SDN-Based Kernel Modular Countermeasure for Intrusion Detection

Tommy Chin[1], Kaiqi Xiong[2](✉), and Mohamed Rahouti[2]

[1] Rochester Institute of Technology, Rochester, USA
tommy.chin@ieee.org
[2] University of South Florida, Tampa, USA
xiongk@usf.edu, mrahouti@mail.usf.edu

Abstract. Software-Defined Networking (SDN) is a core technology. However, Denial of Service (DoS) has been proved a serious attack in SDN environments. A variety of Intrusion Detection and Prevention Systems (IDPS) have been proposed for the detection and mitigation of DoS threats, but they often present significant performance overhead and long mitigation time so as to be impractical. To address these issues, we propose KernelDetect, a lightweight kernel-level intrusion detection and prevention framework. KernelDetect leverages modular string searching and filtering mechanisms with SDN techniques. By considering that the Aho-Corasick and Bloom filter are exact string matching and partial matching techniques respectively, we design KernelDetect to leverage the strengths of both algorithms with SDN. Moreover, we compare KernelDetect with traditional IDPS: SNORT and BRO, using a real-world testbed. Comprehensive experimental studies demonstrate that KernelDetect is an efficient mechanism and performs better than SNORT and BRO in threat detection and mitigation.

Keywords: Aho-Corasick · Bloom filters
Intrusion detection system · Security
Software Defined Networking (SDN)

1 Introduction

Software-Defined Networking (SDN) has played a key role in Science DMZ (demilitarized zone). SDN grants an open-source asset and a great tool for developers and researchers to design and discover new solutions to networking challenges such as end-to-end delay minimization, traffic management, and network attack detection. However, SDN itself is vulnerable to various adverse attacks. Hong et al. [34] identified threats including Denial of Service (DoS) in SDN and examined DoS attacks under the environment of eight different SDN controllers, but there remain grand challenges to detect and mitigate them.

This research considers an environment like Science DMZ where there is a need to high-speed network access to computation and storage for science research. As mentioned before, Hong et al. [34] have presented DoS threats and

X. Lin et al. (Eds.): SecureComm 2017, LNICST 238, pp. 270–290, 2018.
https://doi.org/10.1007/978-3-319-78813-5_14

proved that the exploitations are serious attacks in an SDN environment. Traditional network approaches to detect and mitigate DoS threats is through the use of Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), but they present serious concerns including system performance [19], network communication constraints [28], and detection validity [32]. Additionally, IDS detection methods present a critical flaw to identify new or unknown network attacks due to limiting threat signatures and comparison approaches. Recent studies have suggested a variety of threat mitigation and detection solutions including FloodGuard [19], SPHINX [28], and an entropy-based solution [32], but none of them, to the best of our knowledge, has studied a modular kernel-level IDPS approach within SDN environments.

In this paper, we propose KernelDetect, a lightweight modular-based filtering approach inspired by Amann and Sommer [21] and Mekky et al. [18], to detect and mitigate threats within an SDN environment. Specifically, KernelDetect is an independent application-plane network Test Access Point (TAP) approach using Switch Port Analyzer (SPAN) interfaces [20] on SDN switching devices. Moreover, by using a modular approach as a key component, KernelDetect can interchange the technique for string matching in addition to updating its signatures while providing threat mitigation capabilities within a kernel space. As we know, IDS signature methods are to compare a list of given strings or a set of rules with incoming network traffic signatures. In this paper, the proposed KernelDetect provides the ability to dynamically update the rule set in SDN environments in which we can optimize traffic inspection when detecting network threats.

To examine KernelDetect, we utilize Global Environment for Network Innovations (GENI) [27] to conduct our real-world experimental evaluation. Additionally, we comparatively examine KernelDetect to the popular IDS solutions: SNORT [29] and BRO [39] where KernelDetect leverages the Aho-Corasick [26] algorithm and Bloom filter [11] with SDN. To provide hybrid network communications, we utilize D-ITG [33] and iPerf [14] as traffic generation software for normal user data in the SDN experiments. To mix normal user traffic with malicious ones, we implement DoS attacks [32] in our threat detection and mitigation experiments. We further implement KernelDetect in an environment driven by Floodlight [3] using Representational State Transfer (REST) Application Program Interface (API) as our method of communication for KernelDetect to mitigate adverse threats and attacks.

KernelDetect resides on each switching device within an SDN environment and offers management controls using REST API calls. Such controls provide the ability to apply Access Control List (ACL) rules to SDN switches from a controller to mitigate an adverse threat. To be concise, KernelDetect listens to traffic on respective switching devices, and if a threat is detected, then mitigation occurs by informing the controller of the actions needed to thwart the attack. We further comparatively examine KernelDetect over traditional IDPS technologies –SNORT and BRO for the detection and mitigation of DoS attacks in a real-world testbed environment where we test various numbers of packets ranging from 100K to 500K and examine SYN flooding attack with different packet sizes and sampling times. In our extensive experiments, we measure the average load of system resources, inspection time, mitigation time, true positive, false positive, and false negative.

To summarize, we make the following main contributions in this research:

- DoS has been identified as a serious attack in an SDN environment [34]. We present KernelDetect, a lightweight kernel-level IDPS approach to thwarting DoS threats with the ability to interchange string matching detection mechanisms between the Aho-Corasick algorithm [26] and the Bloom filter algorithm.
- Existing IDPS tools such as SNORT and BRO utilize a culmination of user and kernel space due to the necessary user interaction needed to configure both solutions. Contrary to existing conventional studies, KernelDetect is a pure kernel-space solution. Furthermore, the default installations of SNORT and BRO provide many detection rules for their respective systems. The more number of rules we use, the more performance overhead is added. KernelDetect has a much less overhead compared to SNORT and BRO.
- We leverage the common architecture of Science DMZ with SDN technologies to develop KernelDetect. Thus, KernelDetect applies to Science DMZ, and it can enhance data-driven research in academia and national laboratories, and other related applications in industry and government agencies.
- As SNORT [29] and BRO [39] are traditional IDS solutions, we experimentally evaluate KernelDetect against the two well-known kernel-space and user-space detection tools in a real-world testbed, whereas many existing studies are evaluated either through a simulator such as Mininet [4] or in a lab environment whose results are often away from realistic.

The rest of this paper is organized as follows. Section 2 provides the background and challenges of our research problem. Section 3 discusses related work. While Sect. 4 presents threat models and attack vectors, Sect. 5 outlines the architectural design of the proposed solution. In Sect. 6, we give the experimental setup of KernelDetect evaluation with results. Lastly, Sect. 7 concludes our study and gives future work.

2 Research Background and Problem

In this section, we provide a brief background of kernel-space detection techniques and outline our research challenges.

