This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

A Novel Stealthy Attack to Gather SDN Configuration-Information

Mauro Conti, *Senior Member, IEEE*, Fabio De Gaspari, Luigi V. Mancini,

***Abstract*—Software Defined Networking (SDN) is a recent network architecture based on the separation of forwarding functions from network logic, and provides high flexibility in the management of the network. In this paper, we show how an attacker can exploit SDN programmability to obtain detailed knowledge about the network behaviour. In particular, we introduce a novel attack, named *Know Your Enemy* (KYE), which allows an attacker to gather vital information about the configuration of the network. Through the KYE attack, an attacker can obtain information ranging from the configuration of security tools, such as attack detection thresholds for network scanning, to general network policies like QoS and network virtualization. Additionally, we show that the KYE attack can be performed in a stealthy fashion, allowing an attacker to learn configuration secrets without being detected. We underline that the vulnerability exploited by the KYE attack is proper of SDN and is not present in legacy networks. Finally, we address the KYE attack by proposing an active defense countermeasure based on network flows obfuscation, which considerably increases the complexity for a successful attack. Our solution offers provable security guarantees that can be tailored to the needs of the specific network under consideration.**

Keywords: SDN, Side-Channel, Stealth Information-Gathering

I. Introduction

Software Defined Networking (SDN) is a recently proposed network architecture which aims to address the shortcomings of traditional architectures by simplifying the management of network infrastructure. SDN is based on the premise that network logic and the physical network devices should not be conflated into a single entity, but rather separated in different layers. To this end, SDN introduces the concepts of *data plane* and *control plane*: the data plane is comprised of the physical network devices (from here on, called switches) and implements the forwarding functions of the network, while the control plane manages the network logic and decision making process. The decisions related to traffic management are taken only by the control plane, which pushes instructions to the switches in the data plane. The data plane is therefore relegated to the role of an actuator, implementing the decisions of the control plane. This separation between logical control and physical implementation of the network functions provides a high degree of flexibility, which is one of the main reason for the widespread adoption of SDN even amongst big companies [9], [39].

Mauro Conti is with the Department of Mathematics, University of Padua, Padua 35131, Italy (e-mail: conti@math.unipd.it).

Fabio De Gaspari is with the Department of Informatics, Sapienza University of Rome, Rome 00198, Italy (e-mail: degaspari@di.uniroma1.it).

Luigi V. Mancini is with the Department of Informatics, Sapienza University of Rome, Rome 00198, Italy (e-mail: mancini@di.uniroma1.it).

While the programmability of SDN allows for fast prototyping and high adaptability to different scenarios, it also creates new vulnerabilities [21], [22], [46]. In this paper, we show that the centralization of the decision making process in the control plane, and the consequent distribution of the policy enforcement throughout the switches, introduces a new interesting venue for attack. We show that an attacker can exploit the distributed policy-enforcement of SDN in order to gather intelligence about the control logic of the network in a stealthy fashion. In particular, our attack only requires the adversary to obtain a flow table side-channel, i.e., a way of learning which rules are installed, for a single switch (which we call *entry switch*, see Section II). By analyzing the conditions under which a rule is pushed, and the type of such rule, an attacker can infer sensitive information regarding the configuration of the network. The final result is that through a single switch, the attacker can gather information which, in a classical network, would have required access to numerous distinct devices, such as firewalls, intrusion detection/prevention systems, etc. The information gathered can subsequently be exploited to mount different attacks, tailored to the target network, without being detected. Finally, we propose a configurable countermeasure to the KYE attack, which allows network managers to choose the best trade-off based on the needs of the specific SDN.

To summarize, in this paper, which is an extended version of our work in [30], we make the following contributions:

- We propose a novel, attack, the *Know Your Enemy* (KYE) attack, that allows stealth intelligence gathering about the configuration of a target SDN network. The information that an adversary can obtain ranges from configuration of security tools, such as attack detection thresholds for network scanning, to general network policies like QoS and network virtualization.
- We prove the feasibility and efficacy of the KYE attack through its implementation and a thorough experimental evaluation on a test network.
- We further discuss new, concrete instances of the KYE attack. In particular, we explain how the attack can be used to infer SDN-related configuration information, such as control plane scalability and flow table saturation countermeasures.
- We further propose a new countermeasure to the KYE attack, called *flow obfuscation*. The countermeasure, which is based on the obfuscation of inbound network flows, is configurable: this allows network managers to tailor it based on the vulnerability of the considered network.
- We experimentally evaluate our countermeasure and provide an accurate evaluation of the overhead it introduces.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

- We analyse the security of the proposed coutnermeasure, along with potential solutions to improve its efficiency.

The remainder of this paper is organised as follows. In Section II we introduce our threat model and assumptions. Section III presents the details of the KYE attack and Section IV describes several possible instances of the attack. In Section V we describe our experimental evaluation of the KYE attack and present our results. Section VI describes our contermeasure to the KYE attack, as well as the results of our experimental evaluation of the countermeasure. Section VII discussess the limitations of the KYE attack and venues for possible improvements from the point of view of attack automation. In Section VIII we discuss related works and we present our conclusions in Section IX.

II. Assumptions and SDN Issues

While having a logically centralized point of control allows to improve the decision-making process, distributing the policy enforcement introduces new problems with regard to information disclosure. Where network functions in legacy networks are relegated to the specific devices implementing them, providing higher control over access to their configuration, in SDN they are distributed throughout the OpenFlow switches. Indeed, network policies and functions like intrusion detection/prevention systems (IDS/IPS), network virtualization, or access control, are often enforced by the OpenFlow switches, through the application of the flow rules installed by the controller [29], [33], [34]. Unfortunately, this behaviour considerably broadens the attack surface for an attacker.

As we show in this paper, by having a flow table side-channel on a single OpenFlow switch (called *entry switch*), an attacker can gather a relevant amount of information regarding the behaviour and configuration of the SDN network the switch belongs to. A flow table side-channel is defined as any mean by which an attacker can learn or infer the content of the flow table of a switch. The KYE attack is independent from how this side-channel is obtained, and the detailed description of how to obtain it is out of the scope of this paper. However, for completeness, we list some of the possibilities an attacker could leverage:

- Connect to a passive listening port on the entry switch to retrieve the flow table. In fact, most OpenFlow switches can be remotely debugged by means of a passive listening port, which also allows retrieval of the flow table [26], [47]. For instance, an attacker could use the `dpctl` utility [4] on the unprotected listener port of an HP Procurve [7].
- Exploit poor device configuration settings, such as the use of default or weak passwords. Indeed, misconfiguration is an important source of vulnerabilities in deployed systems [2], [6], and is already identified as one of the main venues of attacks in the literature [28].
- Use Round Trip Time (RTT) variance to infer information about the content of the flow table [31], [41].
- Sniff the control traffic. Indeed, the use of TLS on the control channel is optional [12], and many commercial switches do not support it [26]. Additionally, in most cases OpenFlow switches that support TLS do not implement certificate authentication [47].
- Exploit backdoors built-in by the device manufacturers [1], [5], [8] and government agencies [14].
- Compromise the switch. Indeed, attackers demonstrated multiple times the ability of compromising several different types of switches [3], [15]. In particular, this venue of attack is going to become increasingly important with further studies on switch vulnerability and frameworks for automated vulnerability discovery, such as the recently presented DELTA [50] and FlowFuzz [44].

