Analysis of Policy-based Security Management System in Software-defined Networks

Keshav Sood, Kallol Krishna Karmakar, Vijay Varadharajan, Uday Tupakula, and Shui Yu

***Abstract*—In software-defined networks, policy-based security management or architecture (PbSA) is an ideal way to dynamically control the network. We observe that on the one hand, this enables security capabilities intelligently and enhance fine grained control over end user behavior. But, on the other hand, dynamic variations in network, rapid increases in security attacks, geographical distribution of nodes, complex heterogeneous networks etc., have serious effects on the performance of PbSAs. These affect the flow specific Quality of Service (QoS) requirements with further degradation of the performance of the security context. Hence, in this letter, PbSAs performance is evaluated. Key factors including number of rules and rule-table size, position of rules, flow arrival rate, and CPU utilization are examined, and found to have considerable impact on the performance of PbSA's.**

***Index Terms*—Performance analysis, SDN security, PbSA.**

I. Introduction

IN Software-defined Networking (SDN), policy-based security management[1] aims to find more intelligent ways to reestablish fine grained control over the network and user behavior [1]. Policy servers are, in effect, general-purpose rule (expression) engines in which a rule can, in principle, be written to achieve a desired outcome. This outcome may include not just generalized traffic management, but also flow-oriented security management, maintenance of Quality of Service (QoS), deep packet inspection, queuing mechanisms, load balancers etc. The aim of proposing policy based security management is to pick off fine-grained secure traffic flows with a high degree of visibility and transparency in order to identify end user meta-data at increasingly granular levels [1].

There are many policy-based solutions existing in SDN including Procera [2], CloudWatcher [3], Fresco [4], Frenetic [5], PolicyCop [6], OpenSec [7], PbSA [1], etc. Researchers have noted that existing solutions have some limitations such as granular control, security, and complexity in expressing the security rule [1], [7]. Very recently, the authors proposed a novel policy-based security architecture (PbSA) [1] to secure end-to-end service in an SDN enabled autonomous domain. The design and implementation of this novel architecture enables fine granular enforcement of security policies. We observe that the generic work-flow of most of the existing policy based security management proposals [1]- [7] is the same, i.e., the PbSA system stores different policy expressions or rules. These policy expressions are capable of implementing polices, checking the conflicts and enforcing required policy, accordingly. Furthermore, other work has been done to detect effectively the policy conflict or violations so that no two policy expressions have conflicts [8]- [10], however, this is an entirely new module to work on. We focus instead on the general performance analysis of policy based security frameworks [1]- [7].

To the best of our knowledge, existing solutions do not include a theoretical performance analysis such as the one proposed here. We emphasize that the evaluation of existing architectures is essential. Firstly, this helps in the design of PbSAs so as to constantly monitor network parameters, detect ongoing attacks and help in determining the best course of action without degrading optimal performance. Secondly, this evaluation helps to determine the volume of data that can be supplied into the PbSA without compromising the application's QoS requirements of flow. Further, the service time to check the flows cannot violate the specific application's QoS. This not only affects the QoS but actually allows reasonable time for attackers or hackers to access the network and perform malicious activities [11]. Finally, it can alleviate the issue of lookup speed in memory architectures of PbSA. The number of policies considerably affects the reaction time of the policy enforcement architecture to alert the associated network when the traffic rate increases [7].

These significant benefits motivated us to analyze the performance of the policy based security architectures. For our analysis, we picked a very recent proposed solution [1]. Our findings can be generalized and applied on any existing solution. This is because at a high level the work-flow of every policy based solution is the same, i.e., based on match-action sequence. More detail is given in Section II.

We used classical queuing model tools, i.e., $M/Geo/1$ analytical tool [11], [12] to evaluate the PbSA's behavior at a given flow arrival rate. Key factors such as: number of policy rules, rule matching probability of flow in policy-table, flow arrival rate, the impact of rule expressions on throughput, CPU utilization, and CPU heap memory are evaluated.

Our contributions are as follows:

1) The PbSA in SDN is evaluated and it is shown that the PbSA's performance is considerably affected by key factors, i.e., high flow incoming rate, number of policy expressions or rules, and rule position.
2) This early evaluation gives us insight into the design of policy based security management architectures in order to structure the size of the multi-domain networks. Structuring and prioritization of the rules in the PbSA provided QoS must be preserved.