2.1 Kernel-Space Detection Background

Kernel-space detection is a vital catalyst for intrusion detection systems due to its fundamental view of high-performance computing and minimal overhead. The use of deploying such a space/region has limited visibility as a traditional IDS utilizes user and kernel-spaces [28]. Moreover, system applications and services utilize both the regions of computing, but only using one region for such processes is not a common approach. Within SDN, there are numerous IDS solutions, but many utilize a culmination of kernel and user spaces to identify their respective adverse threats. Moreover, SDN switching software such as Open vSwitch (OVS) attaches itself to both user and kernel-spaces and it requires packet data from raw sockets on their respective operating systems to carry appropriate network traffic to the SDN switching service. Using a kernel-space provides capabilities for high-performance and a low overhead but presents a concern due to the instability of a kernel panic, resulting in the following challenges.

2.2 Research Challenges and Assumption

Common approaches to detect and mitigate adverse threats is through the use of an IDPS. One major issue of such a technique is through user-space utilization. Moreover, numerous IDS solutions rely on user-space interfaces to allow administrators to manage and maintain the various services that are implemented to identify and thwart malicious attacks. (1) *Kernel Panic:* The first challenge through the use of a kernel space is when a system is panic. Commonly, when a kernel module or a kernel-space application generates an erroneous issue such as a programming bug or a buffer overflow, a panic occurs such that the operating system is no longer function to provide service to the end user. When such an event occurs, a sequence of recovery mechanisms is executed such as memory dumping and a total system restart. We identify this challenge as a significant area to address as KernelDetect resides purely on a kernel-space. We identify this challenge as a significant area to address as the operation of KernelDetect resides purely in kernel-space and that if KernelDetect malfunctions or generates a programmatic error, a kernel panic would occur. (2) *Root Access and System Vulnerability:* Using kernel-space detection requires a significant level of system access to identify such malicious traffic. This level of access is known as root-access and proposes a serious challenge if the IDPS solution [15] were to be compromised or exploited. Moreover, to both inspect traffic and determine adverse behaviors, elevated access is required on such service to gain a control of raw sockets on an operating system. Using traditional IDS solutions such as BRO and SNORT, service accounts are created to secure the system from exploitation through techniques such as chroot and jailing. These concerns present the second challenge.

For the first challenge, the IDPS solution [16] should be robust from techniques such as a buffer overflow, resilient to obfuscated attacks, and exploitation schemes [24]. Using the operating system's raw socket feature provides the ability to handle and evaluate the large quantity of network traffic in an efficient manner. During a scenario of a DoS [42], excessive packet drops would occur as the system would be unable to handle the quantity properly. Additionally, the overhead and congestion presented from a DoS would create a significant delay as the inspection system would place each packet into a queue for evaluation. Overtime, this queue would significantly increase and may present a concern for a buffer overflow if mishandled incorrectly. The simple solution to prevent an overflow would be to drop packets aggressively to prevent resource exhaustion on the inspection system. One concern for this procedure will be if the network traffic has a level of urgency or priority regarding guarantee delivery [5,10], but this situation would heavily depend on the configuration and design of a network. A more serious concern for the use of kernel space detection is the configuration that the application requires root-level privilege on the IDPS system and presents a concern if the system becomes compromised.

The second challenge of the proposed kernel-space IDPS solutions requires an elevated user or root-access to gain accessibility to a variety of raw socket communication to collect and inspect network traffic [35]. Although such access is necessary for inspection purposes, it raises a potential concern for an emerging threat vector. Moreover, a compromised switching device draws a significant concern as network visibility becomes large such that a threat actor gains a larger attack surface to identify potential targets. One approach to attaining such access is through a vulnerability in the inspection step of our IDPS solution such that a malicious payload may be misinterpreted [1].

In this research, we assume that the implementation of KernelDetect is bugs free on a secure kernel where OVS is also secure.

3 Related Work

A common technique to identify adverse behaviors within network traffic is through the use of string matching techniques. Such identification has been examined using approaches such as Bloom filters [11] and Aho-Corasick [26]. There have been numerous studies to comparatively identify each string matching approach [11,26] for performance evaluation, but these studies lack in the identification of kernel-space detection. Furthermore, there have numerous developments of IDS solutions [6–8,16,19,28,31,36] to deter malicious traffic, but they heavily rely on user-space detection. Examination of IDS solutions in kernel-space detection has been evaluated through research work [17,23] but their approaches do not address traffic dynamics.

In this research, we introduce SDN to address this concern. As SDN has been widely used to improve network management, performance, and usability, FloodGuard [19], SPHINX [28], and FortNOX [31] employed SDN for attack detection and mitigation. Scott-Hayward et al. [35] summarized recent studies on the vulnerabilities of existing approaches in an SDN environment. FortNOX [31] addressed an SDN tunneling attack and solved the rule conflicts of an SDN flow table. Furthermore, Mahout [13] introduced a solution to improve the prevention mechanism for flooding attacks in an OpenFlow environment. SPHINX [28] attempted to detect attacks that contravene learning-based flow graphs and modules by designing a network flow graphs-based prototype. RAID [21] also introduced a control prototype to monitor the network systems passively and to target operational exploitation in a large-scale environment, but the effectiveness of this prototype was assessed only through OpenFlow backed connecting to three hardware switches. Moreover, Wang et al. [32] considered DoS attacks and gave an entropy-based solution to check detection validity. FRESCO [37] suggested a framework to simplify the scheme for the composition of security applications.

TopoGuard [34] considered the security of SDN controllers where TopoGuard attempts to capture attack poison in an SDN environment (i.e. the holistic visibility of a network environment and topology) based on security omission's fixation. Rosemary [38] adopted a practical approach to addressing the issue of control layer resilience through an extension of a NOS design. While their efforts have primarily focused on protecting the data plane of SDN from malicious applications, our proposed solution will have the ability to dynamically update the rule set in SDN environments and optimize traffic inspection when detecting network threats.

Furthermore, existing studies often suggested to combat one type of threats using SDN techniques, e.g., Wang et al. [32]. SDNScanner [40] and AVANT-GUARD [36] introduced solutions to detect and mitigate saturation attacks (data-to-control plane saturation) by altering flow management at a switch level, but their approaches are limited to TCP saturation attacks. Furthermore, they exposed only those flows that complete a TCP handshake based on a SYN proxy implementation.

Moreover, VeriFlow [25] detached a holistic network environment into subclasses that have exactly similar forwarding behaviors exploiting a multi-dimensional prefix tree so that all forwarding policies and determined policies would be checked in live time whenever a network update occurs. NetPlumber [30] proposes a real-time policy verification tool based on Header Space Analysis (HSA). NICE [16] introduces an approach to detecting network software bugs in OpenFlow applications based on symbolic execution and model checking. While FAST [9] identifies areas in conducting a forensic study on switching devices.

