryu also supports multiple protocols in the SBI for hardware integration and configuration. The event handler of a Ryu controller dispatches the events to all of the subscribed applications. Applications need no authentication in order to subscribe to such events. These aspects will be emphasized more in the analysis section of the paper.

Even though it is targeting the research community, Ryu is a controller of interest for the analysis mainly due to its single process and multi-threaded architecture design in which new components will be plugged in and executed as a new thread.

Since most of the controllers functionally follow a generalized data flow mechanism, we will be using a single DFD as depicted in Fig. 2 for all four controllers. Even though they follow standard architecture and similar DFD, each controller has achieved a different level of maturity for each of the security properties, hence four different STRIDE threat matrices will be created and compared.

IV. Security Analysis

From Fig. 2 it is clear that there are several ways the data flow to/from a controller which could be exploited by an intruder to compromise the security of a controller. As described in [17], data flows between two controllers shall face multiple security issues including inter-controller trust, and inter-domain trust when controllers are placed in a different domain. In this work, a single controller model is used for the analysis. The security analysis of multi-controller and hierarchical controllers are out of the scope of this paper.

The controllers have a set of processes which implement and perform important networking functions such as topology management, load balancing, authentication mechanism, access control, etc. Security analyses of these processes are crucial in order to investigate the security of a controller.

Since all of the three layers of an SDN are fully decoupled, there is a need of an external medium or channel for communication between these layers. Communication channels between a switch and a controller or between an application and a controller are prone to well known threat categories like tampering and information disclosure. Hence, to verify the integrity and confidentiality of information exchange, data flows will be considered in the analysis.

The core process of a controller should be flexible and free of DoS attack. This could be achieved with a modular structure in which individual modules are responsible for certain functionality and the main process is offloaded. However, in designs where the core process of a controller acts as a single monolithic module, it becomes a bottleneck, and if not handled properly, may lead to a DoS attack.

During the threat analyses, a STRIDE threat matrix for each of the controllers has been created. In the matrix, three symbols are used: a capital x (“X”) is used when a threat exists, a capital m (“M”) is used when a mitigation mechanism exists, a minus sign (“-”) is used when no mitigation information is available in the studied materials such as research papers and websites, and a blank space means it is not necessary to check according to the STRIDE method.

A. OpenDaylight Controller

The OpenDaylight community has emphasized more on the security aspects in their recent release (Lithium-SR3).

Spoofing of identity is not viable in ODL because the Authentication, Authorization and Accounting (AAA) service is embedded in the controller platform. This module controls access from applications and users of north bound to controller resources. AAA is implemented as a token-claim based authentication. In case of direct authentication, the controller checks for the credentials provided by the user and returns the token upon successful authentication. In case of federated authentication where a third party identity provider authenticates the user, the controller provides a token to the user upon receiving status of successful authentication from the third party application. This token needs to be used by user applications to access controller resources, which prevents data access by an unauthorized user. Spoofing or impersonating an identity of a switch could not be achieved in ODL because the Secure Network Bootstrapping Infrastructure (SNBI) module handles the switch authentication process by checking the

certificate of an individual switch. In case of host tracking [18] where an already connected host may choose to move to a different switch and requests that the traffic should be redirected to the new location, the controller redirects the traffic to the new location of the host. If only the MAC address is used by the HostTracker service to keep track of all the hosts connected, an intruder can impersonate as an authentic host by learning the MAC address of the host [19]. Since the movement of a host is solely tracked by using a PACKET_IN message without any authentication, an authentic host may experience DoS or degraded performance resulted by packet loss. This anomaly is handled in ODL by using a combination of MAC, VLAN ID, IP and Location address [20] as an index in the Device Manager service.

An accounting method of the AAA module records access request made by an user or an application [21]. These logs are recorded for billing, analysis, diagnostics and security audit service purposes which effectively mitigates **repudiation** in ODL. For static elements like switches and users, the AAA module will mitigate the risk by checking the authenticity prior to allowing them to communicate with the network or making any changes in the data. However, ODL has no specific access control mechanism for dynamically created SDN elements like tunnels, VLAN etc., this is an area which needs to be addressed for AAA integration.