[1]The terms policy-based management, policy-based security, and policy-based security architecture (PbSA) are used interchangeably in this letter.

II. A High Level View and Work Flow of Policy Server

In this section, the background of PbSA and its work flow is discussed to help readers to better comprehend the concept of PbSAs in SDN. A high level view of PbSA architecture is shown in Fig. 1. PbSA is a software-based application that runs over the SDN controller or can be a part of the controller [1]. The end hosts are connected to the forwarding devices (underlying network) which is then connected to the SDN controller. The SDN controller forwards the flow to the PbSA via the policy manager at PbSA. The Policy manager is responsible for coordinating every single operation in the network. This includes capturing information from edge devices, extracting the required information to match the security rule or policy expression, and more. A simplified policy expression or rule could be as follows:

$$PE =< FlowID, SourceAS, DestAS, SourceHostIP$$
$$DestHostIP, SourceMAC, DestMAC, User, FlowConst$$
$$DomConst, Services, Seq-path >:< Actions >$$

Here, PE indicates the policy expression. FlowID is the unique ID indicates that the sequence of the flow belongs to that ID. The range of attributes that a FlowID (or Flow ID tuple) comprises is: type of packets, security profile indicating the set of security services (example, secure routing, identifying certain types of attacks, etc.) associated with the packets in the flow, etc [1]. More attributes can be added in the tuple based on the requirements, and wild card rules are also accepted. Further, SourceAS and DestAS are the network domain attributes of source and destination respectively, SourceHostIP and DestHostIP are source and destination host attributes, SourceMAC and DestMAC are the MAC address of source and destination respectively, Flow constraints and Domain constraints are defined as FlowConst and DomConst associated with flow, services indicates to the particular service for which the policy expression applies, sequence path indicates the sequences of switches whereas with domain communicates with another domains. This template is an example and one can create it or any similar template according to the network and service requirements. Now, below, we have discussed the working of each sub-module.

Policy Manager. It coordinates different domains and forwards the flow to the state-collector sub-module. It contains policy repository and topology repository. The first contains the different policy rules which is written in a simple language-based template [13] (JSON Policy repository). The latter contains the network topology information derived using a trace out mechanism specified by [1].

Decision Maker. This is used to analyze the arriving network traffic against the existing security policy expression, stored in policy-repository, for a particular target. Whenever an exact policy-match is found, it conveys the associated rule to the decision enforcer.

Decision Enforcer. It enforces the decision on particular flow and also updates the state-collector and repository sub module.

From Fig. 2, when a flow arrives at the PbSA application, it needs to go through the rule-match engine or expression to find an existing policy expression. This lookup varies widely depending on many key factors which have an influence on PbSA's performance. In case the rule-table is not sorted, the flow needs to be checked against every existing rule in the search engine. However, the time required for the matching varies widely, depending on many key factors that include the number of tuples to be matched, the performance impact of multiple tables, etc. This evaluation is not a part of this article but nevertheless, it emphasizes the critical importance of investigating the PbSA's performance. When a flow arrives at PbSA and when a policy expression is satisfied, only then is the associated action performed, e.g., allow or deny the request. For example, $PE_{10} =< *, *, 172.56.18.01, 172.56.16.08, 38 : 2C : 6A : 1E : 60 : FF, *, *, , 80, Conf, Intg(SW1 : SW6 : SW10 >:< Deny >)$, indicates that the flow from host IP 172.56.18.01 and MAC 38:2C:6A:1E:60:FF accessing the HTTP server on IP 172.56.16.08, should be securely routed via. Switch $SW1- > SW6- > SW10$.

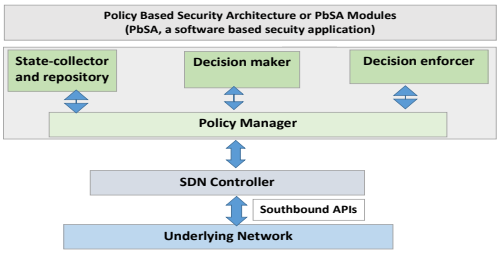


Fig. 1. A high level view of policy-based security framework.

