Performance Oriented Dynamic Bypassing for Intrusion Detection Systems

Lukas Iffländer
University of Würzburg
Germany
lukas.ifflaender@uni-wuerzburg.de

Jonathan Stoll
University of Würzburg
Germany
jonathan.stoll@uni-wuerzburg.de

Nishant Rawtani
Hewlett Packard Enterprise
India
nishant.rawtani@hpe.com

Veronika Lesch
University of Würzburg
Germany
veronika.lesch@uni-wuerzburg.de

Klaus-Dieter Lange
Hewlett Packard Enterprise
USA
klaus.lange@hpe.com

Samuel Kounev
University of Würzburg
Germany
samuel.kounev@uni-wuerzburg.de

ABSTRACT

Attacks on software systems are becoming more and more frequent, aggressive, and sophisticated. In 2018, with the changing threat landscape, organizations are looking at 'when' they will be attacked, not 'if'. An Intrusion Detection System (IDS) can help in defending against these attacks. The systems that host IDS require extensive computing resources as IDS tend to detect attacks under overloaded conditions wrongfully. With the end of Moore's law and the growing adoption of the Internet of Things, designers of security systems can no longer expect processing power to keep up the pace. This limitation requires ways to increase the performance of these systems without additional computation power. In this work, we present two dynamic and a static approach to bypass IDS for traffic deemed benign. We provide a prototype implementation and evaluate our solution. Our evaluation shows promising results. Performance is increased up to the level of a system without an IDS. Attack detection is within the margin of error from the 100% rate. However, our findings show that dynamic approaches perform best when using software switches. The use of a hardware switch reduces the detection rate and performance significantly.

CCS CONCEPTS

• **Security and privacy** → **Intrusion detection systems**; **Virtualization and security**; • **Networks** → **Network control algorithms**; *Network performance analysis*; *Network security*; Middle boxes / network appliances;

KEYWORDS

intrusion detection, software-defined networking, network function virtualization, adaptive networking

ACM Reference Format:
Lukas Iffländer, Jonathan Stoll, Nishant Rawtani, Veronika Lesch, Klaus-Dieter Lange, and Samuel Kounev. 2019. Performance Oriented Dynamic Bypassing for Intrusion Detection Systems. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3297663.3310313

1 INTRODUCTION

In recent years, the number of services running in a cloud has grown exponentially. The development of tools for the simple use of Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) has made it easier to outsource existing services to the cloud environment. In addition to reducing operating costs, scalability, flexibility, and availability are vital reasons for using the cloud or hybrid cloud solutions.

Forecasts predict that the size of the hybrid cloud market will grow from $25.28 billion in 2014 to $91.74 billion by 2021 [1]. Since IT systems are part of the critical infrastructure for many companies and organizations, a cloud must have essential security features. In surveys, 90% of the interviewed companies said they had security concerns about moving to cloud systems [15]. Of all cloud users, 58% stated that security aspects had been a significant challenge when migrating to the cloud [8]. Thus, security concerns far outweighed any other concerns.

With the spread of the cloud and the increasingly sophisticated attacks via the Internet, there is a need for defensive measures to evolve. Not only are the software stack and the computers hosting this software stack targets of these attacks, but these attacks also target the complete hardware infrastructure such as storage and networks. With the further development of existing security systems, it seems that future threats can be averted no longer. The next logical step will be to combine security systems with emerging technologies such as Software-Defined Networking (SDN) and the Network Function Virtualization (NFV), as they are currently entering cloud computing centers [10].

In addition to firewalls, Intrusion Detection Systems (IDS) have become a security standard for data centers. The constant work of security researchers and the community ensures a regular surge of new signatures for IDS to defend against attacks. However, the use of IDS is proving inflexible for cloud solutions, which must react to new requirements within the shortest possible time. As

SDN and NFV solutions are increasingly finding their way into data centers, the question arises as to whether and how active cooperation between these systems can contribute to further gains in performance and security.

To evaluate the potential of SDN and NFV solutions, this work comprises the following **contributions**. We design SDN-based algorithms for handling network traffic including detection of attacks by the IDS:

Adaptive Blacklisting: Traffic of specific protocols reaches the IDS for a defined time interval. If the traffic of a considered connection triggers an alarm in this interval, it becomes permanently redirected via the IDS. Other traffic continues without redirection.

Adaptive Whitelisting: Only whitelisted protocols pass directly to the destination. The remaining traffic passes through the IDS. If the traffic of a specific connection does not trigger an alarm after a certain number of transmitted packets, the traffic of this connection is whitelisted.

Selective Filtering: Incoming traffic of selected protocols is permanently redirected via the IDS and only then forwarded to the original destination. Outgoing and other incoming traffic usually is switched to the destination without the detour via the IDS.

We implement these algorithms as an SDN-Controller App. We perform multiple experiments inside a testbed environment to measure the performance and security metrics throughput, delay, and detection rate of attacks of the IDS under consideration of the attack detection. We evaluate and discuss the results of the measurements during the experiments.

The results are promising. The dynamic approaches can remove a majority of the negative performance impact from the IDS. The detection accuracy remains high within the margin of error to 100% in most scenarios. However, this applies only when using a software switch. When applying the algorithms on a hardware switch, it reduces the improvements in performance. Furthermore, detection accuracy falls to extreme values. The Selective Filtering shows that with little effort, a static solution can improve performance. While it does not reach the performance of the other approaches, the hardware switch has only a marginal effect on the performance of Selective Filtering and does not reduce detection accuracy.