It is worth noting that the attacker uses the flow table side-channel only to read the state of the flow table, therefore the overall state of the entry switch is not modified in any way. This means that, even in case of a controller monitoring the integrity of the entire state of the switches, e.g., through direct query [40] or checksum [26], the flow table side-channel would not be detected. Through the KYE attack the attacker can learn all sorts of relevant information, from routing policies, to more complex and important behaviours regarding attack detection and defense mechanisms.

We would like to point out that, while OpenFlow allows to pre-install flow rules in the switches, this severely limits the capabilities of SDN, reducing its advantages over classical network architectures. Indeed, most of the novel applications that can be realized with SDN are based on on-demand management of network flows, like reactive defence mechanisms against various types of attacks [53], [37], [45], [22], [54] (see Section IV-A). Using proactive rule installation would also impact network efficiency by preventing dynamic traffic engineering [35], [36], one of the key applications of SDN that allows to considerably reduce the cost of the network infrastructure by eliminating the need of overprovisioning [39]. In brief, dynamic network flow management is one of the main features of SDN and is the key enabler of most SDN applications, therefore we assume that an SDN network will employ reactive rule installation.

Threat Model. Our threat model assumes that the attacker has a flow table side-channel for a single switch. His goal is to gather information on the network without being detected. Therefore, the attacker uses the flow table side-channel only to read the state of the flow table, without modifying the overall state of the entry switch in any way. In particular, the only abilities of the attacker are (*i*) sending packets through the target network and (*ii*) using the side-channel to learn the rules that are installed on the entry switch. The attacker can be either internal or external. Furthermore, we assume that the switch only provides the functionality described in the OpenFlow standard [12], and that the attacker does not increase its capabilities in any way.

In our threat model, the attacker can be modeled as a host connected to the target network, either directly or indirectly, who can generate arbitrary traffic though the network. Additionally, the attacker has the ability to arbitrarily read the flow table of one switch in the network and identify which flow rules are installed in response to his traffic.

III. The KYE Attack

The details of the attack vary based on the specific information the attacker wants to gather (see Section IV). However, all

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

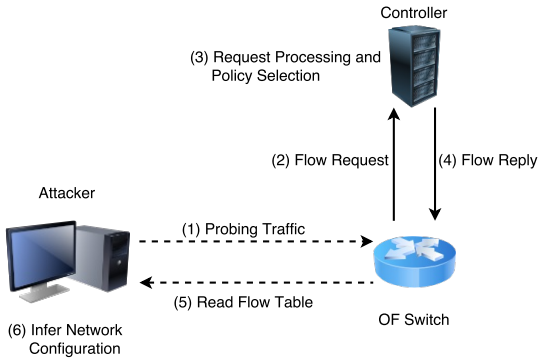


Fig. 1: Overview of a general KYE attack.

instances of the KYE attack share a common kernel, as detailed below. The KYE attack exploits the on-demand installation of rules of SDN, allowing an attacker to gather knowledge about which conditions trigger the installation of a given rule, as illustrated in Figure 1. The KYE attack is structured in two phases: (*A*) the *probing phase* and (*B*) the *inference phase*.

In the probing phase (*A*), which is repeated numerous times, the attacker attempts to trigger the installation of flow rules on the entry switch (steps 1 through 5 in Figure 1):

- The attacker sends carefully crafted *probing traffic* through the entry switch in order to trigger the installation of new flow rules. The specific characteristics of the probing traffic depend on what kind of information the attacker is interested in learning (see Section IV and Section V).
- Through the flow table side-channel, the attacker obtains the flow rule (if any) installed in response to the probing traffic.

In the inference phase (*B*), the attacker analyzes the correlation between the probing traffic sent in the probing phase and the corresponding rules installed (step 6 in Figure 1). From this analysis, it is possible to infer what network policy is enforced for specific types of network flows (see Section IV). Additionally, by studying the features of the probing traffic, the attacker can potentially infer the trigger conditions for network policies that require a specific trigger before being activated (e.g., scanning detection thresholds, see Section V).

From the point of view of the attacker, knowing which type of attacks are detected by the target network, the countermeasures employed and the detection treshold is a great advantage. Ideed, knowing the detection and defence mechanisms employed by the network, the attacker can, for instance, craft malicious payloads specifically designed so that they will not be detected by IDS/IPS, perform DDoS attacks while disguising the traffic so that the attack is not detected, or learn which machines in the network are honeypots (by looking for possible redirection rules towards such hosts). In general, the information provided by the KYE attack allows an attacker to minimise the risk of detection when moving about the target network, and to maximise the potential damage by means of targeted attacks.

It is worth noting that introducing probing traffic in the network does not expose the KYE attack to detection. Indeed, one of the strengths of the KYE attack is that it is hidden by other attacks: for instance, an attacker performing a KYE attack to gather information related to DoS countermeasures will generate DoS traffic through the target network during the probing phase. When the DoS traffic is detected, the network will categorise it as a DoS attack, and the KYE attack will remain undetected. Effectively, the probing traffic hides the real attack to the network, making it extremely hard to detect the KYE attack reliably. Moreover, the attacker can employ IP spoofing or other techniques, such as proxies, during the probing phase, so that it is not possible to link him to the probing traffic.

A. KYE Attack and Network Probing in Traditional Networks

Attackers already employ network probing in today's networks to learn as much information as possible about their target. With this regard, our main contribution is to instantiate the general concept of network probing in SDN networks, and show how an attacker can exploit dynamic flow rule installation to gather a much greater amount of information that traditionally possible. This level of information gathering is not achievable in traditional networks, where probing yields less information due to the separation of network functions across different devices. Moreover, in general the accuracy of the information obtained is also lower in traditional networks, as it is more complex to understand exacly when a given countermeasure is activated (as opposed to an SDN environment, where it is easily detectable by analysing the rules pushed by the controller). Indeed, while in traditional networks it might be possible to understand when a flow is dropped, it becomes increasingly difficult to distinguish more complex (and interesting for the attacker) actions such as, for instance, flow redirection.

IV. KYE Instances

As we discussed before, the KYE attack is a general attack strategy, and the details of the attack vary based on the specific information the attacker wants to gather. In this section, we discuss different instances of the KYE attack, with respect to what type of information the attacker wants to obtain. Additionally, we provide a non-exhaustive list of concrete examples of KYE attack, showing how an attacker can exploit the flow table side-channel to infer different configuration features of the network. Due to space constraints, we limit our discussion to KYE attacks aimed at disclosing security-related configuration information in Section IV-A, and SDN-related configuration information in Section IV-B. However, the KYE attack can be easily employed to disclose almost any type of network-configuration information, such as network virtualization functions or quality of service rules.

A. Gathering Network Security Configuration Information

When planning an attack, knowing what detection and defense mechanisms are used by the target network is obviously invaluable to an attacker. In this section we discuss how, through repeated probing and analysis of the flow table, an attacker can infer detection mechanisms and defense

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

measures in place in an SDN network for different types of attack. Due to space limitations, in our analysis we focus on some popular SDN-based defense mechanisms proposed in the literature [17], [49].

1) Worm Infection/Scanning: Scanning is one of the main preliminary intelligence gathering techniques, used by attackers to gather information about a given target network and by worms to detect vulnerable targets to spread the infection to. Through scanning, an attacker can learn about the number, type and address of hosts in a network, along with what services are offered on which port. This information is a prerequisite for mounting more complex attacks. Therefore, being able to detect and mitigate scanning is extremely important for any network.

