Design of an SDN Security Mechanism to Detect Malicious Activities

Christopher Mansour
Department of Electrical and Computer Engineering
Villanova University
Villanova, Pennsylvania 19085–2128
Email: cmansour@villanova.edu

Danai Chasaki
Department of Electrical and Computer Engineering
Villanova University
Villanova, Pennsylvania 19085–2128
Email: danai.chasaki@villanova.edu

***Abstract*—Software Defined Networks (SDN) is a modern networking paradigm that introduces lots of benefits to the next-generation networks. It offers the necessary programmability that allows the dynamic configuration of the networks and thus the deployment of new services on the fly to meet the different use cases. This programmability allows the development of a plethora of third-party applications that can be used by network administrators in order to implement different networking functionalities. However, this programmability also induces various threats regarding potential malicious activities against the SDN networks. In this paper, we propose the design and implementation of an SDN security mechanism that helps in the detection of malicious activities that might target the SDN network. The design we are proposing leverages the benefits of centralized control and programmability of SDN in order to periodically monitor the system calls utilization of the different SDN applications installed and statistically correlates such information to the baseline/benchmark information, thus detecting the existence of an unusual activity or a malicious application.**

I. Introduction

Software Defined Networking (SDN) is a novel networking paradigm that facilitated the design and management of computer networks in the past few years. This technology did not appear suddenly, but it is a part and a fruit of efforts that tried to make the networks more programmable [1].

A. SDN Network Architecture

The SDN architecture relies on the following pillars:

- Decoupling of control and data planes: control is removed from forwarding devices which will become simple forwarding elements
- Forwarding decisions are flow based instead of destination based; where a flow is a sequence of packets between a source and a destination and all the packets of a flow receive identical service policies at the forwarding devices
- Control logic is moved to an external entity known as the SDN controller. The latter is a software platform (network operating system) that will run on general purpose processors and will provide the necessary abstraction to facilitate the programming of forwarding devices based on a logically centralized, and abstract network view
- Programmable network through software applications running on top of the network operating systems

SDN mainly consists of three main layers: the management layer which captures complex network functionality, the control layer which acts as the kernel module that runs the applications and the infrastructure layer which includes the data plane layer of traditional networks. This layered approach helped network administrators to quickly provision the network connections on the fly instead of manually configuring policies.

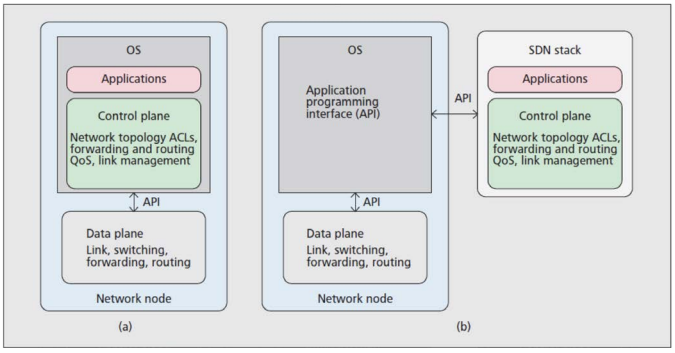

Fig. 1: (a) Traditional Computer Networks with Programmability, (b) SDN View With Control Plane Extracted (Figure retrieved from [2])

Figure 1 represents the transition from modern networks with traditional programmability to the SDN view of network. In the traditional networks with programmability shown in (a), the control plane is responsible for configuration of the node and programming the path to be used for data flows. Once the paths have been determined, they are pushed down to the data plane. Thus, the data forwarding is based on this control information. In such an approach, any modifications should happen through another device configuration [2].

In the SDN view shown in (b), the control is moved out of the individual network node into a separate centralized controller. The forwarding elements (routers, switches etc.) are controlled by the network operating system which collects information using the necessary API and manipulates their forwarding plane. The controller can thus benefit from the

network-wide visibility to optimize the flow management and implement many services [2].