The remainder of this paper is structured as follows: At first, we introduce related work in Section 2 and the relevant technical background in Section 3. In Section 4, we present the underlying problem and our approach. We detail the implementation of the approach in Section 5 and evaluate it in Section 6. Finally, Section 7 concludes this paper and gives an outlook on future work.

2 RELATED WORK

Related work mostly deals with either the measurement of IDS performance or the inference of essential factors on the performance and optimized signatures.

Sen, in her report "Performance Characterization and Improvement of Snort as an IDS" [16], discussed the performance characteristics of Snort as an IDS and proposed ways to improve its performance by introducing new data structures. She performed a series of experiments to investigate the effect of the changes on Snort's performance. Her experiments showed that as network packet size increases bandwidth increases, whereas as the number of signatures supported on Snort increases, throughput decreases. The correlation between the number of packets and packet size on the bandwidth of Snort was an important aspect which can be taken from this paper to evaluate a system for Snort's performance.

Schaelicke et al. also investigated the performance characteristics of Network Intrusion Detection Systems (NIDS) in their work in [14]. As with [16], they generated packets of varying sizes and tested them against a varying number of rules. In their experiments, they also differentiated between the header and body signatures. Their tests showed that with larger packets, more signature rules could be present before observing a significant packet loss by Snort. Moreover, it was seen that microprocessor performance was not the only criterion for Snort's performance. Also, the header signatures exhibited a high processing load, limiting the packets per second.

Meng et al., in their report [9], are working towards improving NIDS by developing an Enhanced Filter Mechanism mitigating issues such as network packet overload, expensive signature matching, and high false alarms experienced in large-scale networks. The proposed solution consists of three major components, namely (1) a context-aware blacklist-based packet filter, (2) an exclusive signature matching component, and (3) a KNN-based false alarm filter. This paper's "Context-Aware Blacklisting" worked on blacklisting IP addresses with the help of a look-up table, and the "Exclusive signature matching component" that quickly identified a mismatch increased the signature matching process, thereby increasing NIDS performance.

Alhomouda et al. [2] compared the Suricata and the Snort NIDS. This paper focused on determining the packet loss at different network speeds for the two NIDS. Various trials evaluated the performance of applications hosted on different operating systems. Based on the results, the authors concluded that the choice of NIDS and the OS used should be dependent on the type of traffic.

Day and Burns [6] also examined Suricata and Snort in an overloaded condition. In contrast to [2], this paper analyzed accuracy, dropped packet rate, system utilization, and offline speed. This publication identified the metrics that were responsible for the characterization of the performance under different situations.

The work of Tjhai et al. confirmed the high false negative rate of Snort [18]. Under the division into "true" and "false alarm rate," they analyzed different frequently used signature sets to their false positive rate. The paper concluded that the high false positive rate was one of the future challenges for the development of NIDS.

The work of Chahal and Nagpal in [4] continued with the problem of false positive subordinate rule sets and developed a concept of the generalization of signatures. Similar signatures were combined to reduce the number of signatures to reduce false alerts. Less and more primitive signatures reduced the false positives.

Finally, Alomari and Menascé in [3] use an autonomic computing based approach to balance the performance and security's QoS requirements. An autonomic controller was presented that identifies an optimal security policy which improves the security and QoS of the system on the basis of analytical models that estimate the performance impact for a given security policy, thereby managing the performance and security tradeoff.

While all of these works dealt with either improving or evaluating the performance of IDS, none of the works applied SDN to reduce the load, increase the throughput, or reduce the false positive rate of existing SDN solutions.

3 TECHNICAL BACKGROUND

This chapter provides background information about the used technologies and introduces IDS, their categorization and their signatures. Our proposed solutions all rely on SDN, and we will later also evaluate the influence of NFV.

3.1 Intrusion Detection Systems

Intrusion Detection and Prevention Systems (IDPS) combine IDS and Intrusion Prevention Systems (IPS). IDS can be used to detect attacks [13], and many IDS provide additional defense mechanisms. IPS are capable of actively defending against incoming attacks, and many IPS are deployable in a detection-only mode.

In this work, we focused on network-based, misused-based, non-distributed, and real-time IDS.

Misuse-based approaches primary target singular attacks that are usually carried out in a single step [19], exploiting a selected vulnerability. Here, an IDS uses signatures containing features of an attack for its detection. **Real-time** or event-based IDS intercept an activity before it reaches the target system inspecting it synchronously to the traffic flow. **Non-distributed** IDS are deployable at a singular (central) position inside the system.

3.2 Software-Defined Networking

Software-Defined Networking (SDN) takes on the challenges posed by the increasing number of participants in networks and the associated exponential increase in costs due to the directly correlated growth in resource demands. The objective during development was to achieve greater scalability, flexibility, automation, and independence from hardware manufacturers to reduce acquisition and operating costs.