To the best of our knowledge, KernelDetect gives the first kernel-level solution instead of traditional user-space IDPS ones. It is a lightweight kernel-level detection mechanism. Contrary to the existing conventional work, we investigate IDPS on a kernel space that overcomes the implementation difficulty of a kernel space (e.g., SoftFlow [12]). As Snort and Bro are popular tools in this area, we choose them in our comparison study.

Likewise, most existing evaluation techniques deploying SDN for detection and mitigation, for example, TopoGuard [34] prototype evaluation is based on Mininet [4] - a simulator whose results may be practically far from real-world scenarios. Instead, KernelDetect is evaluated on GENI, a real-world testbed.

4 Threat Models and Attack Vectors

This research examines adverse users within an SDN environment where a series of normal traffic will communicate with normal users (or called clients). While SDN is widely used in traffic management, a variety of serious attacks such as DoS [32], LDS [34], and MITM [34] have been found in SDN. That is, the threat model includes the methodologies of launching DoS attacks using research work [32,34]. Although KernelDetect can be used for the detection and mitigation of other emerging network threats, we specifically consider DoS as our attack vector for this paper. To be concise, we will periodically implement our methods of DoS attacks on GENI as described in Sect. 6. Although a threat actor can launch any methods of attacks in a series or simultaneously, we will examine the effects of each threat *individually* for the performance evaluations and detection validity of KernelDetect. Trust needs to be identified in our SDN topology where we outline a variety of weaknesses in our infrastructural design to establish threat detection. To clarify, we assume that all SDN controllers and switching devices are safe from a threat actor, but leave end devices vulnerable to attacks. Mitigation is a critical factor to thwart an attack, and to prevent false positive events carefully; whitelisting will be required.

Whitelisting is a common approach to safeguarding mitigation faults such as disabling the WAN interface at an edge router and a network link to a known trusted computing device. In our threat detection approach, we do not implement any whitelisting for end devices attached to SDN switches as all users can be adverse at some point of time. Moreover, using KernelDetect, we implement detection on each suitable switching device for inspection purposes that will be further described in our experimental evaluation. Inter-switch links, commonly identified as a shared network link between two switching devices, contain a variety of network traffic intent from malicious to a normal user. Moreover, if these links were to be disabled through mitigation techniques, network operations would potentially fail. We inter-switch links to prevent mitigation faults from occurring. Although safeguarding inter-switch links provides reassurance from mitigation faults, a compromised end device has a greater potential to establish a significant threat to an SDN environment.

Lastly, we treat KernelDetect trustworthy even though adverse users can potentially obfuscate, exploit or overfill buffers specific to IDS solutions in addition to our string matching methods, Bloom filter, and the Aho-Corasick algorithm. We will identify an attack method in our experimental evaluation of Sect. 6. Following our evaluation, we have also investigated an IDS solution for other threats. However, we only present our study for DoS in this paper due to the page limit.

5 Design of KernelDetect

This section presents the architectural design of KernelDetect with discussions. We further discuss a threat signature structure for our proposed detection solution.

5.1 KernelDetect Placement and Architecture

The placement of KernelDetect is critical to detection and mitigation timings of an emerging threat. Before we present the architectural design of KernelDetect,

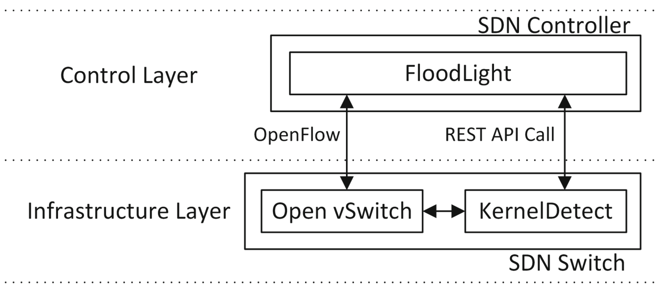

Fig. 1. The placement and functionality of KernelDetect for network traffic flow.

Fig. 1 shows the location and functionality of KernelDetect whose implementation is done in a configuration that operates in tandem with an SDN switching device.

Traffic duplication occurs within KernelDetect as both KernelDetect and OVS utilize raw socket communications in the back-end of the software system. The SDN Controller receives REST API calls from each switch when identifying a threat for mitigation. In this research, we use Floodlight as the controller software due to its REST API features. Figure 2 provides an architectural design of KernelDetect for both traffic inspection and signature matching with decision-making processes.

Let kds be a KernelDetect score, a an administrative-set incremental value for adverse traffic, b a decremental value for trustworthy traffic, and kdt a threshold value to determine whether such traffic should be placed in an inspection through either the Aho-Corasick algorithm or Bloom filter, called *Aho-Corasick inspection* or *Bloom filter inspection*, respectively. M simply denotes the matching scheme for KernelDetect.

When traffic enters an interface on a respective switch, the value is temporarily stored, and the information is forwarded to OVS and KernelDetect for their appropriate purposes of forwarding and inspecting traffic, respectively. During the initial state of KernelDetect, that is, when the service begins, an administrative configuration is examined to verify if a secure mode is enabled. We define the secure mode as a parameter such that if the placement of the switching device is in a critical data region, KernelDetect will enforce a detailed inspection using Aho-Corasick. If the placement does not have severe inspection approaches, then KernelDetect may use Bloom filter for detection. During the inspection process, we identify and examine to see whether the traffic has malicious intent through signature matching. If the intent is considered trustworthy, then we simply forward the traffic and decrease kds by a value of b, and add a when the intent is not trustworthy. Using a threshold condition of comparing kds to kdt, we examine whether future traffic should remain in Aho-Corasick or Bloom filter inspection. If the traffic has a malicious intent, we simply drop the packet from the raw socket and inform the SDN controller using REST API calls to block the adverse threat.

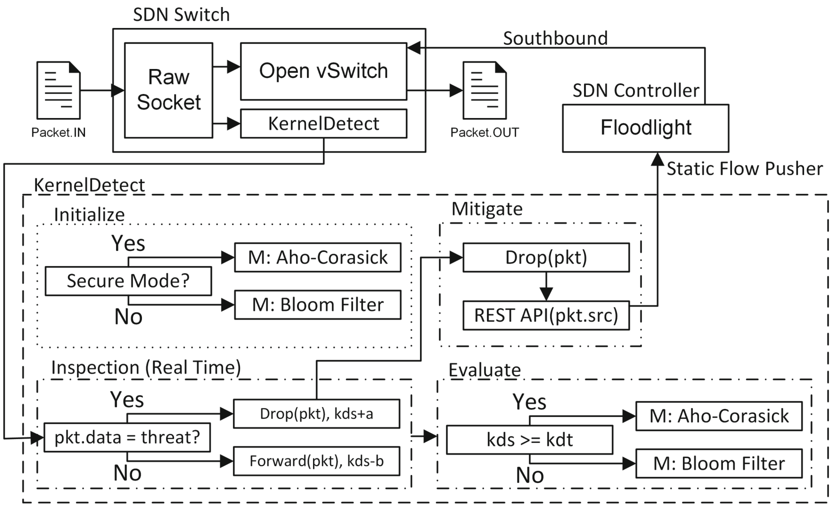

Fig. 2. The architectural design of KernelDetect consisting of four states: "Initialize," the beginning state of the SDN switch operations, "Inspection," a real-time inspection of traffic obtained from the raw socket of the operating system, "Mitigation," a critical step to thwart an attack and to prevent false positive events carefully, and "Evaluation," the examination of incoming traffic through 'Aho-Corasick' or 'Bloom filter' with a global view of the network.