B. Benefits of SDN

The development of SDN provides a lot of benefits and features such as: Dynamic Flow Control, Network-Wide Visibility with Centralized Control, Network Programmability, and a Simplified Data Plane [3].

1) Dynamic Flow Control: The data plane layer when receiving a packet that has no matching flow will request additional flow information from the control plane layer. Hence, a network application running on top of the SDN controller will be able to control the flow of the packets dynamically. This helps in many applications. For example a security application can benefit from the dynamic flow to separate the malicious flows from benign flows. Other application examples benefiting from the dynamic flow feature include: a load balancer application [4], and a network management application [5].

2) Network-Wide Visibility with Centralized Control: In the SDN architecture, the controller is logically centralized and hence sends control messages (flow rules and configurations) to the data plane layer. Additionally, the controller can request additional network status information by sending query messages. Hence, an application running on top of the SDN controller will have a complete network-wide visibility of the the underlying networking infrastructure and its corresponding status. Multiple SDN applications utilize this feature such as the work found in [6], [7], [8].

3) Network Programmability: Since the control logic is separated from the data plane, this offered an abstraction of the underlying infrastructure and thus lead to the potential of programmability. To empower this SDN feature, multiple programming languages have been proposed [9], [10].

4) Simplified Data Plane: This feature is evident in the SDN architecture and lead to the development of new data plane layer technologies such as the NETFPGA [11] which we use in one of our designs.

Therefore, SDN networks have the ability to enable innovative control through plethora of applications. These applications provide lots of services to facilitate the different use cases. However, such applications include a sophisticated code-base that is prone to a variety of bugs and vulnerabilities and thus exposing the SDN network to malicious activities [12].

With the emergence of SDN app-stores such as HP's SDN-store [13], the SDN applications deployed are most likely to be provided with limited testing and security checking from third-party entities. Therefore, attackers can exploit vulnerabilities existing in such applications turning them into malicious applications to be used to gain access to the network resources and manipulate the network operations [14]. Malicious application can sniff the network topology or even tap into the network disclosing confidential information to the attackers site, or can even modify a message that is supposed to be processed by another running application causing the latter to throw an unexpected exception. The reason such applications can be installed is that there is no guideline about malicious behaviors of SDN applications. Additionally, most SDN controllers do not adopt an inspection mechanism for SDN applications before executing them [15]. Furthermore, the monolithic design of the SDN controllers implies that a failure of any of one of the applications can render the entire control plane unavailable which is particularly dangerous and can cause a complete network shutdown [12].

However, when an application starts to act maliciously, the system calls characteristics utilized will vary from the normal expected behavior. Additional types of system calls might exist and the frequency of each type will change. Therefore, periodically monitoring the system calls of the SDN applications will help detect malicious activities.

That is why in order to detect the existence of malicious applications running in the SDN network, we propose the design of an SDN security mechanism that leverages the benefits of the centralized control and programmability that the SDN offers. The design proposed will monitor the system calls of the applications running in the SDN network periodically and compare such information with the benchmark/baseline expected behavior. The results of the latter comparison will determine the existence of an unusual activity.

The specific contributions of this paper include:

- The design of an SDN security mechanism that detects malicious applications/activities
- Periodically monitoring the system calls of the SDN applications and comparing the results against expected baseline thus detecting malicious applications
- Experimental measurement results from a prototype proof of concept implementation using an SDN framework based on mininet [16] emulator and Open vSwitch [17].

The remainder of the paper is organized as follows: section II discusses some related work. In section III, we describe the system calls and the way they are used in order to determine the existence of a malicious behavior. Evaluation and results of the proof of concept implementation are presented in section IV. Section V summarizes and concludes the paper with some future work.

II. Related Work

A. SDN Monitoring Techniques

With the holistic view that the SDN offers, several solutions were proposed to monitor and detect network attacks by collecting the network statistics.

In [18], the authors propose a flow-graph model learned from SDN messages to detect network level attacks on the network topology and the data plane forwarding. In [19] Braga et.al. suggests an application to monitor the network flows to detect network flooding attacks. Similarly, netfuse [20] was proposed in order to monitor the network and find suspicious flows.