Five principles are fundamental to SDN: The **separation of control and data planes** divides the switching process into the control plane, using routing algorithms to decide on packet forwarding and the data plane technically handling the packet. SDN allows influencing the forwarding process from the outside via a software interface to communicate with the switch changing its behavior at runtime without having to replace the hardware components. The **central control instance**, also called controller, enables the configuration and administration of the network. Through **programmability**, the behavior of a switch can be changed using software, enabling the installation from algorithms or other applications from different manufacturers independent of the hardware producer. Additionally, **protocol independence** allows running different network protocols. **Open interfaces** are a prerequisite for vendor independence.

SDN brings together many areas that are handled separately in traditional networks via various application programming interfaces (API). There are four essential APIs that can be implemented in many ways [7, 11]. The **Southbound API** connects the control and data layer. The **Westbound API** is used to communicate different control layers of different domains. The **Northbound API** exchanges information between the application and control layers. The **Eastbound API** provides a contact surface for non-SDN components, such as Multi-Protocol Label Switching (MPLS) or other routing algorithms that function across domains.

OpenFlow (OF) is an open protocol for the Southbound API in SDNs. Due to its continuous further development and extensive hardware support, it has become the quasi-standard for SDNs. OF can be used to configure and evaluate the statistics of a network device, usually a switch.

3.3 Network Function Virtualization

Network Function Virtualization (NFV) is a new paradigm for networks. Typically deployed on proprietary specialized hardware in the past, these functions are replaceable by software solutions running on commodity hardware [5, 12]. The implementation of a function is usually referenced as Virtualized Network Function (VNF) as it is commonly deployed inside a Virtual Machine (VM) to allow for higher flexibility and scalability.

Many NFV solutions are usually implemented in conjunction with specialized operating systems or drivers to minimize bottlenecks. Mapping the network functions in software separates the data and control layers. Although both NFV and SDN share this separation, both paradigms still can be distinguished and implemented independently of each other.

4 APPROACH

In this section, we describe our approach to increasing IDS performance, while still considering security aspects. We first describe why IDS performance often can become a problem. Next, we introduce our two dynamic algorithms *Adaptive Blacklisting* and *Adaptive Whitelisting*. Finally, we present a simple static approach to the problem as a baseline for comparison to our dynamic algorithms.

4.1 Problem Statement

The current implementations of network-based IDS make use of a technique called DPI. They inspect every network packet in detail, thereby making them compute-intensive and a possible bottleneck in the network infrastructure. Also, studies have shown that under overloaded conditions, IDS can experience packet loss and network delays, resulting in reduced network bandwidth. Additionally, overloaded conditions can result in a high number of false-positive alarms, making them ineffective in the network. While it is possible to counter these issues by adding multiple IDS instances and load balancers, this approach is (1) very expensive, and (2) increases the attack surface (e.g., in the load balancer). Furthermore, another attack trend is that most of the attacks from an attacking host are launched as soon as possible so that the attacker stays in contact with the victim machine for the least amount of time to reduce the chances of detection. This characteristic creates an opportunity to design adaptive algorithms that make use of such findings to dynamically adjust the routing of packets either to the IDS or directly to the internal network.

4.2 Initial Situation

In the initial situation, the network consists of three segments as shown in Figure 1. First, an external network is the source for potentially malicious network packets. Next, the security portion of

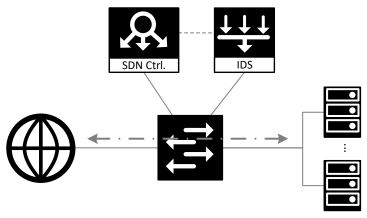


Figure 1: Traditional Switching: Direct routing from source to sink. IDS inline mode possible with single SDN Flow.

our network consists of an SDN-enabled network (represented by a switch), an SDN controller, and the IDS. Finally, the third segment is the protected internal network. In the default configuration, all traffic is routed directly between the external network and the internal network. Hence, the default configuration provides no protection. Resembling the inline deployment typical for IDS requires a single additional flow. This flow forwards all packets from the external network to the IDS. Then, it forwards the benign packets received back from the IDS to the internal network.

4.3 Adaptive Blacklisting

Adaptive Blacklisting distinguishes between blacklisted and non-blacklisted traffic. The principle behind Adaptive Blacklisting is that initially only those packets from applications, services, and protocols are routed via the IDS for which the IDS has signatures configured. Therefore, this approach puts traffic types with signatures on the blacklist. Traffic for other services is not routed via the IDS and is instead forwarded directly to its destination. This distinction eliminates the traffic load on the IDS for which it does not have any signatures.

In contrast to the existing static blacklisting approaches (e.g., the Selective Filtering described in Section 4.5), in Adaptive Blacklisting, connections are removed from the blacklist once they have not triggered an alarm for a certain amount of time. When a new connection, supported by the IDS arrives, the system requests instruction from the controller as seen in flow (1) in Figure 2a. After confirming that the requested connection is indeed a new connection, the controller creates two flows with different durations. The first flow forwards the network traffic to the IDS. The figure depicts this flow as flow (2). This flow has a higher priority but a shorter lifetime (X). Many attacks occur within the first few packets after establishing a new connection, so that they give an administrator the least amount of time to detect them. Therefore, once the lifetime of flow (2) times out, a second flow (3), created at the same time, forwards the traffic directly to the host destination by bypassing the IDS. This flow has a lower priority but a higher lifetime. If attacks are detected in flow (2) before the lifetime (X) times out, this flow is made permanent, and all the traffic from this host will pass through the IDS without compromising the system. Additionally, we can configure a timeout for the lifetime of bypassing the IDS (flow (3)) to be either permanent or temporary.