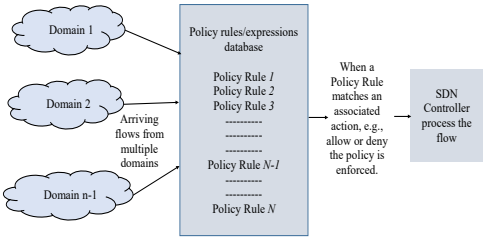


Fig. 2. Generalized work-flow sequence policy-based security framework.

III. Mathematical Model

SDN researchers assume that the controller service time obeys exponential distribution which is in line with the existing research work [12], [14]. However, PbSA's work-flow is different. The policy matching function must be considered.

We assume that there are N rules or policy expressions $F_i(i = 1, 2, 3, ...N)$ in the PbSA rule search engine as shown in Fig. 2. Upon arrival of a new flow, the PbSA goes through the policy expression table one by one in order to verify the existence of a rule with matching fields. The probability that a matching rule (of incoming flow) or policy *succeeds* at a particular rule is specified as F_i and p_i respectively. In such a case, the system has no prior information on flow matching probability distribution, thus, it is reasonable to assume that the rule matching probability of N rules is the same [15], i.e., $p_1 = ... = p_i = ... = p_N = p > 0$. Now, assume R is the

(a) (b) (c)

Fig. 3. (a) The influence of number of policy rules at a given flow arrival rate on average response time, (b) The rule matching probability and its impact on the mean response, time, (c) The incoming flow rate and the number of rules impact on PbSA's CPU utilization.

number of trails, to match rule, for the first match, then using geometrical distribution we obtain the matching probability at rule F_i as;

$$P_r[R = i] = \begin{cases} (1-p)^{i-1}p & 0 < i < N \\ \\ (1-p)^{N-1} & i = N. \end{cases} \quad (1)$$

Here, the geometric distribution's mean is calculated as

$$E[R] = \frac{(1-q^N)}{p}, \quad (2)$$

here, $q = 1 - p$, $p = 1/N$, and the service time is defined as

$$\bar{t_s} = \sum_{i=1}^{N} \left(Pr[R = i] \sum_{j=1}^{i} T_j \right), \quad (3)$$

we assume that T_j is the rule matching time for the j^{th} rule. We have mentioned that the N rules have the same matching time ($T_1 = T_2 = ... = T_N = T$). Therefore,

$$\bar{t_s} = \frac{T(1-q^N)}{p}. \quad (4)$$

Here, we observe that the service time, $\bar{t_s}$, is influenced by multiple factors, i.e., the rule numbers as well as the matching probability of the rule. Finally, using $\mu = 1/\bar{t_s}$ we can calculate the response time of PbSA to particular flow. We use $\mu = 1/\bar{t_s}$ in order to calculate the response time as our key metric of this performance evaluation.

From [11] and [12] we consider PbSA as an $M/Geo/1$ model where the incoming flows obey Poisson distribution and the service times obey Geometric distribution. Other work has shown that by comparing the average response times in the queue, the required service time by flow to match policy can be easily determined [11]. In this case, the Imbedded Markov Chain theory is helpful. We apply this in $M/Geo/1$ queue to find out the number of flows and the expended service time on them, respectively, in the system. We use the Pollaczek-Khinchin (P-K) formula from [15] and further have taken the key equations from [11] and [15] and rearranged as below.

$$\bar{r} = \bar{q}/\lambda = \bar{t_s} + \frac{\lambda T^2 E[R^2]}{2(1-\rho)}. \quad (5)$$

This concludes that the total time the flow spent in system is the average of system service time plus the average time spent in the queue. Another interesting factor for investigation is the PbSA's CPU utilization determined by $U = \lambda\bar{r}$.