ODL was vulnerable to XML eXternal entity attack (XXE) [22] in the SBI when the NETCONF protocol was used which led to **information disclosure** of the (configuration) files located in the controller. However, this issue has been addressed within the recent release by removing Document Type Definition (DTD) completely from the requests.

Both **DoS** and **elevation of privileges** threats are tackled in ODL by the Defense4All and AAA modules. The Defense4All module is able to detect and mitigate different types of network attacks including DoS to its NBI, SBI, processes, and data store. The AAA module ensures that only authenticated users get access to the resources of the network. Due to strict access control, ODL processes can only be accessed by highly privileged (root) users.

The data flows in the south bound interface for the communication between switches and the controller are secured with TLS protocol, similar protection is achieved in the north bound interfaces for the communication between applications and REST APIs with HTTPS protocol (which use TLS protocol). This secures the channel against threats like **tampering** and **information disclosure**.

From the observation, Table II has been created.

B. ONOS Controller

For enhancing security, ONOS provides an implementation of a conservative-mode access model in the latest release called Drake [23]. Security-Mode ONOS (SM-ONOS) [24] requires auditing of north-bound applications prior to their deployment, and constant access monitoring of applications during run-time.

Type	Components	S	T	R	I	D	E
Data Flow	North Bound Interface		M		M	M	
	South Bound Interface		M		M	M	
Process	Controller Core	M	-	M	-	M	M
Data Store	Internal data structures		-		-	M	
Interactors	Switches and North Bound Applications	M		M			

TABLE II
STRIDE THREAT MATRIX FOR OPENDAYLIGHT CONTROLLER. M INDICATES MITIGATION MECHANISMS EXIST, - INDICATES NO INFORMATION PROVIDED, BLANK MEANS NO NEED TO CHECK AS THE COMPONENT IS NOT AFFECTED BY THE THREAT

Unlike ODL, ONOS has no additional security mechanisms applied for HostTracking mechanism, attacker can achieve topology **spoofing** by exploiting a host tracking mechanism [20] [25] with the help of an already learnt MAC address of the host.

Information Disclosure is prevented in ONOS with the use of SM-ONOS, which provides a mechanism to grant fine grained access permission to different applications and users [26]. The user role could be limited to an application user or may play a role of a root user. Application access also has been classified into different categories. Applications such as auditing/monitoring may have only READ access to controller resources. Applications such as IDS may have READ_WRITE access. Applications can have EVENT access enabled then receives network events such as PACKET_IN, and FLOW_REMOVED etc. These permissions are requested by applications during registration phase and access grants are approved by the network administrator.

Unauthorized access to ONOS resources are tracked and blocked at run-time by SM-ONOS which effectively handles the **elevation of privileges** threat.

Similar to ODL, the **information disclosure** and **tampering** threats in both SBI and NBI are handled in ONOS by enabling TLS in SBI and HTTPS in NBI.

Unauthorized change of internal data-structures are prevented with fine grained access control to internal libraries which is managed by SM-ONOS.

Proactive measures such as strict access control and security audit service in addition to logging prevents **repudiation** of identity of ONOS interactors and processes.

DoS on ONOS could be achieved through data plane or SBI. When ONOS encounters a malfunctioned packet or a Jumbo Ethernet frame [27], it throws an exception which is currently caught and logged. This is a potential issue which could result in disconnection of the switch that sent the packet. This issue is being tracked in [28], however, this should still be mitigated with the help of proper exception handling. The existence of this exception might lead to a DoS attack.

A summary of these observation has been recorded in Table III.