5.2 Threat Signature Structure

Identifying adverse network traffic could be challenging as it depends on IDS signatures and threat identification markings. Particularly, two common approaches are considered to identify traffic threat through string-based matching, and traffic over time where an observation of a pattern of network packets occurs in a given period. As mentioned before, although KernelDetect applies to various attacks, this paper focuses on a DoS attack vector due to the page limit.

DoS: The identification of a DoS attack can be a challenge in an at-scale network. There are multiple methods to create a DoS attack from TCP SYN-flooding to other detailed approaches such as OSI Layer 7-based flooding. Like [32], we can identify a traffic pattern over an interval of time to determine if there is a DoS attack. That is, if the quantity of traffic exceeds a given threshold, KernelDetect considers that a DoS attack occurs, and it raises an alert. This threshold is a fixed value among all the approaches studied in our experiments later. The correlation with signature matching relates towards the frequency of alerts that is, KernelDetect raises an alert when a match occurs. The observation of a threat can originate from one or multiple sources where the attacker may spoof the source address of the DoS. Based on this given knowledge, KernelDetect accounts for such threats.

Signature-based matching may not be the appropriate tool to detect DoS attacks where the adversary can often insert arbitrary data into a packet payload.

This approach renders signature-based detection ineffective. In some cases, DoS attacks may not have any form of data for its payload, such as a low-profile TCP SYN flood attack. However, KernelDetect, considers matching the header information of a network packet rather than its packet payload, which increases the performance of threat detection. Below is the algorithm for KernelDetect where TH and THP are threshold values for time and packet intervals, respectively.

```
P = PACKET_IN
while P do
  TS = TIMESTAMP
  if P.TYPE == ICMP then
    Q{P.SRC_ADDR}++
    if P.SRC_ADDR NOT IN S then
      S{P.SRC_ADDR} = TS
    else
      if TS - S{P.SRC_ADDR} > TH then
        if Q{P.SRC_ADDR} > THP then
          REST API Call to SDN Controller
        else
          S{P.SRC_ADDR} = TS
        end if
      end if
    end if
  end if
end while
```

6 Experimental Evaluation

We have carried out the comprehensive evaluation of KernelDetect by choosing different experimental parameters such as the varying number of packets and threshold time. This section summarizes the evaluation of KernelDetect and presents a part of experimental results. For this purpose, we start with the topology design of our experiments using GENI.

6.1 Experimental Topology Design

To measure the effectiveness of KernelDetect, we utilize GENI [27] for experimental evaluation. GENI is a real-world heterogeneous virtual testbed with networking capabilities including SDN. To evaluate KernelDetect, we construct a topology with the following three constraints: (1) An adverse user attached to a single network link identifying major areas of mitigation. (2) A shared network link used by both a normal user and an attacker. (3) An edge network link that carries both normal and attack traffic. This edge link has limited SDN controller management. Figure 3 gives a visual view of the experimental topology that considers the previous research challenges where the locations of adverse users are

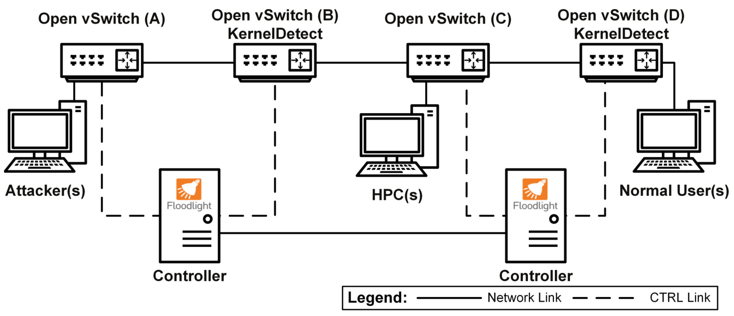

Fig. 3. GENI experimental topology for evaluation where KernelDetect is only implemented in switches B and D as depicted in the diagram. Moreover, CTRL links are the communication medium between each SDN switch and their respective controller. Lastly, our experimental evaluation interchanges KernelDetect-enabled switches with SNORT and BRO for our comprehensive study.

explicitly labeled. For presentation purpose in this paper, we give a relatively simple topology for our evaluation as shown in Fig. 3. However, KernelDetect is applicable to any complex network topology.

Although client nodes may have the potential to be compromised, we do not evaluate this scenario as we do not utilize any white listing techniques to safeguard end-devices from our mitigation approach. Specifically, normal users could have the potential to be prone to mitigation techniques depending on IDS signatures and rule sets.

6.2 Detection Rules in BRO and SNORT

For a detection system to identify adverse traffic, rules are necessary for network traffic evaluation. The following demonstrates the rule to identify a DoS attack for SNORT where an alert is raised once 70 packets are sent within a 10 second interval that is TCP-SYN flagged.

```
alert tcp any any -> $HOME_NET 80(flags:S;
msg:"Possible TCP DoS is Detected !!";
flow: stateless; detection_filter: track by_dist, count 70,
seconds 10; sid 10001;rev:1;)
```

6.3 Traffic Generation Techniques

To mix normal traffic into the grand scheme of our experiments, we utilize iPerf [14]. Although we cannot fully emulate a normal user, we believe that iPerf should provide a fundamental approach to measuring our solution. The main reason for such an approach is that iPerf provides the ability to saturate a network link in addition to real-time network throughput analysis. To be concise, we configure iPerf with the default parameters for operational use.

6.4 Experimental Results

In the evaluation of KernelDetect, we study its inspection time, mitigation time, detection accuracy, and system resource consumption comparatively compared to SNORT and BRO.

Inspection Time. Packet inspection time is critical to the mitigation of threat actors and adverse network traffic. Specifically, as packets arrive at an IDS, the information is placed in a buffer and waits for inspection. This waiting time increases the time needed to mitigate the adverse threat if the packet has malicious intent. To measure such inspection time, we established a near-equal configuration for each IDS solution with the quantities of threat signatures in each respective database for measurement purposes. We establish this approach to examining the effectiveness of each solution to have near mirror-like configurations and to examine the performance of each IDS solution closely.