KYE Attack. In SDN, an attacker can infer information regarding the type of defense mechanisms used to mitigate scanning and, depending on the detection mechanism employed, the detection threshold for scans. In order to infer such information, an attacker simply needs to send scanning probes from a spoofed address IP_A, varying the characteristics of the scan (e.g., increasing scanning rate, different duration of the scan, various success/failed connection ratios). After each probe, the attacker reads the content of the flow table of the OpenFlow switch and takes note of the new flow rule installed in response to the probe. As long as the scanning is not detected, the flow rules installed simply instruct the switch to forward the traffic coming from IP_A towards different exit ports, based on the destination address. When the scanning rate becomes high enough or the scanning activity lasted long enough for the attack to be detected [45], the controller will install flow rules implementing an appropriate defense measure against scanning.

Detection and Defense Mechanism Inference. Depending on the defense measure used by the network, different types of flow rules will be installed, for instance: traffic filtering [37], rate limiting [45], [58], honeypot redirection [53], or whitehole network approaches [21], [22], [54]. All these defense mechanisms require the installation of very specific flow rules, which differ from the normal flow rules installed when no attack is detected (see Section IV-C). Consequently, by recognizing this change in the type of rules installed, the attacker can learn that the network scanning was detected. Moreover, the flow rules required by these mechanisms are easily identifiable and, just by looking at what rule is installed, the attacker can infer the defense mechanism used by the network (see Section IV-C). Finally, for some defense mechanisms that are activated on-demand by the controller, the attacker can also infer the traffic features that trigger detection by the network. For instance, if we consider TRW-CB [48], which is one of the most frequently used anomaly detection algorithms [24] and is already implemented in SDN [45], [53], the attacker can learn the ratio between successful/unsuccessful connections used as detection criteria through repeated probing. Once the attacker discovers the detection threshold, he is then able to carry out the network scan undetected in a second phase, by alternating unsuccessful scanning with successful connections.

Implementation. In Section V-A we report on the implementation of this instance of the KYE attack. We show that, for realistic detection and defense mechanisms, an attacker is able to learn both the scanning traffic features that trigger detection, and the defense mechanism used.

2) Denial of Service: DDoS attacks are one of the most widespread type of attacks and their diffusion and sophistication increases every year [11]. Indeed, attackers often use DDoS attacks as a smokescreen to cover data theft and malware installation on target systems [11]. Moreover, the financial damage these attacks cause can range in hundreds of thousands of dollars in peak hours [16]. As a consequence, most organizations adopt one or more DDoS detection and mitigation system [16]. In the context of SDN, DDoS detection schemes tend to employ lightweight mechanisms, such as threshold and entropy-based systems [37], [33] to avoid overloading the controller. Other more complex and computationally expensive approaches do exist, like [27], where the authors employ machine learning techniques to detect possible DoS attacks. Depending on which detection mechanism is used, an attacker can perform a KYE attack to learn if a DoS detection mechanism exists, what defense measure is applied by the network and potentially even the detection criteria employed.

KYE Attack. The KYE Attack for DoS detection is very similar to the one used for network scanning; in the probing phase, the attacker starts a DoS with a low attack rate, simulating a behaviour that is as close as possible to that of a legit client, then gradually increases the profile of the attack. Throughout the attack, the attacker monitors the flow rules installed by the controller on the OpenFlow switch, looking for a change in the flow rules pushed. Indeed, under normal circumstances the controller will simply instruct the switch to route the traffic towards the destination host. However, when the DoS attack is detected, the controller pushes different rules based on the defense mechanism employed by the network, allowing the attacker to learn that the DoS was detected.

Detection and Defense Mechanisms Inference. Defense mechanisms proposed against DoS attacks, like traffic redirection [43], rate limiting [58], [45] or traffic filtering [37], require very specific flow rules that an attacker can easily distinguish from normal routing rules (see Section IV-C). Therefore, as soon as these security flow rules are installed on the monitored OpenFlow switch, the attacker will know that the DoS was detected. By analyzing the specific flow rules installed, he can also infer the defense mechanism applied (see Section IV-C). Additionally, when the attack is detected, the attacker can try to infer the detection criteria used by the network. Indeed, for certain type of detection mechanisms such as threshold [33] or entropy-based [37], it is possible to find a good approximation of when exactly an attack is detected. In order to learn these detection thresholds, an attacker can repeat the probing phase several times varying the characteristics of the attack. Upon detection, the attacker will log such characteristics, like duration, attack rate and number of packets sent from each IP address, for instance. After obtaining a sufficiently large sample, the attacker can look for correlations between the characteristics of the detected attacks

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

to learn approximately what values trigger the detection. Even in case of more complex detection systems based on machine learning [27], the attacker can still obtain knowledge about the traffic features used for detection. Indeed, previous work on the area of machine learning shows that it is possible to infer meaningful information about the training set of a classifier [25], which in our case would reveal information about which flows are considered malicious or benign.

3) Access Control: Access control mechanisms, like firewalls, are the first and most basic defense mechanism used by networks to enforce security policies. Through its centralized view of the network and distributed enforcement of rules, SDN provides the optimal functionality to implement a consistent distributed firewall in the network [38], [57], [59]. Given that such access control systems are the first defense mechanism that an attacker needs to bypass before attacking a network, learning the exact configuration of such devices would provide a huge advantage in preparing an attack. While this would be extremely challenging in classical networks, due to the fact that access control rules are relegated only to specific security devices, this task is considerably simpler in SDN.

KYE Attack. By performing a KYE attack, the attacker can infer all the access control policies he is interested in by simply probing the monitored OpenFlow switch. In its most basic form, the attacker will send a probe packet to test any given access control policy. For instance, the attacker can try to connect to a protected service using a set of different IPs, in order to understand which subnets are allowed to access that service. For each of these probes the controller will push either a forwarding flow rule, to forward the packets to their destination, or a *drop* rule if the access is not allowed [20]. By repeating the probing for all interesting services and using different source IP addresses, the attacker can map which addresses (or address ranges) are allowed towards/from a certain critical service.

Defense Mechanism Inference. For access control enforcement, the policy that is most used in general is to drop unauthorized traffic flows [20]. If such defense mechanism is in place, the attacker is able to recognize it immediately just by reading the new flow rule installed (see Section IV-C). SDN also allows for more complex defense mechanisms to enforce access control, like traffic redirection towards a honeypot/IDS for instance [43], [53]. As we discuss in Section IV-C, the attacker can easily identify even this more complex mechanisms just through observation of the OpenFlow switch flow table.

Implementation. In Section V-B we report on the implementation of this instance of the KYE attack. We show that, given an access control mechanism, an attacker is able to learn the complete access control matrix enforced by the controller.

B. Gathering SDN-Related Configuration Information

Through a KYE attack, an attacker can infer vital information about SDN-related network configuration, such as flow table management [19], [32] and control plane scalability measures [21], [54]. Knowledge about the configuration of such critical systems provides an attacker with a wider attack surface, as well as allowing him to better focus his resources during an attack.

1) Flow Table Saturation: Flow rules, and therefore flow tables, are the core enablers of SDN in OpenFlow. While fine-grained flow rules allow for targeted policy enforcement, the number of flow rules that can be installed on OpenFlow switches is limited. If the flow rule limit is reached (e.g., as a result of a deliberate saturation attack), the switch will not be able to accept rules for new inbound network flows, which will be ignored. To mitigate this vulnerability, the control plane can employ wildcard rules to aggregate the management of multiple network flows with a single flow rule [19], [32]. From the point of view of an attacker, learning under which conditions the controller uses wildcard rules and, more importantly, how to force it to install targeted flow rules, is essential to successfully mount saturation attacks.

The main consideration behind aggregate network flow management is that, in general, it is important to route flows on an individual basis only under certain conditions. In particular, for the purpose of network engineering, this conditions are generally related to the weight of a specific network flow; network flows that surpass a certain threshold (be it a data rate [19] or size [32] threshold), are marked for individual routing through targeted flow rules.