Type	Components	S	T	R	I	D	E
Data Flow	North Bound Interface		M		M	-	
	South Bound Interface		M		M	X	
Process	Controller Core	-	-	M	M	X	M
Data Store	Internal data structures		M		-	-	
Interactors	Switches and North Bound Applications	X		M			

TABLE III
STRIDE THREAT MATRIX FOR ONOS CONTROLLER. X INDICATES POSSIBLE EXISTENCE OF A THREAT, M INDICATES MITIGATION MECHANISMS EXIST, - INDICATES NO INFORMATION PROVIDED, BLANK MEANS NO NEED TO CHECK AS THE COMPONENT IS NOT AFFECTED BY THE THREAT

Type	Components	S	T	R	I	D	E
Data Flow	North Bound Interface		M		M	-	
	South Bound Interface		M		M	X	
Process	Controller Core	M	M	M	-	X	M
Data Store	Internal data structures		M		-	-	
Interactors	Switches and North Bound Applications	X		M			

TABLE IV
STRIDE THREAT MATRIX FOR ROSEMARY CONTROLLER. X INDICATES POSSIBLE EXISTENCE OF A THREAT, M INDICATES MITIGATION MECHANISMS EXIST,- INDICATES NO INFORMATION PROVIDED, BLANK MEANS NO NEED TO CHECK AS THE COMPONENT IS NOT AFFECTED BY THE THREAT

C. Rosemary Controller

Unlike ONOS and ODL, Rosemary uses a micro-NOS architecture to allocate resources and services. In micro-NOS architecture, each service is executed as a different process with an individual set of data structures and address space. The interactions between two services take place with the help of an API. An application communicates with only those services from which it consumes the service. The STRIDE analysis of Rosemary is as follows:

An SDN application can not spoof from NBI in Rosemary due to its micro-NOS architecture where no application can access micro-NOS without proper role assignment which makes processes safe against **spoofing**. However, as with ONOS, additional mechanisms are required to protect the interactors from host tracking.

Repudiation of identity could not be achieved in Rosemary because of its logging and auditing service.

DoS could be achieved through SBI of Rosemary by exploiting host tracking, which makes the process vulnerable for attacks.

Similar to ODL and ONOS, **tampering** and **information disclosure** of data flows in both NBI and SBI are protected using TLS and HTTPS protocols, respectively. **Elevation of privileges** is unlikely in Rosemary due to its micro-NOS permission structure, access to internal libraries is denied if the application or user has no permission to do so. This also prevents any unauthorized modification of internal data-structure where by effectively mitigating tampering of data in the controller core. However, fine granular access management could result in a noticeable hit in performance as discussed in [8]. The STRIDE matrix for Rosemary has been created as in Table IV.

D. Ryu Controller

Ryu has a monolithic architecture where the core process executes as a single process and each application and service will execute in different threads. All these threads share single process context. The controller core has been implemented with the help of observer [29] patterns. Every application subscribes the events it is interested in, for example, PACKET_IN, FLOW_MOD, FLOW_REMOVED etc. Once an event occurs, the event handler thread distributes the events to applications based on their subscription. An application can subscribe to multiple such events.

Spoofing is very likely in Ryu due to its lack of authentication mechanism in both NBI and SBI. A switch can send a PACKET_IN message to a port on which the controller is listening without any authentication. An application from NBI needs no authentication check before sending OpenFlow messages to switches, but require only a reference to a datapath which could be acquired by executing get_all() function call by an Ryu application [30].

In the SBI interface, the **tampering** and **information disclosure** threats can be tackled in Ryu by setting up and enabling a TLS connection [31]. By integrating the Snort module, an intrusion detection and prevention system, on top of the Ryu controller or as a separate machine [32], the SBI of Ryu can be protected from the DoS threat.

However, the process remains largely exposed to tampering. A user program is executed in a process context by design, hence an intruder can easily tamper flow tables on switch by sending PACKET_OUT and FLOW_MOD messages. The Ryu controller has no mechanism by which it can check the authorization of the user or an application before sending the above mentioned messages to switch.

DoS of Ryu has been shown in [8], execution of an exit(1) command from any of the threads in Ryu could result in the entire system being stopped. There is no access control implementation in Ryu which prevents such command execution, even an user application can issue the exit command.

Elevation of privileges is very likely in Ryu because no application is audited for the access privilege, an application thread may cause a system crash by over-utilizing the controller resources.