To measure inspection time, we establish communication between two devices using hping3 where we transmit low packet size with the large quantity of traffic at one nanosecond interval of time, achieving a link saturation. We evaluate three threshold values of 5, 10, and 15 s for detection. In this evaluation, we run all the experiments 10 times and then averge their results. Figure 4 shows the average inspection time for 10-second thresholds. Our experimental results demonstrate that KernelDetect has an overall lowest average inspection time compared to SNORT and BRO.

Table 1. A comparison of the average inspection time in seconds among KernelDetect, SNORT and BRO under various traffic loads of 100K, 200K, and 500K SYN flagged packets using detection thresholds of 5, 10, and 15 s.

Traffic load (K)		100	200	500
Threshold (Sec.)	IDS			
5	KernelDetect	0.0048	0.0047	0.0109
	SNORT	0.0033	0.0186	0.0319
	BRO	2.1264	1.5187	2.3996
10	KernelDetect	0.0106	0.0111	0.0112
	SNORT	0.0128	02686	0.0643
	BRO	2.2656	1.1270	4.3337
15	KernelDetect	0.0067	0.0070	0.0069
	SNORT	0.0067	0.0243	0.0172
	BRO	1.9113	1.3433	2.5786

Table 1 demonstrates that KernelDetect has lower inspection time average comparatively to BRO and SNORT while Table 2 describes a 95% confidence interval statistic. In Table 2, we only compare KernelDetect with SNORT because BRO has much higher inspection time than KernelDetect and SNORT

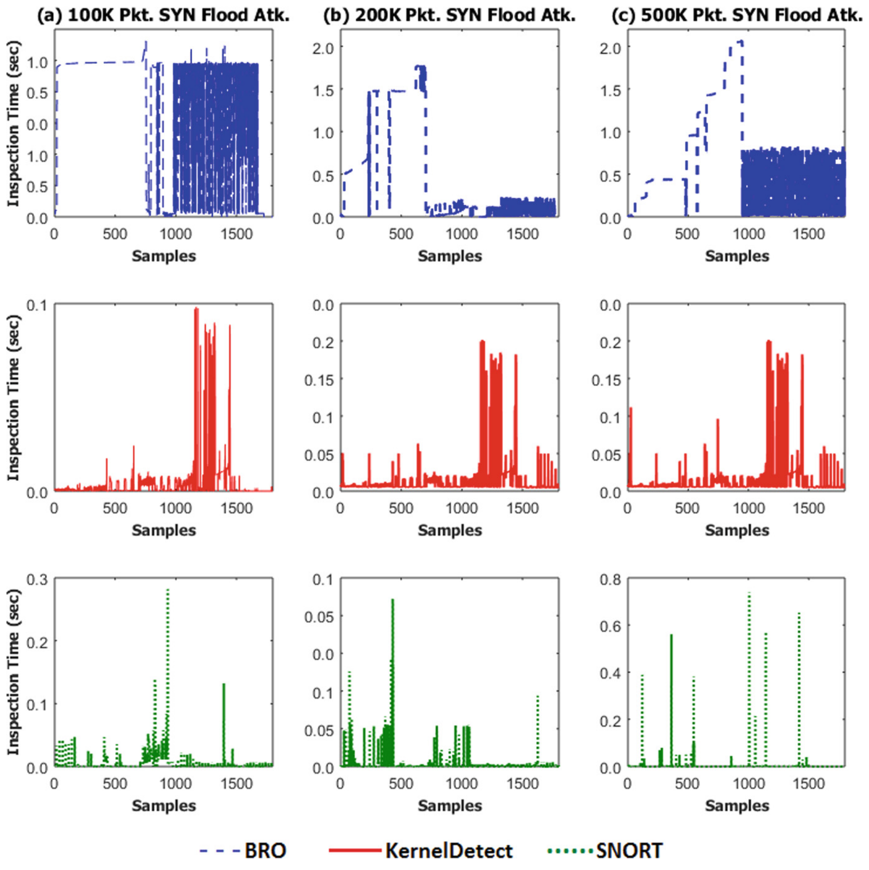

Fig. 4. A comparative analysis of the inspection time for each IDS under 100K, 200K and 500K SYN flagged packet DoS attack using a 10 s threshold signature.

as shown in Table 1 so that it would not be helpful even if we included a 95% confidence interval statistic for BRO in the table. Furthermore, although KernelDetect has some confidence interval overlap with SNORT, it is demonstrated in Table 2 that KernelDetect is still a clear winner in comparison to SNORT under various traffic loads with different signature threshold values regarding inspection time.

Mitigation Time: Mitigation time is the time between an alert raised and the threat stopped. It is key to ensuring the safety and well-being of a network at scale. Although each IDS solution presents its unique attributes to detect an adverse threat, we measure the effectiveness of KernelDetect by studying threat mitigation. To measure the mitigation time, we examine the time between the initiation of each network attack and compared it to the time needed to rectify the threat as expressed in Fig. 5, represented in Table 3 as an average, and described using a 95% confidence interval in Table 4. Table 3 depicts that

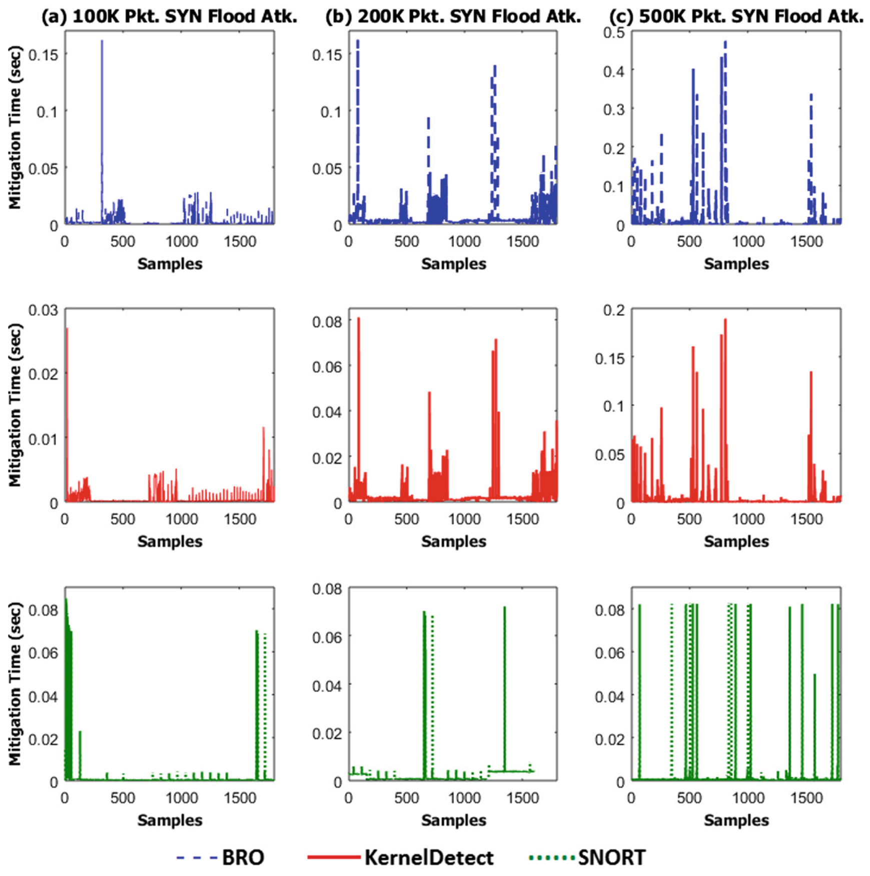

Fig. 5. Threat mitigation time for each IDS under 100K, 200K and 500K SYN flagged packet DoS attack using a 10 s threshold.