KYE Attack. Through a KYE attack an attacker can infer the thresholds used to classify network flows, allowing him to flood the network with flows for which the controller will generate targeted flow rules. Indeed, similarly to a KYE attack against threshold based detection systems (see Section IV-A1), an attacker can easily detect if such aggregation systems are in place and, if so, what thresholds they employ. As a first step, the attacker reads the OpenFlow switch flow table and creates a new network flow not matching any flow rule present. If network flow aggregation is in use, the new flow rule installed on the switch will be a wildcard flow rule, otherwise it will be a flow rule targeted specifically at the new network flow. If a wildcard rule is installed, the attacker can then increase the data rate transmission for that network flow until a new, flow-specific rule is installed by the controller. At this point, the attacker knows the exact attack rate and/or size of the flow (cumulative amount of data transmitted) required for the controller to install a targeted flow rule.

C. Correlating Flow Rules and Network Policies

Through the KYE attack, an attacker can infer the exact network-level defense mechanism employed against specific attacks. In this section, we present a non-exhaustive set of defense policies [29] that are used in relation to our examples in Section IV-A. Furthermore, we explain how an attacker can correlate a sequence of flow rules obtained during the probing phase to the network policy they implement.

1) Traffic Filtering: One of the most basic network-level defense mechanisms is traffic filtering. A traffic filtering policy can be employed to mitigate a large range of attacks, including scanning and DoS attacks [21], [37]. In SDN, traffic filtering is implemented simply by installing a *drop* rule matching the offending network flows on OpenFlow switches. An attacker

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

monitoring the flow table of an OpenFlow switch can easily detect the application of such defense mechanism. Indeed, before the policy is applied, the control plane will push on the OpenFlow switch normal flow rules, instructing the switch to forward the inbound traffic based on the destination address. When the attack is detected and the filtering is applied, the control plane will install only a single drop rule for the all the traffic coming from the attacker's IP address.

2) Rate Limiting: Rate limiting is a simple, yet effective defense mechanism to mitigate a wide variety of attacks like scanning and DoS [45], [58]. The most basic rate limiters, the ones assigning a maximum bandwidth to a given aggregate of network flows, are immediately recognizable for an attacker since they are directly defined in the flow rule matching the aggregate network flows [12]. More complex rate limiting approaches like those implemented in [45], [58] limit the rate of new network flows by delaying the installation of flow rules. In particular, [58] introduces the notion of working set: for each given host, its working set is defined as the set of recently contacted hosts. Whenever a host creates a new network flow addressed at a host outside its working set, the controller will withhold the installation of the corresponding flow rule on the switch for a certain time. After this waiting time expires, the controller instructs the switch to forward the network flow without installing a flow rule. Only when the switch receives a positive reply from the destination host, the controller will install a new flow rule on the switch. An attacker can infer the presence of this defense mechanism by probing the flow table of the switch after creating a new network flow. If rate limiting is in use, the attacker will receive a response as soon as the flow rule is installed in the OpenFlow switch. Conversely, when no such technique is used, there will be a measurable delay between the installation of the flow rule on the switch and the moment the reply is received. Therefore, by monitoring the delay in receiving a response after a flow rule is installed, the attacker can infer the use of this defense mechanism.

3) Whitehole Network: Another defense mechanism proposed in the literature is SYN proxy [21], [54]. SYN proxy techniques aim at countering the SDN-specific control plane saturation attack [21], [54]. Additionally, SYN proxy implements a whitehole network [54] at the switch level, providing mitigation also against network scans. The attacker can infer the existence and the exact type of the defense mechanism employed by the network. Indeed when SYN proxy techniques are used, the attacker will receive a response SYN-ACK packet *without a flow rule being installed on the OpenFlow switch* [54]. Additionally, the attacker will always receive a response SYN-ACK packet to each and every of his probes, even if directed to himself. This behaviour exposes the use of proxy techniques at the data plane level to the attacker, who can attack the OpenFlow switches through vulnerabilities of the proxy approach [21].

4) Traffic Redirection: Traffic redirection is a widely-used defense mechanism [43], [51], [53]. For instance, it allows a defender to opportunistically route malicious traffic towards a honeypot, isolating the attacker to study his behaviour [53]. In SDN these defense mechanisms are easy to detect for an attacker. By monitoring how the control plane updates flow rule entries for some given network flows, the attacker can infer if his attack traffic is diverted towards a security middlebox/honeypot, nullifying the effect of the countermeasure. An attacker first generates a new legit network flow towards a given destination D_1 which is routed through a port P_i on the entry switch. The attacker then repeats this step with different destinations, until for a given destination D_n the controller pushes a flow rule instructing the switch to output the matching flow on a port $P_j <> P_i$. As a second step, the attacker generates probing traffic with a high profile (e.g., high scanning or DoS rate) towards destinations D_1 and D_n for a length of time, observing the flow rules installed in response. If traffic redirection techniques are in place, the control plane will install on the switch a new rule diverting the attack traffic towards the remote middlebox/honeypot. Therefore, all the attack traffic will be routed through the same output port on the OpenFlow switch. Since in the first phase the attacker selected the destinations D_1 and D_n such that packets towards them would be output on different ports, if all attack packets towards those same destinations are tunneled through the same port, then a redirection mechanism is present in the network.

V. KYE Simulation Studies

In order to prove the feasibility and effectiveness of the KYE attack, we implemented two instances of the attack on a test network. In this section, we present the detailed implementation of our attacks, that were aimed at disclosing:

1) The presence of a scanning detection and defense mechanism in the network. If present, we also wanted to estimate the detection threshold.
2) The presence of a subnetwork access control mechanism. If present, we wanted to learn the subnetwork access control matrix.

Figure 2 depicts the setup used for the evaluation, which includes: the attacker h_0, a single OpenFlow switch s_1 connected to the controller c, and 100 legit hosts $h_1 - h_{100}$. Hosts $h_1 - h_{100}$ represent known web servers that always reply to connection requests. Hosts $h_1 - h_{100}$, if needed, may be used by the attacker to obtain different connection success ratios during probing, and are not necessarily part of the target network.

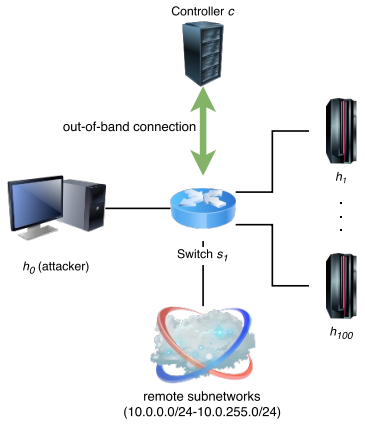


Fig. 2: Experimental evaluation setup.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

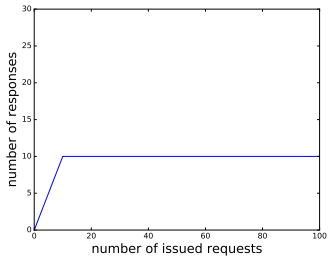


Fig. 3: Number of responses received vs. number of requests issued towards h_1, h_{100} at each batch.

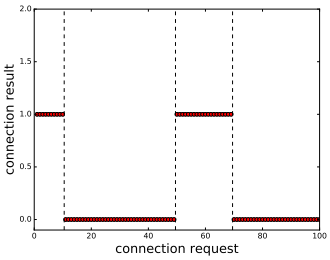


Fig. 4: Connection result for each request of the batch sent towards a known host. 1 indicates a successful connection, 0 a failure (i.e., no response received).