4.4 Adaptive Whitelisting

The fundamental concept behind Adaptive Whitelisting is that it does not necessarily require any knowledge about the configured signatures in the IDS routing all the network traffic except for optional explicitly whitelisted traffic types via the IDS.

For every connection, the receiving component initially queries the SDN controller which creates a flow via the IDS. If after a preset number α of packets the IDS has raised less than β alerts, the traffic becomes whitelisted, resulting in an additional network rule. The subsequent traffic of the tested connection is no longer routed via the IDS and instead is routed directly to its destination. A time limit Z is configurable after which the whitelisted traffic needs to undergo inspection again. The number of packets routed through the IDS in order to work effectively requires empirical studies. The system requests for instructions from the controller upon arrival of a new connection. Figure 2b depicts that action as flow (1). Like Adaptive Blacklisting, once the controller confirms that the requested connection is indeed a new connection, it sets up a single flow with the highest priority, passing all the network traffic through the IDS. The SDN controller now communicates with the IDS to record the packet characteristics detected for the newly created connection. Flow (3) in the figure represents this communication between the SDN controller and IDS.

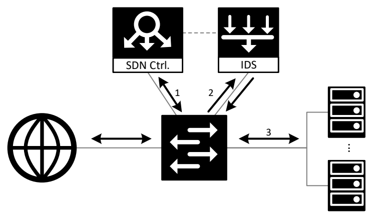
If, after α packets passed via the IDS, the SDN controller records that β or more packets have triggered alerts (in our case $\beta = 1$, i.e., if even one alert occurs in the sample space of α packets), it routes packets from that connection permanently via the IDS. Otherwise, if less than β alerts occur, the subsequent network traffic from this connection is routed directly to the host destination by bypassing the IDS. Like Adaptive Blacklisting, a predefined lifetime is present until the packets for a connection are allowed to bypass the IDS. On expiry of the lifetime, the network traffic for that connection will be passed through the IDS again to prove the connection to be deemed as benign.

4.5 Selective Filtering

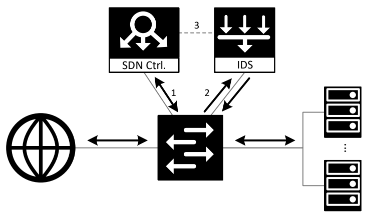
Selective Filtering is not an algorithm but rather a simple SDN-based static solution that helps to baseline our dynamic approaches. Such static flows, once established on the switch, do not change during operation. The concept of Selective Filtering requires only a few flows reducing the initial overhead of the SDN-based traffic analysis. In most production deployments, for optimal performance, most IDS are configured with signatures for a limited set of applications, protocols, and services. Selective filtering attempts only to statically route traffic via the IDS for which protocol or service signatures are available. This distinction requires knowledge about which application workload is running on which host and protected by which IDS. For each host server and application, Selective Filtering adds a flow entry in the switch which redirects all incoming traffic to this combination via the IDS. The remaining network traffic is forwarded directly to the destination. This approach is depicted in Figure 2c. One of the significant advantages of the Selective Filtering approach is its simplified deployment without compromising on security, as potentially malicious traffic passes through the IDS.

5 IMPLEMENTATION

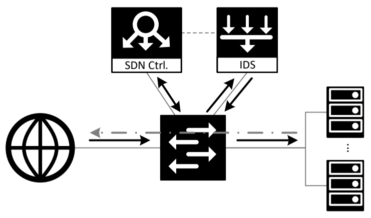
We implemented the algorithms presented in Section 4 to allow their evaluation. Additionally, we realized a load generator. This



a) Connection establishment via adaptive Blacklisting. The switch queries new connections for observed traffic types with the controller (1). The controller creates a new flow forwarding traffic to the IDS (2). If no alert is triggered, after some time the network directly forwards traffic to the service host (3).



b) Procedure for handling a connection in Adaptive Whitelisting. The switch queries the controller for non-whitelisted traffic (1). The controller creates a flow diverting the traffic to the IDS (2) and then queries the IDS for the observed attacks (3). If a threshold for recorderd attacks is not exceeded after a fixed time, the diversion is terminated.



c) Flows at selective filtering for diversion via the IDS. This configuration directly forwards traffic that has no signatures configured at the IDS to the service host. Traffic with configured signature has to pass through the IDS.

Figure 2: Forwarding Approaches

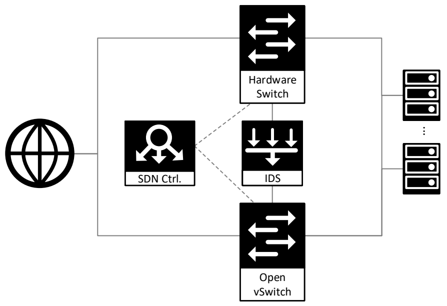


Figure 3: Testbed used for evaluation comprising benign and malicious traffic generation (simulating an external network), an IDS, SDN-enabled hardware and software switches able to send traffic either to the IDS or service host, an SDN controller, and service hosts.

section gives a short overview of the employed technologies. As SDN controller, we use Ryu[1]. Ryu is lightweight, supports basic switching and REST per default, and can be extended using simple Python scripts. We realized every algorithm as a single Ryu module. The choice for the IDS was Snort 2.9.9.0. Detected attacks are provided by Snort using its internal database. Our experiment controller written in Java controls the service host as well as the client(s), generates the workloads, executes the experiments, and records its results.