Table 2. A 95% confidence interval statistic for inspection time (seconds) between KernelDetect and SNORT where L and U represent their lower and upper bound, respectively.

Traffic load (K)		100			200			500		
Threshold (Sec.)	IDS	Stdev	L	U	Stdev	L	U	Stdev	L	U
5	KernelDetect	0.0142	0.0024	0.0031	0.0025	0.0037	0.0056	0.2009	0.0066	0.0150
	SNORT	0.0229	0.0027	0.0039	0.0624	0.0162	0.0211	0.3644	0.0170	0.0470
10	KernelDetect	0.1711	0.0038	0.0174	0.0183	0.0103	0.0119	0.0186	0.0104	0.0120
	SNORT	0.2309	0.0082	0.0173	0.2217	0.2549	0.2824	0.2566	0.0592	0.0690
15	KernelDetect	0.0522	0.0037	0.0068	0.0602	0.0051	0.0088	0.0654	0.0055	0.0080
	SNORT	0.0558	0.0050	0.0083	0.0984	0.0195	0.0253	0.1205	0.0137	0.0210

Table 3. The average mitigation time in seconds for KernelDetect, SNORT and BRO under various traffic loads of 100K, 200K, and 500K SYN flagged packets using detection thresholds of 5, 10, and 15 s.

Traffic load (K)		100	200	500
Threshold (Sec.)	IDS			
5	KernelDetect	0.0036	0.0074	0.0123
	SNORT	0.0056	0.0012	0.0130
	BRO	0.0060	0.0100	0.0108
10	KernelDetect	0.0060	0.0054	0.0044
	SNORT	0.0055	0.0065	0.0078
	BRO	0.0074	0.0080	0.0118
15	KernelDetect	0.0042	0.0049	0.0086
	SNORT	0.0071	0.0101	0.0200
	BRO	0.0096	0.0635	0.1254

Table 4. A 95% confidence interval measurements for mitigation time (seconds) between KernelDetect and SNORT where L and U represent their lower and upper bound, respectively.

Traffic load (K)		100			200			500		
Threshold (Sec.)	IDS	Stdev	L	U	Stdev	L	U	Stdev	L	U
5	KernelDetect	0.0141	0.0032	0.0040	0.0376	0.0063	0.0087	0.0966	0.0097	0.0149
	SNORT	0.0424	0.0044	0.0067	0.1137	0.0078	0.0167	0.0820	0.0094	0.0166
10	KernelDetect	0.0302	0.0051	0.0068	0.0355	0.0045	0.0063	0.0230	0.0038	0.0050
	SNORT	0.0542	0.0027	0.0037	0.0530	0.0026	0.0037	0.0190	0.0081	0.0093
15	KernelDetect	0.0267	0.0035	0.0060	0.0656	0.0030	0.0067	0.0629	0.0068	0.0104
	SNORT	0.0494	0.0058	0.0085	0.0921	0.0065	0.0137	0.0932	0.0096	0.0240

KernelDetect has a similar mitigation time as SNORT, it is superior to BRO and is still slightly better than SNORT on average. Furthermore, similar to Table 2, we do not include BRO in Table 4 for the same reason. As shown in Table 4, KernelDetect has better performance than SNORT when comparing their confidence intervals. Figure 5 depicts a series of DoS attacks executed in the SDN environment where each solution provided necessary alerting and mitigation procedures. The mitigation technique for each solution utilizes the same function such that when an alarm rose, the message presented will be used to block the respective address. Figure 6 provides the clarity of mitigation time using 10 s threshold, which demonstrates that KernelDetect is the best solution.

True Positive and False Positive. False positive and erroneous threat detection can lead to significant downfalls of network communication. Figure 7 presents the use of Receiver Operating Characteristic (ROC) curve techniques for KernelDetect SNORT and BRO. Notably, the curve demonstrates our detection matching sensitivity for our experimental evaluation where we identify the accuracy of each system. BRO demonstrated to have the poorest accuracy rate

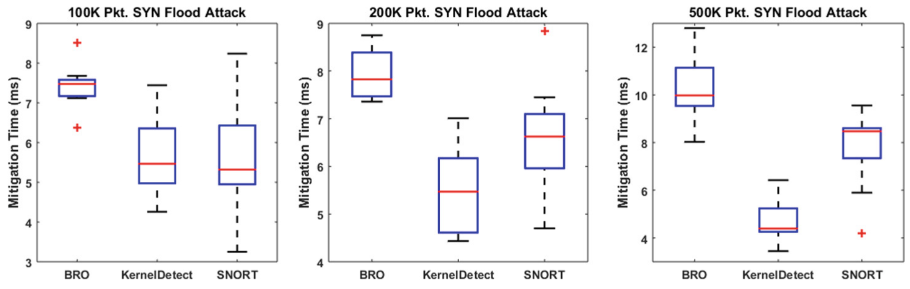

Fig. 6. Threat mitigation for each IDS under 100K, 200K, and 500K SYN flagged packet flood attack using 10 s threshold detection technique and represented as a box plot.

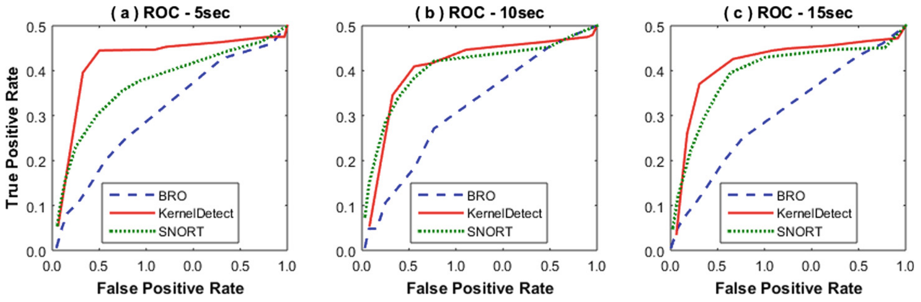

Fig. 7. ROC Curve for threat detection for each IDS under 100K Packets SYN flood attack using various thresholds of 5, 10 and 15 s for threat signatures of a DoS attack.

in comparison to KernelDetect and SNORT where KernelDetect presented the most accurate results based on the analysis of the experimental results.