In order to simplify the simulation, in our experiments the target network is comprised only of the switch s_1 and the network controller c. Even though in our test network we deployed only a single controller and a single switch, our experiments do not lose generality. This is because the logic used by the controller is the same that would be used in a more complex network, and the rules pushed by the controller are also the same. It is worth noting that it is not a requirement for the attacker to be directly connected to the OpenFlow switch, nor it is for hosts $h_1 - h_{100}$. This is just a simplification we adopted in order to run our simulation. In this test network, we implemented the TRW-CB scanning detection algorithm, which is one of the most used anomaly detection algorithms [24]. Our implementation of TRW-CB follows the SDN implementation detailed in [45]. The test network also implements an IP-based access control mechanism. We ran all our experiments in a simulated network using the Mininet network simulator [10] and the POX network controller [13].

A. Disclosing Scanning Detection and Defense Mechanisms

The target network implements TRW-CB at the control plane as a scanning detection mechanism, and traffic filtering as a defense measure (see Section IV-C). TRW-CB employs both credit limiting, used to limit the amount of first contact connections pending, and sequential hypothesis testing to detect scanning hosts. In particular, we configured the TRW-CB algorithm with the same parameters used in the original paper [48] (base credit of 10, false positive rate ≤ 0.00005, precision ≥ 0.99).

As an attacker, the first step is to learn if the target network has any kind of scanning detection mechanism in place. To this end, we first initiated a scan with high packet/sec ratio using a spoofed IP, towards a remote subnet which is protected by the target SDN. At the same time, we constantly monitored the flow table for a flow rule implementing a defense mechanism, which would indicate the presence of a scanning detection mechanism. With our configuration, after the first 10 connection attempts failed, TRW-CB correctly identified our probes as scanning activity. Upon detection, the controller pushed a drop flow rule matching all packets coming from the attacker's IP address. Since we were monitoring the flow table of the switch, we detected the rule installation and concluded that the network indeed implements a detection mechanism for scanning attacks, as well as that it uses traffic filtering as a defense measure.

After confirming the existence of a scanning detection mechanism, the following step is to learn which detection criteria are used. As discussed in Section IV-A1, in order to do so we initiate several batches of network scans with different characteristics, like scan rate and successful/failed connections ratio. The results of these batches of scans show two visible characteristics:

1) The scanning activity is detected regardless of the scanning rate. This behaviour excludes rate-based scanning detection mechanisms.
2) For scan batches where connection requests were sent only towards h_1, h_{100} (which should all send back a response), we received replies only from some of them, as illustrated in Figure 3.

The behaviour shown in Figure 3 is consistent with rate limiting techniques, where connection requests sent at a rate above a certain threshold are dropped. To investigate this anomaly, we started a new scan towards h_1, h_{100} with slightly lower rate. The results are illustrated in Figure 4. As we can see, connections are allowed in bursts: replies were received for the first 10 connections, then 39 requests were dropped, after which connection attempts 50 through 69 were successful, and then connections were dropped once again. Since the scanning rate was constant for all the 100 connection attempts, this behaviour excludes a standard rate limiting technique. Indeed, from Figure 4 we can see how a host is allowed to contact up to 10 new hosts (i.e., 10 starting credits), after which connections are blocked until pending replies are received. Upon receiving the replies, new credits are allocated to the host, whose connections are correctly forwarded once again.

This pattern is consistent with the presence of a credit based rate limiting mechanism [48]. At this point, through the KYE attack, we learned that:

- The network is using a detection mechanism for scanning, which is not rate based.
- The network is using drop rules as a defense mechanism against scanning. Moreover, the network employs an additional preemptive defense measure in the form of credit based rate limiting. Each host is assigned a starting balance of 10 credits and for each successful connection the hosts receives 2 additional credits (the initial 10 successful connections allowed 20 more connections after replies were received).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

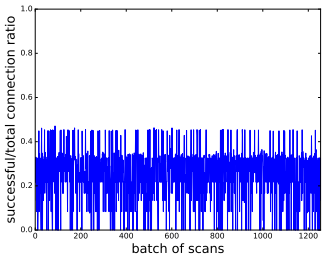


Fig. 5: Ratio of the number of successful connections over the number of total connections, for each batch of scans which resulted in detection.

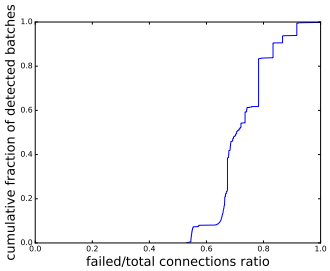


Fig. 6: Empirical cumulative distribution function of the ratio of failed connections over total number of connections, for batches of scans which resulted in detection.

In the final step of the KYE attack, we initiated several batches of scans, with varying successful/failed connection ratios and scan duration. Each scanning batch terminated either after all planned scans were preformed, or abruptly upon detection. Since we are only interested in the characteristics of the scanning attacks that are detected, we isolate detected batches from undetected ones. For the batches of scans that were detected, figure 5 shows the ratio of successful connections over the total number of connections issued and Figure 6 shows the cumulative distribution function of the ratio of failed connections over total number of connections. As these two figures show, the scanning detection criteria employed by the network is clearly based on the ratio of successful and failed connections. Indeed, from Figure 5 we see that network scans are never detected when the ratio of successful connection over the total number of issued connections is above ~ 0.45. Conversely, from Figure 6 we can see that network scans are detected when the ratio of failed connections over total number of issued connections is above ~ 0.55, and never detected when it is below that threshold.

B. Disclosing the Subnetwork Access Control Matrix

After learning the scanning detection criteria used by the network through a KYE attack, we show that we can also infer the complete access control matrix used by the network without being detected by the controller. In our test network, we configured the POX controller with a set of static access control policies, where access to a certain subnetwork is allowed only from a subset of all subnetworks. Whenever a connection request from an unauthorized address is received, the controller instructs the switch to drop the packet without installing any flow rule. If the connection request is from an authorized address, the controller installs a normal forwarding flow rule on the switch for subsequent packets. In this setting, we perform a KYE attack, sending scan probes from each subnetwork to every other. Since with the previous attack we inferred the detection criteria used by the network for scans, *we can now perform this network scan attack completely undetected.*

The attack itself is very simple: at each scan probe, we spoof the source address to make it look like the source of the connection is part of a given subnetwork. We repeat the scan for each pair of source and destination remote subnetworks in the range $10.0.0.0\backslash 24 - 10.0.255.0\backslash 24$, while opening enough successful connections to remain below the detection threshold. After each scan, we read the flow table of the switch to detect which rule is installed. By monitoring the flow table, we see that no flow rules are installed when the source IP is not allowed to access the subnetwork, while a forwarding rule is installed when the access is authorized. Through this observation we are able to build the access control matrix illustrated in Table I, which reflects exactly the access control rules that were set up at the network controller.

VI. Countermeasure to the KYE Attack

In this section, we propose a countermeasure to the KYE attack that does not require modifications to SDN, but rather that takes advantage of SDN programmability. We call this countermeasure *flow obfuscation*. In the following analysis, we use a stronger attack model than the one presented in Section II. In particular, we assume that the attacker can obtain a flow table side-channel for up to n switches in the target SDN network. Therefore, we consider an attacker who is much stronger that the one previously considered.

A. Flow Obfuscation

As described in our attacker model (see Section II), to perform a KYE attack an attacker needs to be able to identify which flow rules are installed in response to his traffic. Our flow obfuscation countermeasures exploits the ability of OpenFlow switches to modify packets in transit, so that the attacker cannot distinguish flow rules for his traffic flows from flow rules of other users. Figure 7 provides an overview of this countermeasure.