6 EVALUATION

In this section, we evaluate our approach. First, we describe the used testbed. Next, we introduce the used metrics, configuration scenarios for the testbed, and workloads. Last, we present and assess the measured results.

6.1 Testbed

The testbed we used to evaluate the presented approach comprises multiple servers and one Open Flow-enabled hardware switch as depicted in Figure 3. The servers take the roles of simulated clients, load driver, target server, software switch, IDS, and SDN controller. We used HPE ProLiant DL360 Gen9 servers with an octa-core Intel Xeon CPU with enabled hyperthreading and 32GB main memory. The choice for operating system fell on a 64-bit Linux with kernel version 4.4.0-72 for x86-64 architectures. The selected switch is an HPE 5130-24G-4SFP+ from the Aruba series. It supports Open Flow 1.3 as an SDN protocol, and its hardware table can contain up to 384 entries. For the SDN controller, we used the python-based Ryu controller. All network connections supported a maximal bandwidth of 1 Gbit/s. An experiment controller was connected to all devices via a separate experiment network. This experiment controller configured the servers and switches, starts and stops the measurements, monitors the experiment, and collects the metrics. The target service runs an Apache web server application.

[1]https://osrg.github.io/ryu/

6.2 Metrics and Their Acquisition

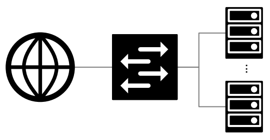
To evaluate the quality of our approach, we need to measure multiple metrics. These metrics comprise the throughput, the response time, and the accuracy of the attack detection.

Network Throughput: A significant metric to assess the performance of web servers and, therefore, also their protection system, is the achieved throughput. This value measures the amount of traffic processed by the system.

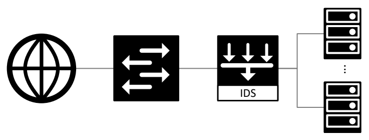
We are making use of the Simple Network Management Protocol (SNMP) to measure the throughput. This protocol is available on many switches and Operating Systems. It allows access to many settings and counters (so-called OIDs) of a system. These OIDs include the state and capabilities of a network interface, the CPU load, and the memory usage. The throughput considered in our paper is the number of incoming and outgoing bytes to the interfaces and ports involved in an experiment at the switch.

Network Delay: In addition to throughput, the delay is another essential metric in computer networks. The use of additional components that a network packet needs to pass through in the network, such as the IDS, leads to additional packet delays. A delay is the amount of time a packet needs from the source to the destination. For this paper, we use the time taken to establish a TCP Handshake.

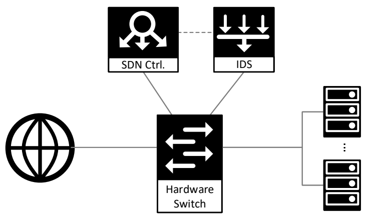
Since many applications use the TCP protocol for the use of IDS, the case is particularly interesting in which the resulting delay of an



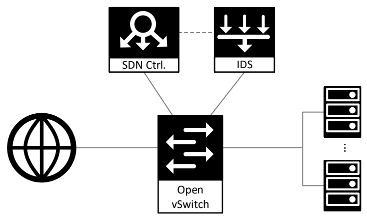
a) Scenario 1a (Unprotected Baseline): Direct connection between source and sink. Baseline for maximum throughput and no security.



b) Scenario 1b (Protected Baseline): IDS between source and target. Baseline for minimum throughput and maximized security.



c) Scenario 2: SDN-enabled network using a hardware switch.



d) Scenario 3: SDN-enabled network using a software switch.

Figure 4: Scenarios used for the Evaluation

entire TCP handshake becomes visible because it is the prerequisite for a TCP connection. The observation of the time difference of the TCP handshake also has technical reasons. The time difference can be determined merely through two time-stamps, before and after the call.

Attack Detection Rate: The attack detection is the number of attacks detected by Snort during the experiment. To determine the number, we used the barnyard2-managed MySQL database on the IDS as the basis to count the number of attacks. Barnyard2 inserts the attacks detected by Snort as an entry in the database. The difference between the number of database entries at the beginning and end of the experiment repetition is the number of attacks detected. This approach can be problematic if some Snort detections enter into the database after the experiment due to excessive delay. However, post-analysis of the database subsequently can reduce such effects. A problem with the automated analysis of attack detection is the false-positive and false-negative analysis. Although barnyard2 writes all the packets that Snort has sent to an alarm, the entries can contain some corrupted data. This limitation makes it difficult to search the database for patterns. For better detection of attacks, generated HTTP attacks contain the plain text 'attack.'

6.3 Scenarios and Workload

Scenarios 1a and 1b (Baselines): The goal of the reference scenarios is to baseline the setup on the two criteria of our interest, namely performance and security.

Scenario 1a consists of only a switch as a node between the source and the destination as can be seen from the Figure 4a. This scenario forwards traffic directly from the external network to the service hosts and back. Hence, this scenario represents the maximum data throughput with the lowest delay. The only limit to the network capacity between the source and the destination is the maximum bandwidth of the switch.