System Resource Utilization. The performance of an IDS/IPS solution is critical to counter adverse network threats and specifically—threat actors. As traffic flows from one host to another, congestion and computational bottlenecks can occur within a network environment in addition to an IDS solution. Inspection and the level of detail in examining the content of the packet can produce resource strain on a computing device. Figure 8 provides the average system resource utilization for each IDS solution under a variety of network attacks for purposes of evaluating the performance constraint. Samples of system resource utilization are used to measure averaging CPU usage in a kernel space. As shown in experiments, BRO demonstrated a higher-level system resource utilization in comparison to KernelDetect and Snort. Although CPU utilization is critical to examine, memory resource consumption is vital in the operation of an SDN device.

Memory is a critical segment for resource examination as network packets traverse between two devices. The information is stored in a buffer, waiting for inspection and forwarding purposes. In Fig. 9, we express our findings for memory usage under a 100K SYN flagged packet attack.

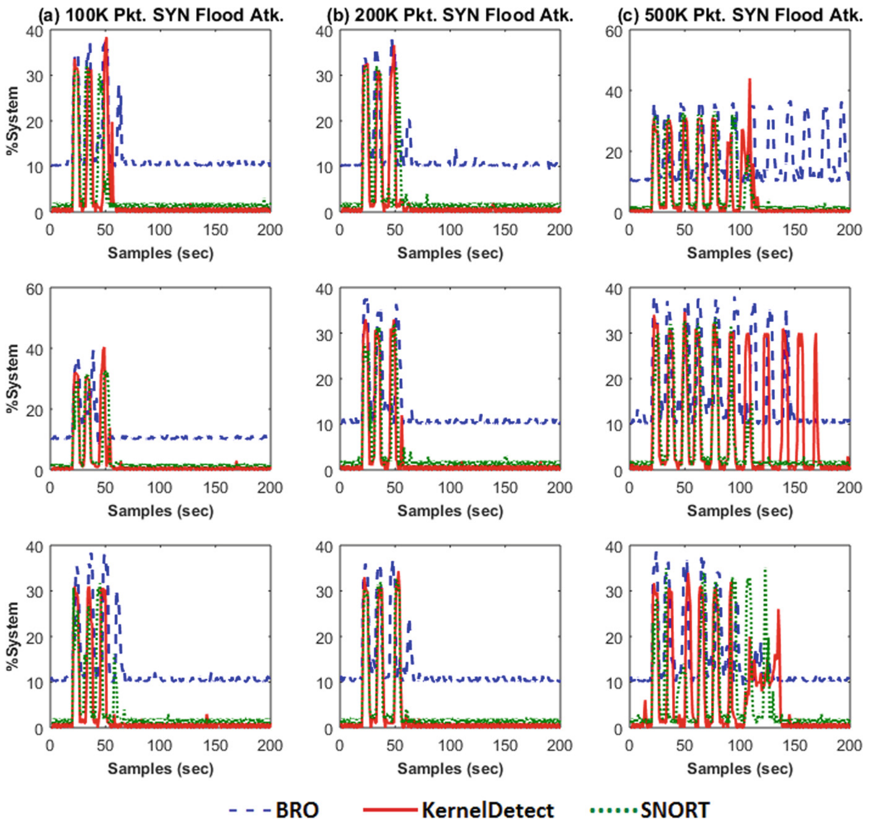

Fig. 8. System usage at 100K, 200K, and 500K packets (Pkt.) loads of SYN flood attack (Atk.)

Memory utilization increases during the events of a DoS attack where KernelDetect is more efficient than SNORT and BRO in the events of a post-DoS scenario. To be concise, once the DoS attack ends, both BRO and SNORT maintain constant memory resource utilization while KernelDetect's usage reduces to the lowest percentage rate.

Discussions. A kernel panic is one serious challenge in the use of kernel-space detection for a security apparatus such as KernelDetect. Moreover, if a kernel panic would occur to an SDN device, practical and operational usage would be lost. Additionally, the library functions that are implemented and imported into the design of KernelDetect may propose a vulnerability that could be haphazardous to the SDN environment. In the design of KernelDetect, this research treats all utilized libraries as trusted modules in the implementation such that the discovery of a serious vulnerability would be well-known and urgent for patching purposes. One configuration that may be sub-optimal for an SDN environment is to implement KernelDetect on an independent computing system

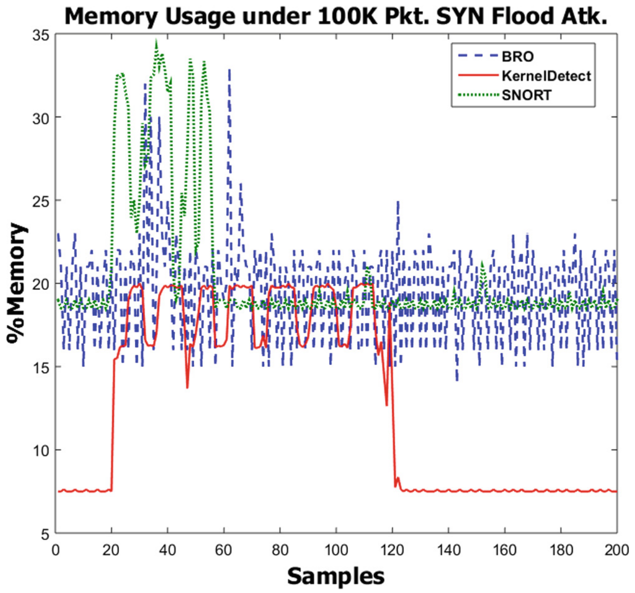

Fig. 9. Memory utilization for each IDS under the scenario of a single attack with 100K SYN flagged packets (Pkt.) using a 10 s threshold detection rate. Additionally, a full-link saturation is achieved during the attack in this evaluation.

that is attached to a port mirroring interface using a SPAN/TAP configuration such that if kernel panic would occur, SDN switching operations would continue to function. Lastly, KernelDetect utilizes raw socket information to read incoming packets. This read procedure could be insufficient for the switching operation such that OVS could process the raw socket information at a faster rate than KernelDetect. We experimented by creating a low packet size full link saturation scenario, but we were unable to emulate the concern. Our belief to such an event would potentially be plausible in a large network throughput interfaces such as 100Gbps. However, our evaluation was limited to only 1 Gbps speeds. Peformance modeling like [41] is helpful to such studies.

7 Conclusions and Future Work

In this paper, we have proposed KernelDetect, a modular countermeasure approach in an SDN environment. It is a new lightweight kernel-level intrusion detection and prevention approach where we have leveraged modular string searching and filtering mechanisms with SDN controller techniques. While KernelDetect is applicable to deal with a variety of adverse network threats, we have specifically explored the events of a DoS attack in an SDN environment.