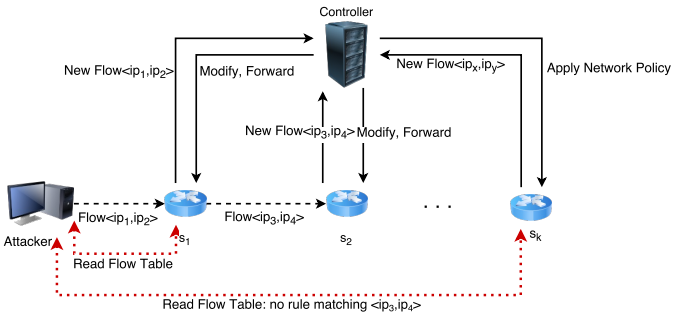


Fig. 7: Overview of the flow obfuscation countermeasure. The dotted lines indicate the flow table side-channel.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

	10.0.0.0/24	10.0.1.0/24	10.0.2.0/24	10.0.3.0/24	10.0.5.0/24	10.0.8.0/24	10.0.10.0/24
10.0.0.0/24	✓	✓	✗	✗	✗	✗	✗
10.0.1.0/24	✓	✓	✗	✗	✗	✗	✗
10.0.2.0/24	✗	✗	✓	✓	✗	✗	✗
10.0.3.0/24	✗	✗	✓	✓	✗	✗	✓
10.0.5.0/24	✗	✗	✗	✗	✓	✓	✗
10.0.8.0/24	✗	✗	✗	✗	✓	✓	✗
10.0.10.0/24	✗	✗	✗	✓	✗	✗	✓

TABLE I: Subnetwork access matrix for the target network, learned by the attacker through a KYE attack. Notation ✓: access allowed; notation ✗: access restricted.

Please note that the use of source and destination IP to identify network flows is just used to present the countermeasure. Indeed, network flows can be matched on arbitrary header fields. Even in such cases, the flow obfuscation countermeasure can be implemented by modifying these fields through *set_field* actions [12].

Any time a new network flow f_i is received by a switch, the controller installs *a single* flow rule on the switch, with two actions: the first action instructs the switch to modify some header fields of the packets of f_i (e.g., source and destination IP), while the second action tells the switch which output port to use for packet forwarding. This process is repeated for the first $k-1$ switches $s_1, ..., s_{k-1}$ in the path, after which, at switch s_k, the controller installs a rule to enforce the appropriate network policy for f_i. We refer to the path comprised of the switches $s_1, ..., s_k$ as the *obfuscation path*. The goal of this countermeasure is to prevent the attacker from learning which network flow causes the installation of a given flow rule; indeed, since the attacker can control up to n switches, when $k \geq n+1$ he will never be able to obtain the complete knowledge required for a successful KYE attack. In fact, when $k \geq n+1$ there are two possible scenarios:

- If the attacker monitors $s_1, ..., s_{k-1}$, then he does not monitor s_k, which is the switch applying the network policy flow rule.
- If the attacker monitors s_k, then he can not know if the installation of a rule on s_k is caused by a network flow he generated. This is because the previous $k-1$ switches modify the packets at each step, and the attacker does not monitor all of them.

It is worth noting that the number (k) of switches on the obfuscation path can be chosen to be lower than the expected number (n) of switch an attacker is able to probe, while still maintain a probabilistic guarantee of security against KYE attacks. Indeed, depending on the average out-degree of the switches in the network, even with a value of $k \leq n$ the probability of the attacker choosing as target the exact k switches used for flow obfuscation can be low, as we show in the next subsection.

Choosing the Value of K. Given an SDN network, let us assume that $o+1$ is the average out-degree of a switch and that an attacker knows the number of switches used in flow obfuscation. If an attacker a can monitor at most n switches (for simplicity, we assume $n \mod k = 0$), the probability $P^s_{s1,sk}$ of him monitoring exactly the k switches $s_1, ..., s_k$ used for flow obfuscation is:

$$P^s_{s1,sk} = \left(\frac{(n/k)}{o}\right)^{k-1},$$

since attacker always knows the identity of s_1 (the edge switch through which the attacker's traffic is routed). A network manager can decide the appropriate value for k based on the expected value of n and the maximum probability $P^s_{s1,sk}$ that he is willing to accept.

B. Evaluation of Flow Obfuscation

In this Section we describe the experimental evaluation of our flow obfuscation countermeasure. For our evaluation, we use the widely used clos network architecture [18], [55]. In particular, we employ the two-tier clos network depicted in Figure 8, with two hosts connected to each aggregation switch for a total of 12 hosts. We simulated this network using the Mininet network simulator and we wrote a module for the POX controller implementing our flow obfuscation countermeasure. In this network, we evaluated the overhead introduced by increasing obfuscation path length.

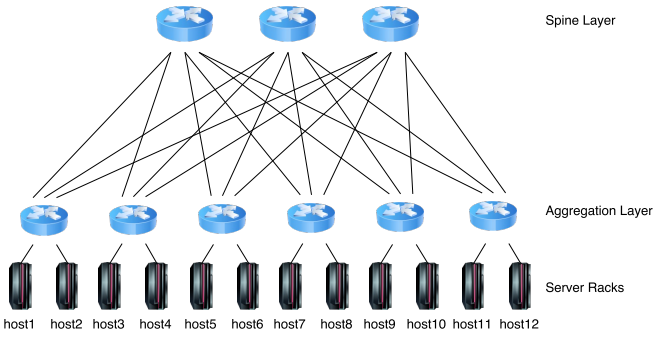


Fig. 8: Two-tier clos network used in the experimental evaluation of the flow obfuscation countermeasure.

In our experiments, we measured the RTT between each pair of hosts in the network varying the length of the obfuscation path between 1 (i.e., no countermeasure) and 6. Specifically, for each host we pinged every other host in the network 10 times and we took the average. Since our network is comprised of 12 hosts, two for each aggregation switch, for each obfuscation path length we have a sample of $10 * 12 * 11 = 1320$ RTTs. Figure 9 illustrates the results of our experimental evaluation. As expected, the average RTT increases as the length of the obfuscation path increases, roughly doubling for $k = 6$. It is interesting to notice how lower lengths of the obfuscation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2018.2806977, IEEE Transactions on Emerging Topics in Computing

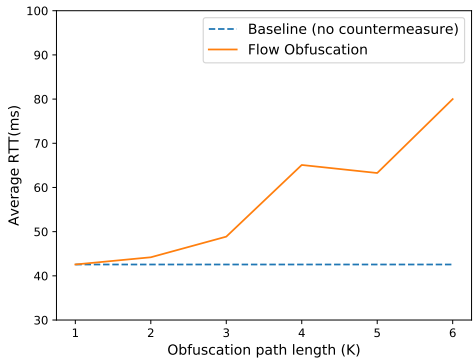


Fig. 9: Average RTT in ms between each pair of hosts in the network, for varying length of the obfuscation path (k).

path increase the RTT between hosts only slightly, while still provide mitigation against the KYE attack. For instance, in this particular network a value of $k = 3$ provides a reasonable tradeoff between increased latency and attack mitigation, with a latency overhead of 15% and resilience against up to two compromised switches. It is worth noting that 3 switches is the average number of switches between each pair of hosts in our network. In general, setting the length of the obfuscation path equal to the average number of switches between any two hosts will provide a good tradeoff between attack prevention and overhead mitigation.