In [21], the authors propose a monitoring framework which provides monitoring services for cloud networks. The framework they proposed detours network packets to be inspected

by pre-installed network security devices. In [22], the authors present a new concept of network security virtualization, which virtualizes security resources and functions to network administrators. This will maximize the existing middlebox utilization. In addition, it enables security protection to desirable networks with minimal management cost. In [23], Fayaz et.al. propose the design and implementation of Bohatei, which is a flexible and elastic DDoS defense system. In their design, the authors address key challenges with respect to responsiveness, and adversary-resilience.

The existing literature work focus on packet inspections and detecting security attacks. However, the design we propose in this paper does not perform any packet inspection. Instead, it leverages the benefit of centralized control and programmability to detect the existence of malicious applications.

B. Attacks against SDN Framework

Several studies exist in the literature that focus on the attack possibility against SDN. In [24] the authors argue that the benefits introduced with the development of SDN such as the centralized control and programmability has introduced a lot of attack vectors to the SDN networks. Christian et. al. [25] proposed an SDNRootkit which actively hides its malicious behaviors from the SDN controller and provide remote access. In [15] the authors reveal three attack scenarios that leverage the SDN applications to attack the SDN network. They test multiple attack scenarios against three of the most popular controllers. However, the authors did not offer a detection mechanism against malicious applications that might look benign and are later used by attackers to perform malicious activities.

In order to address the SDN faults and failures the authors in [26], introduce Ravana as a fault-tolerant SDN controller platform that processes the control messages transactionally and exactly once, thus enabling SDN applications to to execute in a fault tolerant fashion. In [27] and [12], Chandrasekaran et. al. address the reliability and fault tolerance of the SDN showing the possibility of controller and network failures due to SDN applications.

None of the previously mentioned work offers a security mechanism that detects the existence of malicious applications in an SDN network. The existing literature work focus on fault and failure tolerance of the controller. It is unknown whether they are able to detect malicious applications that will not fail but instead corrupt the services or even threatens the trustworthiness and reliability of the SDN network. However, our design leverages the centralized control and programmability offered by SDN in order to monitor the system calls characteristics utilized by the installed SDN applications in order to detect any malicious activity.

III. System Calls

System calls are used by the applications in order to request services from the operating systems. Each application has a specific number and order of system calls that it utilizes when performing an operation/task. Capturing and monitoring such information will help in determining the existence of a malicious and faulty application.

A. System Calls Monitoring

System calls monitoring is very useful in detecting malicious applications specifically those that are exploited to perform code injection attacks. The exploited application will utilize the system calls in a different manner than what is expected. The variation of system calls might be reflected with new calls, i.e. new type of calls, that have never been used before, additional number of particular existing function calls, or a different order of calls. Hence, the use of an additional function call, i.e. a call that has been never used before, is a clear indication of a malicious activity.

B. Monitoring Process

Figure 2 represents the monitoring system operation. The application to be installed should be analyzed thoroughly in order to extract: (1) the type of system calls utilized when running a particular task, and (2) their corresponding frequency. Such information will be used by the controller as a baseline in order to determine possible deviations during runtime and thus detecting malicious activities.

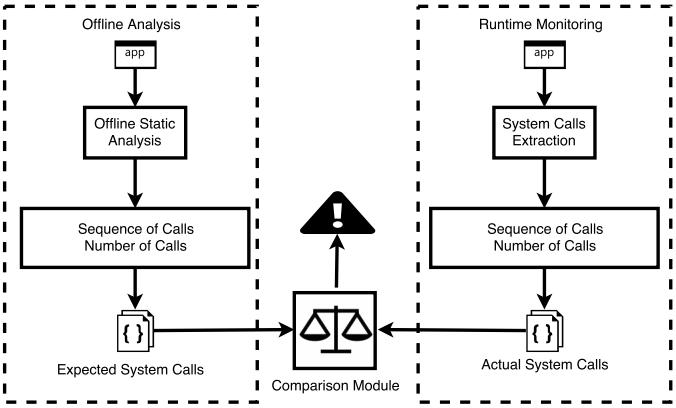

Fig. 2: System Architecture

In real-time, the controller will periodically poll the applications while running a particular task, will extract the type and frequency of the system calls involved, and then compare such information with the previously extracted baseline information to determine the existence of malicious activities.