A **repudiation** of identity threat is still persistent in Ryu from NBI applications because authentication and logging are not done, however it has been mitigated in SBI with a logging

mechanism which records the data path id (dpid) of individual messages received [33].

Information Disclosure is feasible from a data flow between the controller and NBI since the channel is not encrypted and uses only HTTP instead of HTTPS [34]. This data flow is vulnerable to Man-In-The-Middle (MITM) attacks. An intruder can avail all information from the traffic between controller and a north bound application with a simple ARP spoofing and MITM attack. When the communication channel is not encrypted, an intruder can also modify the content of packets. This supports the claim that NBI is vulnerable to data **tampering**, too.

Table V provides a STRIDE matrix for the Ryu controller.

Type	Components	S	T	R	I	D	E
Data Flow	North Bound Interface		X		X	-	
	South Bound Interface		M		M	M	
Process	Controller Core	X	X	-	-	X	X
Data Store	Internal data structures		-		-	-	
Interactors	Switches and North Bound Applications	X		X			

TABLE V
STRIDE THREAT MATRIX FOR RYU CONTROLLER. X INDICATES POSSIBLE EXISTENCE OF A THREAT, M INDICATES MITIGATION MECHANISMS EXIST, - INDICATES NO INFORMATION PROVIDED, BLANK MEANS NO NEED TO CHECK AS THE COMPONENT IS NOT AFFECTED BY THE THREAT

V. REVIEW AND RECOMMENDATIONS

From the above observations, we could draw a conclusion that no controller is completely secure from threats and free from vulnerabilities. It is also worthy to note that secure modes which are available in few of the controllers are optional to be used, this flexibility was given to achieve higher performance. After careful review, we have the following findings; ONOS is vulnerable to known security threats and the ONOS team has recently started focusing on these aspects. The multi-threaded architecture of Ryu may lead in DoS because of lack of authorization for the addition of flow rules and event consumption. Rosemary handles resource access in an effective manner with the help of fine granular access control mechanism. However, in the mean time, creating multiple micro-NOS instances for every application is an overhead on performance, especially when strict security mode is enabled mainly due to Inter process communication (IPC).

We recommend to prefer OpenDaylight over other controllers for three major reasons: firstly, the code is continuously checked in terms of security vulnerabilities and improved by the security experts as it is an open-source project, secondly, the modular structure of ODL has more performance benefit over other studied controllers, thirdly, ODL has provided security mechanisms both for NBI and SBI including AAA and SNBI. fourthly, ODL is supported by a security team that is continuously monitoring and solving open and new vulnerabilities which may be exploited by the attackers as shown in Table VI. Lastly, Defense4All module of ODL is a first step towards preventive security measures.

Security Threats	Reasons	Vulnerable Version	Updated Version
DoS	SQLite memory corruption	All releases of Helium	Lithium GA / Helium SR4
MITM/LOGJAM	TLS connections which support export grade DHE key-exchange	All releases of Helium	Lithium GA / Helium SR4
Information disclosure	The ODL MD-SAL API docs did not require authentication	All releases of Helium	Helium SR4
Remote code execution	Insecure deserialization	Not affected	Not affected
Remote attack to access interfaces such as the northbound neutron AP	The authentication mechanism of karaf-tomcat could authenticate any username and password combination	All releases of Helium	Helium SR3
Topology spoofing via LLDP	Attacker can inject packets and disrupt the flow of data	All releases of openflowplugin	Helium SR3
Topology spoofing via hosttracker	User can update host location without any validation, authentication or authorization	All releases of l2switch	Patch build under progress
Information disclosure	Defense4all: Users can request report data be exported on a servers file system	All releases of Defense4All after 1.1.0	Builds of Defense4All on or after 15 Jan 2015
Remote attacker can exfiltrate files on the ODL controller	XML eXternal Entity (XXE) vulnerability	OpenDaylight Helium GA and SR1	Helium SR1.1

TABLE VI
SECURITY VULNERABILITIES AND UPDATES OF OPENDAYLIGHT CONTROLLER [SUMMARIZED FROM [35]]

Since no controller implements all of the security features, we recommend few best practices which we think are necessary for controller security. This could be adapted to design an approach to secure an SDN controller.