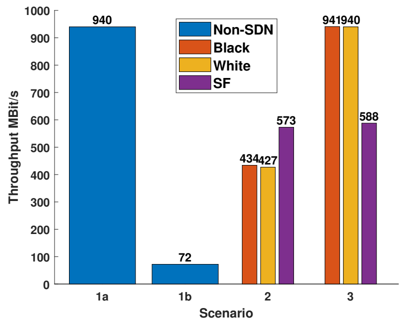
Scenario 1b consists of an inline IDS between the source and the sink as can be seen from Figure 4b. Therefore, all the network traffic is routed and examined by the IDS. This scenario helps us generate a reference for the performance of the network with the highest level of security by using an IDS. Here, some of the primary influencing factors to the bandwidth of the network are the speed of the Ethernet interfaces of the host system as well as the maximum supported throughput of the IDS that can be achieved based on scaling the IDS's system resources such as CPU and RAM. Further possible limitations include the I/O performance of the overall system and the operating system used.

Scenario 2 (Hardware Switching): This scenario employs an SDN-capable hardware switch in conjunction with an SDN controller as depicted in Figure 4c. The SDN controller allows us to manipulate the network's flow tables at runtime as for the Adaptive Blacklisting and Whitelisting algorithms described previously. The controller also has a feedback from the IDS's interface to obtain information about the detected attacks. The connection from the SDN Controller to the IDS is on a separate network to avoid interferences.

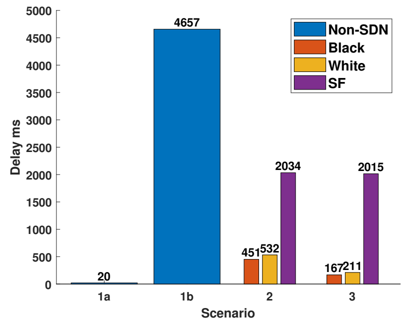
This scenario is significant in various ways. Firstly, the type, size, and performance of flow tables vary significantly with different switch models. An evaluation of how the nature of flow tables affects real applications is therefore particularly interesting. An OpenFlow compatible switch has two types of flow tables: software and hardware. While a dedicated processor processes the entries of the hardware table, the CPU takes over the processing for a software table reducing the performance. Even between the hardware tables, there are performance differences in the priority of flows within a table. If multiple tables are present, this fact also adds inter-table prioritization.

Secondly, apart from a pure QoS perspective, there are three other performance criteria for flow tables during the runtime: Adding, modifying, and deleting existing flows in a table. When managing many flows (more than 100), adding new flows can take a longer time than when there are only a few flows (less than 10). Additional delays can result in further problems, such as the additional triggering of a packet-in event for buffered packets.

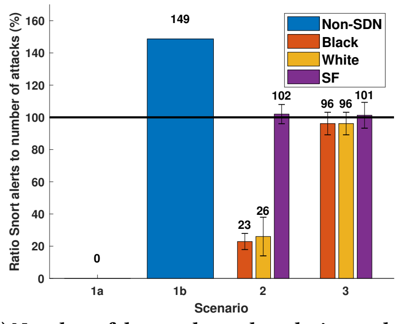
Scenario 3 (Software Switching): In the last experimental setup a software switch, Open vSwitch, is used instead of a hardware switch that was used in Scenario 2 as can be seen from Figure 4d. In contrast to a hardware switch, a software switch has an entirely



a) Throughput in MBit/s.



b) Latency in milliseconds.



c) Number of detected attacks relative to the expected number.

Figure 5: Performance and Security Results

different behavior. The flow capacities and the performance of the device are no longer dependent on the hardware specifications of the device. Moreover, the separation between the software and hardware tables vanishes and the complete table has consistent performance. Additionally, the size of the table is no longer dependent on the hardware. Although a more powerful host system leads to faster performance, the ability to use commodity hardware guarantees flexibility. We chose this scenario to assess the performance of this switch VNF in comparison to the hardware switch.

Workload: In our workload, we evaluate the performance and security properties of our applications. The workload puts the system under a constant load and focusses on evaluating the feasibility of our algorithms. To achieve this constant load, we query the server with HTTP requests. Exactly 170 requests are open at any time. We have chosen this number to ensure that this load would not exceed the hardware flow table of the employed switch specified at 384 entries. Once the client receives the completion of a request, it starts a new one. Each request consists of an HTTP POST-Request for a two Mebibyte file on an Apache web server. This size orients itself at the size of an average website [17]. Additionally, to this benign requests, our load generator starts five attacks every ten seconds. Attacks are packtes that match a configured signature. The IDS has signatures configured for these attacks. Every execution takes five minutes with 30 repetetions.

6.4 Performance and Security Results

Result Description: From two baseline scenarios, it becomes evident that Snort in inline mode results in a significant reduction in throughput. Figure 5a shows that while Scenario 1a achieves the theoretical maximum of 940 MBit/s [12], the routing via the IDS results in a drop to 72 MBit/s or by 92%. Scenario 2 shows that all bypassing algorithms increase the throughput in comparison to the baseline scenario with inline IDS. While Adaptive Blacklisting and Whitelisting reach approximately the same results with 434 MBit/s (Blacklisting) and 427 MBit/s (Whitelisting), the Selective Filtering reaches 573 MBit/s. In Scenario 3 this balance changes. Both dynamic approaches reach the theoretical maximum throughput while Selective Filtering gains only a minimal improvement in throughput.