IV. Results and Evaluation

In order to evaluate the efficiency of monitoring the system calls of applications, we present a prototype implementation, as a proof of concept, based on an SDN framework consisting of an Open vSwitch (OVS) and Floodlight controller.

A. Benchmark Extraction

In order to extract a baseline/benchmark to be used in the detection process, we run the experiment $N = 100$ different times without any attacks. We collect the different types of system calls and their corresponding frequency ft_i. For each

TABLE I: Sample System Calls

System Call	Set I	Set II	Set III	Mean (μ_t)	Stdevσ_t
write	120492	121115	120389	120792	420
select	12408	12186	12117	12301	212
mmap	12	12	12	12	0
read	12	12	12	12	0
open	5	5	5	5	0

TABLE II: Malicious Application System Calls

System Call	Values	$\mid f_t - \mu_t \mid$	$\geq \sigma_t$
write	123691	2899	**TRUE**
select	13408	1107	**TRUE**
mmap	12	0	FALSE
read	12	0	FALSE
open	5	0	FALSE

system call type t, we calculate the average μ_t and the standard deviation σ_t. The average μ_t will be used as a baseline and the standard deviation σ_t will be used as a threshold.

B. Detection Mechanism

During runtime, the system calls of the monitored application and their corresponding frequencies are collected. For each type t of system calls, the deviation from the mean, i.e. $\mid f_t - \mu_t \mid$ is calculated and compared to the standard deviation σ_t. Algorithm 1 describes the detection process.

Algorithm 1 Detection Algorithm

Require: Mean μ_t, Standard Deviation σ_t
1: Collect the frequency f_t of system call type t
2: Calculate the deviation $dev_t = \mid f_t - \mu_t \mid$
3: **if** $dev_t \geq \sigma_t$ **then**
4: $\quad ALERT \leftarrow TRUE$
5: **end if**

Table I illustrates three sample sets of system calls utilized by an SDN application acting normally. It also shows the overall average of each system call type and its corresponding overall standard deviation. As described earlier, the mean and standard deviation are used as baseline/benchmark information. Table II illustrates a proof of concept malicious application. As presented in table II, the deviation $\mid f_t - \mu_t \mid$ is greater than the expected standard deviation σ_t and therefore this application is flagged as malicious. It is important to note that one type exceeding the accepted deviation is enough to flag an application as malicious.

C. Failure Rate

In order to assess the efficiency of the prototype proof of concept implementation, the false negative rate was evaluated according to the following procedure:

- We run the experiment $N_n = 100$ times without malicious activity
- For each type t of calls, we denote the number of iterations whose deviation exceeded σ_t as $(K_t)_1$
- For each type t of calls, we denote the number of iterations whose deviation fell below σ_t as $(N_t)_1$
- We run $N_a = 100$ times attack experiments
- For each type t of calls, we denote the number of iterations whose deviation exceeded σ_t as $(K_t)_2$
- or each type t of calls, we denote the number of iterations whose deviation fell below σ_t as $(N_t)_2$

From the numbers evaluated above, it follows that the false negative rate can be calculated as described in equation 1.

$$falseNeg = \frac{(N_t)_2}{(N_t)_1 + (N_t)_2} \tag{1}$$

In our particular experiment, the false negative rate was determined to be 0.83%. The low rate is appealing because there lies the real danger of a potential attack not being detected.

V. Conclusion and Future Work

In this paper, we proposed the design and implementation of an SDN security mechanism that helps in detecting malicious activities in SDN networks. The proposed security framework leverages the centralized control and programmability of SDN and performs periodical monitoring of the system calls utilized by the SDN applications installed. Malicious applications when executing a malicious code or performing a different task will have different system calls utilization. We showed that by correlating the real-time system calls characteristics to previously extracted baseline/benchmark information, unusual activities such as attacks can be detected.