VII. Limitations of the KYE Attack and Future Work

The KYE attack requires constant assessments of the flow table by the attacker and the ability to recognise deviations in the type of flow rules installed. A human attacker can find this process bothersome and non-trivial in certain cases, leading to imprecise identification of detection conditions and/or defense measures applied. Additionally, the attacker might not be able to correctly identify the network policy applied if the policy itself is new or unknown to the community. While this might be the case, we argue that the attacker does not necessarily need to learn the exact nature of the network policy applied, as long as he is able to learn enough information for his purposes. For instance in case of network scanning, not being able to identify the defense mechanism used might be acceptable for the attacker, as long as he is able to understand when the scanning traffic is detected and when it is not. The attacker can learn when the scanning is detected by observing a deviation in the flow rules installed for different types of probing traffic (e.g., very low scan rate probing traffic v.s. fast scan rate probing traffic).

In our future work, we plan on automating the KYE attack by means of machine learning techniques. Indeed, the efficacy of the KYE attack is based on the ability to recognising patterns in the features of the generated traffic and in the flow rules installed in response, as well as to recognise significant deviations from such patterns. We believe that machine learning can be successfully applied to detect the baseline behaviour of the control plane under normal traffic conditions. Once this

TABLE II: Latency overhead

Obfuscation Path Length	RTT (ms)	Ratio
1	42.57	1.0
2	44.2	1.04
3	48.87	1.15
4	65.09	1.53
5	63.27	1.49
6	80.0	1.88

Latency overhead and ratio for different lengths of the obfuscation path.

baseline behaviour has been identified, we can use a classifier to detect and categorize deviations from such behaviour triggered by specific traffic flows. It would then be possible to use another classifier to find the most relevant features of the attack traffic which caused the abnormal network state (e.g., detection of an attack). In this system model, the OpenFlow switch acts as an oracle: the switch can be repeatedly queried to learn if some network flows trigger a deviation in the usual type of flow rules installed. By analyzing the difference between the features of network flows that triggered the abnormal network state versus those which did not, it is possible to learn which are the characteristics that caused the reaction.

Finally, in complex networks it is possible to have several subnetworks, each with separate network policies. In this scenario, the KYE attack allows to learn the network configuration of the subnetwork for which the attacker has a flow table side-channel. However, we argue that it is highly unlikely for subnetworks belonging to the same network to have completely different policies. Consequently, even in case of extremely complex networks, the information that the KYE attack provides about one subnetwork also applies to the other subnetworks, and can be used to mount subsequent attacks.

VIII. Related Work

SDN has become a popular research topic in recent years, especially in relation to security. In [42], Kreutz et al. discuss the effects of an attacker compromising a switch in the network, which can result in traffic injection attacks, man-in-the-middle attacks and traffic filtering. Similarly, in [23] the authors consider an attacker with full control over the switch and discuss several possible attacks like man-in-the-middle, state, and topology spoofing. However, in these works the attacker actively modifies the state of the switch, exposing him to detection by the controller through querying the state of the switch [26], [40] or RTT analysis [23]. In [41], Klöti et al. prove that by analyzing the RTT for a specific network flow, an attacker is able to infer if a rule is already installed for that network flow. While the goal of this attack is to obtain some information on the state of a switch, the KYE attack allows to learn a much greater amount of information about the logic and policies of the network, such as attack detection threshold and defense mechanisms applied. More recently, Sonchack et al. [56] developed a timing attack aimed at disclosing sensitive information about the network. This attack shares the same goal of obtaining knowledge on the behaviour of the network

with our attack, but differs in both how it is achieved and in the detail of information that can be gathered. Another set of works related to ours pertain to SDN fingerprinting [52]. SDN fingerprinting techniques use RTT to infer if a given network is an SDN or a classical network. These techniques apply the observation, made in [41], that in SDN the first packet belonging to a network flow has a higher RTT then subsequent ones. By exploiting this asymmetry, an attacker can successfully infer if a network is an SDN with high accuracy. Fingerprinting attacks differ from our work since the type, the amount, and detail of information retrieved with the KYE attack is much greater and more fine-grained.

IX. Conclusions

In this paper, we proposed a thorough analysis of the vulnerability introduced by the on-demand installation of flow rules in SDN. We presented the novel KYE attack which, with minimal requirements, allows an adversary to gather an extensive amount of information regarding the configuration of the network, ranging from security-related aspects to network engineering policies. We implemented the KYE attack and conducted a thorough evaluation, showing its feasibility and efficacy against a popular scanning detection algorithm and against standard access control policies. Finally, we proposed and experimentally evaluated the flow obfuscation countermeasure. We showed how the countermeasure provides provable security guarantees and can be configured to adapt to the needs of a specific network under consideration.

References

[1] http://blog.trendmicro.com/trendlabs-security-intelligence/netis-routers-leave-wide-open-backdoor/. Accessed: 06-2017.

[2] Algosec: The state of automation in security survey report. https://www.algosec.com/wp-content/uploads/2016/03/The-State-of-Automation-in-Security-Survey-Final.pdf. Accessed: 08-2017.

[3] Cisco confirms two of the shadow brokers' 'nsa' vulns are real. https://www.theregister.co.uk/2016/08/17/cisco_two_shadow_brokers_vulnerabilities_real/. Accessed: 06-2017.

[4] Dpctl documentation. https://github.com/CPqD/ofsoftswitch13/wiki/Dpctl-Documentation. Accessed: 06-2017.

[5] The eight security backdoors that helped kill faith in security. http://www.computerworlduk.com/security/security-backdoors-that-heped-kill-faith-in-security-3634220/. Accessed: 06-2017.

[6] Gartner: One brand of firewall is a best practice for most enterprises. https://www.gartner.com/doc/3215918/brand-firewall-best-practice-enterprises. Accessed: 08-2017.

[7] Hp procurve 5400 openflow switch. http://archive.openflow.org/wk/index.php/Configuring_HP_Procurve. Accessed: 08-2016.

[8] Juniper screenos authentication backdoor. https://community.rapid7.com/community/infosec/blog/2015/12/20/cve-2015-7755-juniper-screenos-authentication-backdoor. Accessed: 06-2017.

[9] Microsoft's DEMON. https://sharkfest.wireshark.org/sharkfest.12/presentations/A-4_Leveraging_Openflow_to_create_a_Large_Scale_and_Cost_Effective_Packet_Capture_Network.pdf. Accessed: 06-2017.

[10] Mininet network simulator. http://mininet.org/. Accessed: 06-2017.

[11] Neustar annual ddos attacks and impact report. https://www.neustar.biz/resources/whitepapers/ddos-protection/2014-annual-ddos-attacks-and-impact-report.pdf. Accessed: 06-2017.

[12] Openflow specification. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf. Accessed: 06-2017.

[13] Pox network controller. https://github.com/noxrepo/pox. Accessed: 06-2017.

[14] Snowden: The nsa planted backdoors in cisco products. http://www.infoworld.com/article/2608141/internet-privacy/snowden--the-nsa-planted-backdoors-in-cisco-products.html. Accessed: 06-2017.

[15] Synful knock - a cisco router implant - part i. https://www.fireeye.com/blog/threat-research/2015/09/synful_knock_-_acis.html. Accessed: 06-2017.

[16] US DDoS attacks and protection report. https://www.neustar.biz/ddos-attacks-report. Accessed: 06-2017.

[17] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov. Security in software defined networks: A survey. *IEEE Communications Surveys Tutorials*, 2015.

[18] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, 2008.

[19] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, 2010.

[20] E. S. Al-Shaer and H. H. Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 2004.