Figure 5b shows the effect of the various scenarios on the network delay. The addition of the IDS increases the delay from 20 ms in Scenario 1a to 4 657 ms in Scenario 1b. In Scenario 2, we see a significant reduction in the delay for all bypassing approaches. Unlike the throughput, dynamic approaches achieve a smaller delay at 451 ms (Blacklisting) and 532 ms (Whitelisting) than the Selective Filtering at 2 034 ms. Selective Filtering again only marginally improves its performance upon adding the software switch in Scenario 3. At the same time, the dynamic approaches further improve more than halving their delay to 167 ms respectively, to 211 ms.

Figure 5c shows the ratio between the number of detected attacks and the number of executed attacks. Apparently, without an IDS, Scenario 1a detected no attacks. Scenario 1b adds the IDS and already shows around 149% detection rate. So, more attacks are detected than are executed. Scenario 3 shows a significant drop in detection rate for the dynamic approaches. Adaptive Blacklisting achieves only 23% detection rate, and Whitelisting is only slightly better at 26%. The selective Filtering achieves a detection rate of 99% which is within the margin of error to the ideal 100% rate. Using the software switch in Scenario 5 increases the ratio to 96% for both dynamic approaches and 101% for Selective Filtering. All three approaches are within the margin of error of 100%. When examining the attacks in Snort's database, they all contain the sequences that should trigger the signatures. Thus, the attacks appear to be correctly detected.

Discussion: The comparison between Scenario 1a and 1b confirms the motivation for this paper. Adding an inline IDS reduces the throughput and increases the latency drastically. Furthermore, an inline IDS under high load triggers more alerts than actual attacks occurred.

As expected, bypassing the IDS increases the performance. However, the dynamic approaches behave differently than Selective Filtering. While in Scenario 2 they increase the performance relative to the inline IDS, they decrease the rate of detected attacks to about a fourth of the actual attacks which is an unacceptable security characteristic.

When replacing the hardware switch with a software switch in Scenario 3, the performance of the adaptive approaches increases even further to the theoretical throughput maximum and a latency acceptable for a web server. The attack detection ratio increases as well, and the detection of all executed attacks is within the margin of error. On further investigation, it appears that in some conditions the hardware switch starts using the software table. This behavior is unexpected since we chose the number of active connections below the hardware table capacity. The switch becomes significantly slower and frequently does not react to requests from the controller when using the software table. When the rerouting flows are not installed or incorrectly installed, traffic could either be sent permanently via the IDS (reduced performance) or directly to the service host (no attack detection). The results for the Selective Filtering show that the bypassing with simple rules can yield an increased performance compared to inline mode.

In summary, the bypassing algorithms show promising results. They can improve performance up to an extent where the introduction of the IDS has no impact on performance. Attack detection is within the margin of error to 100% under typical load for both algorithms. The algorithms profit from the use of the software switch.

Limitations: The main limitation of our measurement approach is that it counts only the number of detected attacks. Therefore, it is not yet possible to assert which attacks are detected and account for false-positives and false negatives. The approach can be extended to collect this type of information as well.

Also, our framework at the moment allows no direct tracking of what happens at the hardware switch when it becomes unresponsive in Scenario 2. Any assertion which flows get redirected and those that do not will require this functionality.

7 CONCLUSION

We introduced the problem that the performance of security devices is crucial to the success of cloud systems. After looking at related work and the technical background, we presented three algorithms - two dynamic ones, (1) Adaptive Blacklisting and (2) Adaptive Whitelisting, and a static one, (3) Selective Filtering, to improve the performance of network intrusion detection systems by selectively bypassing them. We evaluated these approaches using four scenarios realized in a testbed environment. The results show that our approch improves the performance using bypassing while upholding a high level of detection accuracy. The dynamic algorithms have severe problems when using a hardware switch. Performance, as well as detection accuracy, drops. The static approach behaves similarly for software and hardware switches and gives a decent increase in performance. Thus, this work confirms the potential of bypassing algorithms to improve intrusion detection performance.

In future work, we will extend our testbed environment to support 1:1 accounting for the attacks carried out. This addition will allow the automatic detection of false positives and false negatives, as well as duplicate detection. Furthermore, we plan to track all network traffic. With this additional information, we will analyze the effect of the hardware switch in detail. Additional data also allows improving our algorithms further to ensure better detection ratios. Also we will investigate further IDS including IDS capable of parallel processing as well as various attack/signature combinations.

8 ACKNOWLEDGMENTS

This work was funded by the German Research Foundation (DFG) under grant No. (KO 3445/16-1).

REFERENCES

[1] 2018. Global Hybrid Cloud Market 2014-2021 | Statistic. (Oct. 2018). https://www.statista.com/statistics/609581/worldwide-hybrid-cloud-market-size [Online; accessed 30. Oct. 2018].