IV. Experimental Results and Discussion

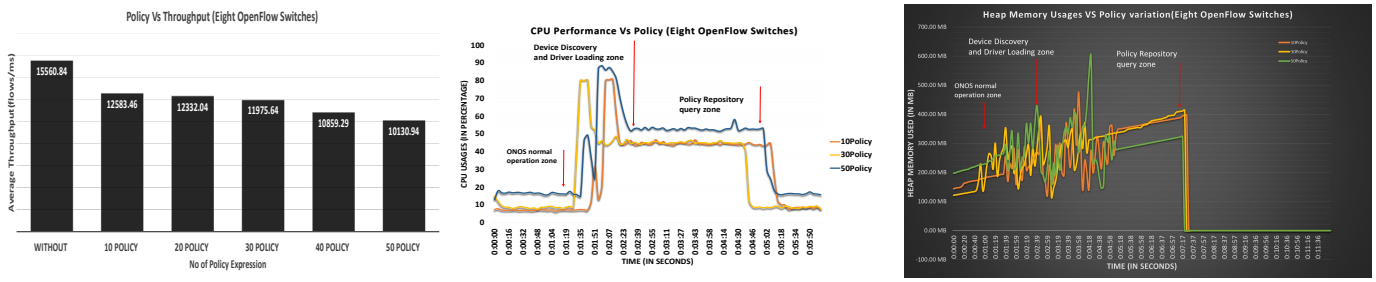
Now, the performance analysis and evaluation of the PbSA architecture is discussed. The average flow response time metric is used for PbSA performance evaluation. Firstly, we use Matlab to analyze the average flow response time, μ, we use $\mu = 1/\bar{t_s}$, here, $\bar{t_s}$ is taken from equation (4). Here, we set $N = 1,000 - 7,000$, $p = 1/N$, and $T = 27\mu s$. In existing studies, scientists determined that the time to match the incoming packet with the existing rule depends on the average service rate distribution, and experimentally proved that the time to match any rule must be in microseconds [11]. Researchers determined the mean rule matching time, at 10,000 rules, is nearly $27\mu s$ [11]. We thus reasonably set T as $27\mu s$ based on existing research conclusions.

In Fig. 3(a), we observe that more rules or policy expressions at PbSA degrade the service capacity (QoS), and also, PbSA takes more time to process the future arrival packets, i.e., more policy expressions impact the mean response time with respect to flow incoming rate λ. Thus, we emphasize that the rule matching at PbSA, for security check of the incoming flow, must be fast enough to maintain the security and application specified QoS.

The rule matching probability vs. average response time is shown in Fig. 3(b) at $\lambda = 70$. We observe that if the probability of matching any rule is high, the PbSA's response time to flow is less. This motivates PbSA designers to structure the rules in a rule search engine so that they match quickly. Thus, we suggest that the rule position significantly affects packet response time, however this is not viable in case a flow requires us to check every rule for overlaps regardless of position. This too affects the cost of updating, and modifying the rules.

Now, using $U = \lambda\bar{r}$, Fig. 3(c) depicts that these factors, the mean response time of flows at a given flow arrival, number and the position of rules, also have significant effect on the CPU utilization at a given packet arrival rate, targeting different rules. There is a corresponding increase in CPU utilization with the increase in targeting rules. The utilization is higher at $N = 1,000$ because the rules do not impact heavily on CPU processing requirements (low-priority processes consume the CPU power at this time). In contrast with $N = 3,000$ and $N = 7,000$ rules, the CPU is mainly busy processing policy expressions or rules.

Now, emulation results using Mininet are discussed. We used cBench in throughput-mode to measure the throughput of



(a) (b) (c)

Fig. 4. (a) The impact of the policy expressions on throughput, (b) The relationship between CPU performance and policy expressions (c) The relationship between CPU heap memory and number of policy expressions.

the controller. Each switch sends as many packet_in messages as possible to record the number of responses for the requests it has sent to the controller. We use Core i7−7700k @4.20GHz with 64 GB of RAM machine. The VMs are configured with 4 core CPUs and 12GB of RAM. In Fig. 4(a), we increase the policy rules with eight switches, connecting 500 dummy hosts in each case to an OpenFlow switch. As expected, the throughput decreases as we increase the policy expressions. The highest throughput is $12,583$flows/ms at 10 policy expressions. As the number of policy expressions increases, the throughput decreases. Thus, more policy rules at PbSA system increases the average response time to flow to get process at PbSA (Fig 3(a)). Also, it reduces the controller throughput because PbSA acts like another barrier before the flow actually passes on to the controller (Fig. 4 (a)).