A. Isolation

Isolation of the user process context from the core context is very important to achieve role based security for individual

layers. In current controller architectures, except Rosemary, each one uses a single context for execution. For a secure system, it is necessary to have a clear separation between these two layers.

B. Monitoring of resource

As mentioned in [8], a single malicious process can lead to DoS of an entire SDN network. Malicious entities could be found in both NBI and SBI, hence a controller should have distributed monitoring for SBI and a centralized monitoring for NBI. For SBI, monitoring could be pushed on to data path for an early detection and load distribution purposes.

We are in favor of such a design after a due consideration of many of the available monitoring applications like PayLess [36], OpenSample [37], and DREAM [38]. Current monitoring techniques are inefficient because either they are sampling based or passive, in both cases there will not be any proactive measure in case of resource crunch and anomaly detection. There is a need of an active monitoring with real time information analysis to detect if there is any abnormality in the system or in the traffic.

C. Intrusion detection system

We suggest an extension system of the above-mentioned monitoring since they might not be able to detect the whole range of network attackers. There is a need of a system which could monitor malicious traffic or system upon notified by a monitoring agent and neutralize the same upon detecting the anomaly. For example, Defense4All in ODL, Snort in Ryu, or OrchSec [39] (which is controller independent). In this case, traffic is diverted to a dedicated component which later makes a decision.

D. Secure communication channel

Even if the communication entities are secured but the channel is vulnerable, there are greater chances of compromising messages in terms of tampering, DoS, and information disclosures. We emphasize on mandatory usage of secure channels such as recent version of TLS.

E. Safe mode or partial restart

This proposal is mostly for recovery purposes than prevention of attack. Since no system is completely secured against all of the threats, it is important to have a faster recovery mechanism when there is an attack. It becomes critical in case of the controller due to its importance in the network management. When the security of a system is compromised and requires a restart, there should be a mechanism to bring only critical systems up as a first phase of recovery followed by non-critical components, this is currently supported in Rosemary.

F. Conflict resolver

This feature is mainly necessary for a system if it participates in a multiple-domain or multiple-controller network. There could be a conflict of trust between switches, domains, and controllers. Currently, ODL and ONOS have a flow conflict resolver based on the user role. If there is a flow conflict between two users, then a user with higher privilege overrides the other change. However, this needs to be further extended to controller level conflicts. FlowVisor [40] is one such controller which makes use of network virtualization to share physical resources among isolated networks which may have been controlled by different controllers.

VI. Related and future work

Apart from the work mentioned in this paper, the STRIDE framework has also been used to accomplish security analyses of SDN SBI protocols including OpenFlow, OF-Config, and OVSDB [41]; SDN architectures including PCE, 4D, and SANE [42]; SDN security applications including OpenFlow Random Host Mutation and Resonance [43]; SDN applications to integrate middleboxes such as SIMPLE and OpenNF [44], SDN applications for monitoring and measurement including sFlow and BigTap [45], and SDN wireless and cellular applications such as OpenRadio and SoftRAN [46]. All of these work not only unveil potential security threats but also provide suggestions to tackle those threats using existing well-defined mechanisms. Future work will be to verify the threats using penetration testing with or without considering the suggestions. All of these work fall into the category of security-centric SDN, additionally, network security can be improved by using SDN such as OrchSec [39] and AutoSec [47] architectures.

VII. Conclusion

Controller security is crucial in SDN and the design of controllers has been changing significantly to fulfill the needs of security, performance, and scalability. Four major controllers have been analyzed in this paper by considering their latest releases. We have also analyzed and compared some of their security features.

It is important to note that an industrial level controller will have to showcase a balancing act between security, performance, and resilience. Performance is one of the main reasons that strict security policies are not being used by users of ODL and ONOS. ONOS has incurred 5%-20% additional overhead when used with strict security policies [24] [26]. Rosemary has shown 20%-30% decrease of throughput in secure mode [8]. Controller which strikes the right balance between performance and security wins the race.