In the future, this design can be improved to become more robust. This can be achieved by combining it with another SDN monitoring framework that leverages the network-wide visibility and monitors other traffic characteristics such as packet interarrival time. Monitoring the interarrival time of packets, that is the time between a specific packet entering and leaving specific node, can provide sufficient information to infer a malicious activity.

Moreover, the proposed design can be improved to become smarter. The latter can be achieved by the introduction of machine learning. Machine learning can be utilized in two distinct ways: attack prediction or application classifier. Utilizing machine learning can help in outsmarting malware and thus perform attack prediction. The design can predict the likelihood of a malicious application by analyzing the sequence and frequency of the different system calls types. Another input to such a machine learning algorithm is the power consumption and processing characteristics utilized by each application. Combining all such features can help predict the existence of a malicious activity. Furthermore, machine learning can be used to build a robust classifier with high accuracy. Such classifier can be used in order to classify the applications that are vulnerable to become malicious from those that are not by performing behavioral analysis of each application.

References

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *Queue*, vol. 11, no. 12, p. 20, 2013.

[2] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[3] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing network security through software defined networking (sdn)," in *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*. IEEE, 2016, pp. 1–9.

[4] M. Qilin and S. Weikang, "A load balancing method based on SDN," in *Measuring Technology and Mechatronics Automation (ICMTMA), 2015 Seventh International Conference on*. IEEE, 2015, pp. 18–21.

[5] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *Communications (QBSC), 2014 27th Biennial Symposium on*. IEEE, 2014, pp. 71–75.

[6] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[7] S. U. Rehman, W.-C. Song, and M. Kang, "Network-wide traffic visibility in of@ tein SDN testbed using sflow," in *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*. IEEE, 2014, pp. 1–6.

[8] 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 *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[9] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *ACM Sigplan Notices*, vol. 46, no. 9, pp. 279–291, 2011.

[10] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with pyretic," *Technical Reprot of USENIX*, 2013.

[11] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA–an open platform for gigabit-rate network switching and routing," in *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, San Diego, CA, Jun. 2007, pp. 160–161.

[12] B. Chandrasekaran, B. Tschaen, and T. Benson, "Isolating and tolerating sdn application failures with legosdn," in *Proceedings of the Symposium on SDN Research*. ACM, 2016, p. 7.

[13] (2018) SDN app store. [Online]. Available: https://www.hpe.com/us/en/networking/applications.html

[14] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[15] S. Lee, C. Yoon, and S. Shin, "The smaller, the shrewder: A simple malicious application can kill an entire sdn environment," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. ACM, 2016, pp. 23–28.

[16] M. Team, "Mininet: An instant virtual network on your laptop (or other pc)," *Google Scholar*, 2012.

[17] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch." in *NSDI*, 2015, pp. 117–130.

[18] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks." in *NDSS*, 2015.

[19] 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*. IEEE, 2010, pp. 408–415.

[20] Y. Wang, Y. Zhang, V. Singh, C. Lumezanu, and G. Jiang, "Netfuse: Short-circuiting traffic surges in the cloud," in *Communications (ICC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3514–3518.

[21] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*. IEEE, 2012, pp. 1–6.

[22] S. Shin, H. Wang, and G. Gu, "A first step toward network security virtualization: From concept to prototype," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 10, pp. 2236–2249, 2015.

[23] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic ddos defense." in *USENIX Security Symposium*, 2015, pp. 817–832.

[24] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[25] C. Röpke and T. Holz, "Sdn rootkits: Subverting network operating systems of software-defined networks," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 339–356.

[26] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research*. ACM, 2015, p. 4.

[27] B. Chandrasekaran and T. Benson, "Tolerating sdn application failures with legosdn," in *Proceedings of the 13th ACM workshop on hot topics in networks*. ACM, 2014, p. 22.