[21] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran. Lineswitch: Efficiently managing switch flow in software-defined networking while effectively tackling dos attacks. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '15, 2015.

[22] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran. Lineswitch: Tackling control plane saturation attacks in software-defined networking. *(IEEE/ACM) Transactions on Networking*, in press, 2016.

[23] M. Antikainen, T. Aura, and M. Särelä. Spook in your network: Attacking an sdn with a compromised openflow switch. In *Secure IT Systems: 19th Nordic Conference, NordSec 2014*, 2014.

[24] A. B. Ashfaq, M. J. Robert, A. Mumtaz, M. Q. Ali, A. Sajjad, and S. A. Khayam. A comparative evaluation of anomaly detectors under portscan attacks. In *Proceedints of the 11th International Symposium on Recent Advances in Intrusion Detection*, RAID, 2008.

[25] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks*, 2015.

[26] K. Benton, L. J. Camp, and C. Small. Openflow vulnerability assessment. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, 2013.

[27] R. Braga, E. Mota, and A. Passito. Lightweight ddos flooding attack detection using nox/openflow. In *Proceedings of the 35th IEEE Conference on Local Computer Networks*, LCN, 2010.

[28] Y. Changhoon and L. Seungsoo. Attacking sdn infrastructure: Are we eady for the next-gen networking? BlackHat 2016, 2016.

[29] C. J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang. Nice: Network intrusion detection and countermeasure selection in virtual network systems. *IEEE Transactions on Dependable and Secure Computing*, 2013.

[30] M. Conti, F. De Gaspari, and L. V. Mancini. Know your enemy: Stealth configuration-information gathering in sdn. In *Proceedings of the 12th International Conference on Green, Pervasive, and Cloud Computing*, GPC, 2017.

[31] H. Cui, G. O. Karame, F. Klaedtke, and R. Bifulco. On the fingerprinting of software-defined networks. *IEEE Transactions on Information Forensics and Security*, 2016.

[32] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM Conference*, SIGCOMM '11, 2011.

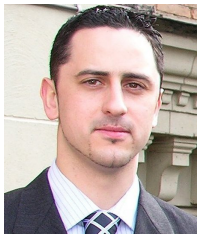
[33] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. Sphinx: Detecting security attacks in software-defined networks. In *Proceedings of the 2015 Network and Distributed System Security Symposium*, NDSS 2015.

[34] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *IEEE Internet Computing*, 2013.

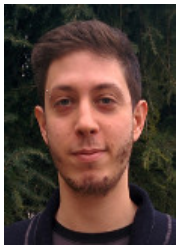
[35] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp. Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In *Proceedings of the Signal Information Processing Association Annual Summit and Conference*, APSIPA ASC, 2012.

[36] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar. Scalable video streaming over openflow networks: An optimization framework for qos routing. In *Proceedings of the 18th IEEE International Conference on Image Processing*, 2011.

[37] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments. *Computer Networks*, 2016.
[38] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao. Flowguard: Building robust firewalls for software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, 2014.
[39] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 2013.
[40] A. Kamisiński and C. Fung. Flowmon: Detecting malicious switches in software-defined networks. In *Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense*, SafeConfig '15, 2015.
[41] R. Klöti, V. Kotronis, and P. Smith. Openflow: A security analysis. In *Proceedings of the 21st IEEE International Conference on Network Protocols*, ICNP, 2013.
[42] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, 2013.
[43] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang. Dfence: Transparent network-based denial of service mitigation. In *Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation*, NSDI'07, 2007.
[44] S. Manuel, G. Nicholas, T.-G. Phuoc, and Z. Thomas. Flowfuzz - a framework for fuzzing openflow-enabled software and hardware switches. BlackHat 2017, 2017.
[45] S. A. Mehdi, J. Khalid, and S. A. Khayam. Revisiting traffic anomaly detection using software defined networking. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection*, RAID, 2011.
[46] R. Mohammadi, R. Javidan, and M. Conti. Slicots: An sdn-based lightweight countermeasure for tcp syn flooding attacks. *IEEE Transactions on Network and Service Management*, 2017.
[47] G. Pickett. Abusing software defined networks. BlackHat 2014, 2014.
[48] S. E. Schechter, J. Jung, and A. W. Berger. Fast detection of scanning worm infections. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection*, RAID, 2004.
[49] S. Scott-Hayward, S. Natarajan, and S. Sezer. A survey of security in software defined networks. *IEEE Communications Surveys Tutorials*, 2016.
[50] L. Seungsoo, K. Jinwoo, and S. Seungwon. Delta: Sdn security evaluation framework. BlackHat 2017, 2017.
[51] S. Shin and G. Gu. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *Proceedings of the 20th IEEE International Conference on Network Protocols*, ICNP, 2012.
[52] S. Shin and G. Gu. Attacking software-defined networks: A first feasibility study. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, 2013.
[53] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium*, NDSS 2013.
[54] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, 2013.
[55] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, 2015.
[56] J. Sonchack et al. Timing-based reconnaissance and defense in software-defined networks. In *Annual Conference on Computer Security Applications*, 2016.
[57] M. Suh, S. H. Park, B. Lee, and S. Yang. Building firewall over the software-defined network controller. In *Proceedings of the 16th International Conference on Advanced Communication Technology*, 2014.
[58] J. Twycross and M. M. Williamson. Implementing and testing a virus throttle. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, 2003.
[59] J. Wang, Y. Wang, H. Hu, Q. Sun, H. Shi, and L. Zeng. Towards a security-enhanced firewall application for openflow networks. In *Proceedings of the 5th International Symposium on Cyberspace Safety and Security*, CSS'13, 2013.



Mauro Conti Mauro Conti is an Associate Professor at the University of Padua, Italy. He obtained his Ph.D. from Sapienza University of Rome, Italy, in 2009. After his Ph.D., he was a PostDoc Researcher at Vrije Universiteit Amsterdam, The Netherlands. He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His main research interest is in the area of security and privacy. In this area, he published more than 150 papers in topmost international peerreviewed journals and conference. He is Associate Editor for several journals, including IEEE Communications Surveys & Tutorials and IEEE Transactions on Information Forensics and Security. He was Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, and General Chair for SecureComm 2012 and ACM SACMAT 2013. He is Senior Member of the IEEE.



Fabio De Gaspari Fabio De Gaspari is a PhD student in Computer Science at Sapienza University of Rome, Italy. His research areas are network security, new network architectures and active defence techniques. Currently, he is working on the security aspects of Software-Defined Networking. Fabio obtained his MSc in Computer Science in 2015 from the University of Padova, Italy.

Luigi V. Mancini Luigi Vincenzo Mancini received the Ph.D. degree in computer science from the University of Newcastle, U.K., in 1989. He is currently a Full Professor with the University of Rome La Sapienza, where he has been the Vice Dean of the Faculty of Ingegneria dell'Informazione, Informatica e Statistica since 2013. He has authored over 100 scientific papers in international conferences and journals, and has received more than 4000 citations. His current research interests include cloud computing security, network and information security, and computer privacy. He has served on the program committees of several international conferences, among which are the ACM Conference on Computer and Communication Security, the ACM Symposium on Access Control Models and Technology, the European Symposium on Research in Computer Security, and the Financial Cryptography and Data Security Conference. He is the Founder of the two master's degree programs in information and network security with the University of Rome La Sapienza, and the Laboratory of Information and Communication Security. He participated in numerous national and international research projects in the area of security and privacy, and in particular, he is the Technical Leader of the SUNFISH project funded by the EC Horizon 2020 research and innovation program..