As with any other hardware and software, SDN controllers should be updated in a regular basis to keep them secure.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, "Simple Network Management Protocol (SNMP)," 1990.

[3] J. Postel and J. Reynolds, "TELNET PROTOCOL SPECIFICATION:RFC854," 1983.

[4] D. J. Barrett and R. E. Silverman, *SSH, The Secure Shell: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

[5] R. R. Krishnan and N. Figueira, "Analysis of data center SDN controller architectures: Technology and business impacts," in *ICNC*, pp. 104–109, IEEE Computer Society, 2015.

[6] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *WoWMoM*, pp. 1–6, IEEE Computer Society, 2014.

[7] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, (New York, NY, USA), pp. 1–6, ACM, 2014.

[8] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A Robust, Secure, and High-performance Network Operating System," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, (New York, NY, USA), pp. 78–89, ACM, 2014.

[9] D. LeBlanc, "The STRIDE Threat Model." "https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx". Accessed : 2016-02-08.

[10] A. S. Sodiya, S. A. Onashoga, and B. A. Oladunjoye, "Threat modeling using fuzzy logic paradigm," Dec. 04 2013.

[11] D. Xu and K. E. Nygard, "Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets," *IEEE Trans. Software Eng*, vol. 32, no. 4, pp. 265–278, 2006.

[12] D. P. Gilliam and J. D. Powell, "Integrating a Flexible Modeling Framework (FMF) with the Network Security Assessment Instrument to Reduce Software Security Risk," in *WETICE*, pp. 153–160, IEEE Computer Society, 2002.

[13] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-Based Modeling Language for Model-Driven Security," *Lecture Notes in Computer Science*, vol. 2460, pp. 426–??, 2002.

[14] D. LeBlanc, "DREADful." "http://blogs.msdn.com/b/david_leblanc/archive/2007/08/13/dreadful.aspx". 2016-02-08.

[15] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. M. Parulkar, "ONOS: towards an open, distributed SDN OS," in *HotSDN* (A. Akella and A. G. Greenberg, eds.), pp. 1–6, ACM, 2014.

[16] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced Study of SDN/OpenFlow Controllers," in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR '13, (New York, NY, USA), pp. 1:1–1:6, ACM, 2013.

[17] H. Song, "SDN Threat Analysis." "https://www.ietf.org/proceedings/93/slides/slides-93-sdnrg-0.pdf", 2015. Accessed : 2016-02-08.

[18] Z. Zhu, H. Li, K. Pan, C. Yu, F. Chen, and D. Li, "Centralized Flat Routing," in *Computing, Management and Telecommunications (ComManTel), 2014 International Conference on*, pp. 52–57, IEEE, 2014.

[19] D. Jorm, "Linux - a small security-enhanced Linux distro for servers." "http://www.openwall.com/lists/oss-security/2015/03/20/3". Accessed : 2016-02-08.

[20] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures," in *Proceedings of 2015 Annual Network and Distributed System Security Symposium (NDSS'15)*, February 2015.

[21] S. Scott-Hayward, "Design and deployment of secure, robust, and resilient SDN controllers," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pp. 1–5, April 2015.

[22] "XML external entity attack." "https://en.wikipedia.org/wiki/XML_external_entity_attack". Accessed : 2016-02-08.

[23] B. O'Connor, "Drake Release ." "https://wiki.onosproject.org/display/ONOS/Release+Notes+-+Drake+1.3.0". Accessed : 2016-07-01.

[24] P. Joshi, "Security-Mode ONOS." "https://wiki.onosproject.org/display/ONOS/Security-Mode+ONOS". Accessed : 2016-07-01.

[25] D. Jorm, "SDN and Security." "http://onosproject.org/2015/04/03/sdn-and-security-david-jorm/". Accessed : 2016-07-01.

[26] C. Yoon, "Security-Mode ONOS." "https://wiki.onosproject.org/download/attachments/2132881/SMONOS_ONS_2016_ONOS_MiniSummit.pdf?version=1&modificationDate=1462250542998&api=v2". Accessed : 2016-07-01.