To combat the above attack, we have considered the Aho-Corasick algorithm that is an exact string matching technique, and Bloom filter that is a partial matching algorithm. In KernelDetect, we have further dynamically leveraged the strengths of the Aho-Corasick algorithm and Bloom filter with SDN controllers. Moreover, we have conducted extensive experiments on GENI, a real-world testbed infrastructure where we have varied the number of network packets ranging from 100K to 500K and launched SYN flooding attacks with different packet sizes and sampling times. We have measured the average load of system resources, inspection time, mitigation time, true positive, false positive, and false negative among 10-run experiments. Section 6 has reported the partial results of our comprehensive experimental evaluation. Through a comparative analysis of KernelDetect with traditional IDS solutions of SNORT and BRO, we have demonstrated that KernelDetect is an effective and efficient solution to detect and mitigate adverse attacks.

We have utilized our inspection approach to detecting network threats within the data plane of an SDN environment. In our future work, we plan to identify the potential areas of threat detection in control plane communications. Moreover, we have stuided DoS attacks in an SDN environment. We plan to examine other adverse threats such as malware where a deep packet inspection is required, and therefore KernelDetect needs to be modified for addressing such threats.

Acknowledgments. We acknowledge National Science Foundation (NSF) to partially sponsor the work under grants #1633978, #1620871, #1620862, and #1636622, and BBN/GPO project #1936 through NSF/CNS grant. We also thank the Florida Center for Cybersecurity for a seed grant. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of NSF.

References

1. Apache Spam Assassin Public Corpus. https://spamassassin.apache.org/publiccorpus/
2. DDoS attack 2007 dataset, CAIDA, UCSD. http://www.caida.org/data/passive/ddos-20070804dataset.xml
3. Floodlight controller. http://www.projectfloodlight.org/floodlight/
4. Mininet: an instant virtual network on your laptop (or other PC). http://mininet.org
5. Akella, A.V., Xiong, K.: Quality of service (QoS)-guaranteed network resource allocation via software defined networking (SDN). In: DASC 2014. IEEE (2014)
6. Chin, T., et al.: An SDN-supported collaborative approach for DDoS flooding detection and containment. In: MILCOM 2015. IEEE (2015)
7. Chin, T., et al.: Selective packet inspection to detect DoS flooding using software defined networking (SDN). In: ICDCSW 2015. IEEE (2015)
8. Chin, T., Xiong, K.: Dynamic generation containment systems (DGCS): a moving target defense approach. In: CPS Week EITEC 2016. IEEE (2016)
9. Chin, T., Xiong, K.: A forensic methodology for software-defined network switches. Advances in Digital Forensics XIII. IAICT, vol. 511, pp. 97–110. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67208-3_6
10. Chin, T., Xiong, K., Rahouti, M.: End-to-end delay minimization approaches using software-defined networking. In: RACS 2017. ACM (2017)

11. Dharmapurikar, S., Lockwood, J.W.: Fast and scalable pattern matching for network intrusion detection systems. JSAC **24**, 1781–1792 (2006)
12. Jackson, E.J., et al.: SoftFlow: a middlebox architecture for Open vSwitch. In: USENIX ATC (2016)
13. Curtis, A.R., et al.: Mahout: low-overhead datacenter traffic management using end-host-based elephant detection. In: INFOCOM (2011)
14. Tirumala, A., et al.: iPerf: the TCP/UDP bandwidth measurement tool (2005). http://dast.nlanr.net/Projects
15. Pfaff, B., et al.: The design and implementation of Open vSwitch. In: USENIX Symposium on NSDI (2015)
16. Chung, C.-J., et al.: NICE: network intrusion detection and counter-measure selection in virtual network systems. TDSC **10**, 198–211 (2013)
17. Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E.P., Ioannidis, S.: Gnort: high performance network intrusion detection using graphics processors. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 116–134. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87403-4_7
18. Mekky, H., et al.: Application-aware data plane processing in SDN. In: HotSDN (2014)
19. Wang, H., et al.: FloodGuard: a DoS attack prevention extension in software-defined networks. In: DSN (2015)
20. Ahrenholz, J., et al.: CORE: a real-time network emulator. In: MILCOM (2008)
21. Amann, J., Sommer, R.: Providing dynamic control to passive network security monitoring. In: Bos, H., Monrose, F., Blanc, G. (eds.) RAID 2015. LNCS, vol. 9404, pp. 133–152. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26362-5_7
22. Ballard, J.R., et al.: Extensible and scalable network monitoring using OpenSAFE. In: INM/WREN (2010)
23. Ko, C., et al.: Detecting and countering system intrusions using software wrappers. In: USENIX Security Symposium (2000)
24. Giotis, K., et al.: Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. Comput. Netw. **62**, 122–136 (2014)
25. Khurshid, K., et al.: VeriFlow: verifying network-wide invariants in real time. In: NSDI (2013)
26. Alicherry, M., et al.: High speed pattern matching for network IDS/IPS. In: ICNP (2006)
27. Berman, M., et al.: GENI: a federated testbed for innovative network experiments. Comput. Netw. **61**, 5–23 (2014)
28. Dhawan, M., et al.: SPHINX: detecting security attacks in software-defined networks. In: NDSS (2015)
29. Roesch, M., et al.: SNORT-lightweight intrusion detection for networks. In: USENIX LISA (1999)
30. Kazemian, P., et al.: Real time network policy checking using header space analysis. In: NSDI (2013)
31. Porras, P., et al.: A security enforcement kernel for OpenFlow networks. In: HotSDN (2012)
32. Wang, R., et al.: An entropy-based distributed DDoS detection mechanism in software-defined networking. In: Trustcom/BigDataSE/ISPA (2015)
33. Avallone, S., et al.: D-ITG: distributed internet traffic generator. In: QEST (2004)
34. Hong, S., et al.: Poisoning network visibility in software-defined networks: new attacks and countermeasures. In: NDSS (2015)

35. Scott-Hayward, S., et al.: A survey of security in software defined networks. IEEE Commun. Surv. Tutor. **18**, 623–654 (2016)
36. Shin, S., et al.: AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In: CCS (2013)
37. Shin, S., et al.: FRESCO: modular composable security services for software-defined networks. In: NDSS (2013)
38. Shin, S., et al.: Rosemary: a robust, secure, and high-performance network operating system. In CCS (2014)
39. Paxson, V.: BRO: a system for detecting network intruders in real-time. Computer Networks (1999)
40. Shin, S., Gu, G.: Attacking software-defined networks: a first feasibility study. In: HotSDN. ACM (2013)
41. Xiong, K.: Multiple priority customer service guarantees in cluster computing. In: IEEE IPDPS, pp. 1–12 (2009)
42. Xiong, K., Wang, R., Du, W., Ning, P.: Containing bogus packet insertion attacks for broadcast authentication in sensor networks. In: TOSN 2012 (2012)