In Fig. 4(b), we have used JProfiler to analyze PbSA's CPU load while running PbSA over an ONOS Controller at eight switches. We observe three phases. The first phase is the controller's normal operation phase in which the controller starts and loads its basic modules, such as DHCP applications and rule assistant. In the second phase, drivers are loaded and in the third phase, PbSA operates and makes the relevant decision. We observe that the CPU usage increases as we increase the policy expressions in the first and second phase.

Finally, Fig. 4(c) shows the heap memory usage again in three phases for a varying number of policy expressions. Initially, memory usage increases gradually, more rapidly in the second phase, and finally when PbSA carries out all security checks, this memory availability increases again. In the third phase (Fig. 4(c), PbSA operates on the requests from the devices and makes the relevant decisions. With PbSA, only permissible devices communicate, incurring less CPU and Heap usages, as shown in third phase. Further, OS maintain and monitor the heap memory usage and dumps the unused heap memories in a cyclic order. Therefore, some cyclic drops are evident in the graph.

V. Summary

PbSA's response time for a given packet arrival rate, number of policy expressions, and position of the rule by a corresponding packet is evaluated. This helps in the design and implementation of policy-based SDN architectures. Further, it expands our knowledge to allow us to structure the size of a domain with respect to the QoS requirements of the flows. For example, structuring the devices in a domain (with policy rules serving as a parameter in this structuring process) and also structuring the rules in the policy base (e.g prioritizing the rules in the policy base as per the response time). These results can significantly improve the design of policy based security architectures.

References

[1] V. Varadharajan, K. K. Karmakar, U. Tupakula, "Securing communication in multiple autonomous system domains with software defined networking," *Proceedings of IFIP/IEEE Integrated Network and Service Management Symposium*, 195-203, 2017.

[2] A. Voellmy, H. Kim, and N. Feamster, "Procera: A language for high-level reactive network control, *in Proc. Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, Aug. 2012, pp. 43-48.

[3] 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 Proc. 20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2012, pp. 1-16.

[4] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular composable security services for software defined networks, *in Proc. Netw. Distrib. Syst. Sec. Symp. (NDSS)*, Feb. 2013, pp. 1-16.

[5] N. Foster et al., "Frenetic: A network programming language, *in Proc. ACM SIGPLAN Int. Conf. Funct. Program.*, Sep. 2011, pp. 279-291.

[6] M. Bari, S. Chowdhury, R. Ahmed, and R. Boutaba, "PolicyCop: An autonomic QoS policy enforcement framework for software defined networks, *in Proc. IEEE SDN Future Netw. Serv. (SDN4FNS)*, Nov. 2013, pp. 17.

[7] A. Lara and B. Ramamurthy, "OpenSec: Policy-Based Security Using Software-Defined Networking," *IEEE Trans. on Networks and Service Management*, vol. 13, no. 1, pp. 30-42, 2016.

[8] S. Pisharody et. al "Brew: A Security Policy Analysis Framework for Distributed SDN-Based Cloud Environments," *in IEEE Transactions on Dependable and Secure Computing*, 2017.

[9] H. Hu et.al, "FLOWGUARD: building robust firewalls for software-defined networks," *ACM Proceedings of the third workshop on Hot topics in software defined networking, HotSDN'14,* pp. 97-102.

[10] A. Khurshid, et al., "VeriFlow: Verifying Network-Wide Invariants in Real Time, *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13),* 2013, pp. 15-27.

[11] M. Liu, W. Dou, S. Yu, and Z. Zhang, "A Decentralized Cloud Firewall Framework with Resources Provisioning Cost Optimization," *IEEE Trans. on Parallel and Distributed Systems,* vol. 26, no. 3, pp. 621-631, 2015.

[12] K. Sood, S. Yu, Y, Xiang, "Performance analysis of software-defined network switch using $M/Geo/1$ model", *IEEE Communications Letters*, vol. 20, no. 12, pp. 2522-2525, 2016.

[13] D. Clark, "Policy routing in internet protocols," request for comment rfc-1102," *Network Information center*, 1989.

[14] J. Hu, C. Lin, X. Li, and J. Huang, "Scalability of Control Planes for Software Defined Networks: Modeling and Evaluation," *Quality of Service (IWQoS), IEEE 22nd International Sym. of*, pp. 147-152, 2014.

[15] L. Kleinrock, *Chapter 5, The Queue $M/G/1$, Queueing System* Wiley Interscience, 1975, Vol. I: Theory..