[2] Adeeb Alhomoud, Rashid Munir, Jules Pagna Disso, Irfan Awan, and A. Al-Dhelaan. 2011. Performance Evaluation Study of Intrusion Detection Systems. *Procedia Computer Science* 5 (2011), 173–180. https://doi.org/10.1016/j.procs.2011.07.024

[3] Firas B. Alomari and Daniel A. Menascé. 2013. Self-protecting and Self-optimizing Database Systems. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference on - CAC '13*. ACM Press. https://doi.org/10.1145/2494621.2494631

[4] Ayushi Chahal and Ritu Nagpal. 2016. Performance of Snort on Darpa Dataset and Different False Alert Reduction Techniques. In *3rd International Conference on Electrical, Electronics, Engineering Trends, Communication, Optimization and Sciences (EEECOS)*. https://pdfs.semanticscholar.org/9634/2f678949bcae35eabda3cfafeb0d0abe1d32.pdf

[5] Margaret Chiosi, Don Clarke, Peter Willis, Andy Reid, James Feger, Michael Bugenhagen, Waqar Khan, Michael Fargano, Dr. Chunfeng Cui, Dr. Hui Deng, Javier Benitez, Uwe Micheel, Herbert Damker, Kenichi Ogaki, Tetsuro Matsuzaki, Masaki Fukui, Katsuhiro Shimano, Dominique Delisle, Quentin Loudier, Christos Kolias, Ivano Guardini, Elena Demaria, Roberto Minerva, Antonio Manzalini, Diego Lopez, Francisco Javier Ramon Salguero, Frank Ruhl, and Prodip Sen. 2012. Network Functions Virtualization (NFV), An Introduction, Benefits, Enablers, Challenges & Call for Action. SDN and OpenFlow World Congress, Darmstadt, Germany. (2012). http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[6] David Day and Benjamin Burns. 2011. A Performance Analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines. https://www.thinkmind.org/download.php?articleid=icds_2011_7_40_90007

[7] Michael Jarschel, Thomas Zinner, Tobias Hossfeld, Phuoc Tran-Gia, and Wolfgang Kellerer. 2014. Interfaces, Attributes, and Use Cases: A Compass for SDN. *IEEE Communications Magazine* 52, 6 (June 2014), 210–217. https://doi.org/10.1109/mcom.2014.6829966

[8] Joseph McKendrick. 2015. 2015 IOUG Data Integration For Cloud Survey. (May 2015). http://www.oracle.com/us/products/middleware/data-integration/ioug-di-for-cloud-survey-2596248.pdf Produced by Unisphere Research, a Division of Information Today, Inc.

[9] Weizhi Meng, Wenjuan Li, and Lam-For Kwok. 2014. Efm: Enhancing the Performance of Signature-based Network Intrusion Detection Systems Using Enhanced Filter Mechanism. *computers & security* 43 (2014), 189–204. https://doi.org/10.1016/j.cose.2014.02.006

[10] Aleksandar Milenkoski, Bernd Jaeger, Kapil Raina, Mason Harris, Saif Chaudhry, Sivadon Chasiri, Veronica David, and Wenmao Liu. 2016. Security Position Paper: Network Function Virtualization. (March 2016). https://cloudsecurityalliance.org/download/security-position-paper-network-function-virtualization/ Published by Cloud Security Alliance (CSA) - Virtualization Working Group.

[11] Open Networking Foundation. 2016. Impact of SDN and NFV on OSS/BSS - ONF Solution Brief. (1 March 2016). https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-OSS-BSS.pdf

[12] Piotr Rygielski. 2017. *Flexible Modeling of Data Center Networks for Capacity Management.* Ph.D. Dissertation. University of Würzburg, Germany. https://opus.bibliothek.uni-wuerzburg.de/frontdoor/index/index/docId/14623

[13] Karen Scarfone and Peter Mell. 2007. *Guide to Intrusion Detection and Prevention Systems (IDPS).* Technical Report. https://doi.org/10.6028/nist.sp.800-94 NIST Special Publication 900-94.

[14] Lambert Schaelicke, Thomas Slabach, Branden Moore, and Curt Freeland. 2003. Characterizing the Performance of Network Intrusion Detection Sensors. In *International Workshop on Recent Advances in Intrusion Detection.* Springer, 155–172. https://doi.org/10.1007/978-3-540-45248-5_9

[15] Holger Schulze. 2015. Cloud Security Spotlight Report. (2015). https://goo.gl/rMGh3x Presented by Information Security, LinkedIn Group Partner.

[16] Soumya Sen. 2006. Performance Characterization & Improvement of Snort As an IDS. *Bell Labs Report* (2006). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.720.2007&rep=rep1&type=pdf

[17] Wired Staff. 2018. The Average Webpage Is Now the Size of the Original Doom. *WIRED* (March 2018). https://www.wired.com/2016/04/average-webpage-now-size-original-doom

[18] Gina C Tjhai, Maria Papadaki, SM Furnell, and Nathan L Clarke. 2008. Investigating the Problem of Ids False Alarms: An Experimental Study Using Snort. In *IFIP International Information Security Conference.* Springer, 253–267. https://link.springer.com/content/pdf/10.1007%2F978-0-387-09699-5_17.pdf

[19] Giovanni Vigna, William Robertson, and Davide Balzarotti. 2004. Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *Proceedings of the 11th ACM conference on Computer and communications security - CCS '04.* ACM, ACM Press, 21–30. https://doi.org/10.1145/1030083.1030088