[27] P. R. Lutui and S. Narayan, "TCP/IP Jumbo Frames Network Performance Evaluation on A Testbed Infrastructure," *International Journal of Wireless and Microwave Technologies, Vol 2, Iss 6, Pp 29-36 (2012)*, vol. 2, no. 6/, pp. 29–36, 2012.

[28] K. Thimmaraju, "Denial of Service in ONOS with Jumbo Frame." https://jira.onosproject.org/browse/ONOS-3349, 2015-11-15.

[29] C. Köppe, "Observations on the Observer Pattern," in *Proceedings of the 17th Conference on Pattern Languages of Programs*, PLOP '10, (New York, NY, USA), pp. 6:1–6:14, ACM, 2010.

[30] "The First Application ." "https://ryu.readthedocs.io/en/latest/api_ref.html". Accessed : 2016-07-07.

[31] "Setup TLS connection for the Ryu controller." "https://ryu.readthedocs.io/en/latest/tls.html". Accessed : 2016-07-04.

[32] "Snort integration for the Ryu controller." "https://ryu.readthedocs.io/en/latest/snort_integrate.html". Accessed : 2016-07-04.

[33] "RYU:invoking application and Configuration." "https://github.com/osrg/ryu/blob/master/doc/source/parameters.rst". Accessed : 2016-07-07.

[34] "RYU SDN Framework." "https://osrg.github.io/ryu-book/en/Ryubook.pdf". Accessed : 2016-07-07.

[35] "Security Advisories." "https://wiki.opendaylight.org/view/Security_Advisories". Accessed : 2016-07-01.

[36] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," in *Presented as part of the 10th USENIX Symposium on NSDI 13*, pp. 29–42, 2013.

[37] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "OpenSample: A low-latency, sampling-based measurement platform for commodity SDN," *SDistributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference*, pp. 228–237, Aug. 2014.

[38] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "DREAM: Dynamic Resource Allocation for Software-defined Measurement," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 419–430, Aug. 2014.

[39] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "OrchSec: An Orchestrator-Based Architecture For Enhancing Network-Security Using Network Monitoring And SDN Control Functions," in *IEEE NOMS 2014*, May 2014.

[40] R. Sherwood, G. Gibb, K. kiong Yap, M. Casado, N. Mckeown, and G. Parulkar, "FlowVisor: A Network Virtualization Layer," tech. rep., Deutsche Telekom Inc. R&D Lab,Stanford University,Nicira Networks, 2009.

[41] M. Brandt, R. Khondoker, R. Marx, and K. Bayarou, "Security Analysis of Software Defined Networking Protocols OpenFlow, OF-Config and OVSDB," in *IEEE ICCE 2014, Special Session on SDN*, July 2014.

[42] D. Klingel, R. Khondoker, R. Marx, and K. Bayarou, "Security Analysis of Software Defined Networking Architectures PCE, 4D and SANE," in *ACM AINTEC*, November 2014.

[43] M. Tasch, R. Khondoker, R. Marx, and K. Bayarou, "Security Analysis of Security Applications for Software Defined Networks," in *ACM AINTEC*, November 2014.

[44] T. Eggert and R. Khondoker, "Security Analysis of Approaches to Integrate Middleboxes into Software Defined Networks," in *ICEEICT 2016*, September 2016.

[45] P. Dauer, R. Khondoker, R. Marx, and K. Bayarou, "Security Analysis of Software Defined Networking Applications for Monitoring and Measurement sFlow and BigTap," in *The 10th International Conference on Future Internet Technologies (CFI)*, June 2015.

[46] D. Magin, R. Khondoker, and K. Bayarou, "Security Analysis of OpenRadio and SoftRAN Using STRIDE Framework," in *The 24th International Conference on Computer Communications and Applications (ICCCN 2015)*, August 2015.

[47] R. Khondoker, P. Larbig, D. Senf, K. Bayarou, and N. Gruschka, "AutoSecSDNDemo: Demonstration of Automated End-to-End Security in Software-Defined Networks," in *IEEE NetSoft 2016*, June 2016.