Watch your step! Detecting stepping stones in programmable networks

Debopam Bhattacherjee[†]
Dept. of Computer Science
ETH Zürich
Zürich, Switzerland
debopam.bhattacherjee@inf.ethz.ch

Andrei Gurtov
Dept. of Computer and Information Science
Linköping University
Linköping, Sweden
gurtov@acm.org

Tuomas Aura
Dept. of Computer Science
Aalto University
Espoo, Finland
tuomas.aura@aalto.fi

***Abstract*—Hackers hide behind compromised intermediate hosts and pose advanced persistent threats (APTs). The compromised hosts are used as stepping stones to launch real attacks, as is evident from an incident that shook the world in 2016 — Panama Papers Leak. The major attack would not go unnoticed if the compromised stepping stone, in this case an email server, could be identified in time. In this paper, we explore how today's programmable networks could be retrofitted with effective stepping stone detection mechanisms to correlate flows. We share initial results to prove that such a setup exists. Lastly, we analyze scalability issues associated with the setup and explore recent developments in network monitoring which have potential to address these issues.**

***Index Terms*—Advanced persistent threat, Stepping stone attack, Network monitoring, Intrusion detection, Programmable network, Software-defined network**

I. Introduction

Since time immemorial kings have leveraged stepping stones to conquer much coveted unbequeathed land. They would besiege and conquer forts and kingdoms which would strategically give them an advantage while attacking the actual target. A hacker, who remains anonymous till date, stole tens of millions of classified documents from a Panamanian law firm, named Mossack Fonseca, in 2015 [1] using a similar technique and exposed individuals and companies across the world engaged in fraudulent and tax-evading financial activities. According to popular speculation, the attack was carried out by compromising an email server and using it as a `stepping stone`. So, as is evident, the perception of war has changed over time, but basic techniques remain the same.

In a stepping stone attack (Fig. 1), the hacker compromises one vulnerable host after other before reaching the target host possibly holding sensitive information. This also anonymizes the hacker who effectively hides himself behind a long chain of compromised stepping stones. Stepping stone attack is a classic example of advanced persistent threat (APT). A network provider or administrator hardens the most critical resources. So the strategy to directly attack these critical resources does not always work. But a locked front door does not necessarily mean that the cellar is also properly closed. A file server behind the firewall can be very hard to access or compromise, but an email server hosted in the demilitarized zone (DMZ) might be somewhat easier to get into. Once compromised, this host can reveal sensitive information sufficient for further exploration. These APTs are often very slow and concealing. The benefit often lies not in taking down a host or crafting denial of service but in stealth. 11.5 million documents, known as Panama Papers, were stolen using such a persistent attack by stealthy hackers. As humans are often in the loop, these threats are often advanced and relies on intuition.

Stepping stone attack is an age-old technique and so are the detection mechanisms. There exists a rich literature on proposals for stepping stone detection (SSD) and we discuss some of those in §II. We, however, do not claim that the discussion on related work is exhaustive; but it provides sufficient insight on the depth and breadth of the field. In this paper, we propose a monitoring system targeted at stepping stone detection for programmable networks [2]. The proposed system leverages sFlow [3] as an interface between the data plane and the monitoring and detection plane.

We argue that a simple network monitoring strategy in large programmable networks can fall short in terms of scalability. Exhaustive monitoring adds large overhead while lower sampling or mirroring rates might miss critical information and insight. Hence, we select some of the recently proposed network monitoring and debugging tools and analyze how these tools can be used for SSD. Monitoring tools, which we discuss in §VI in further details, often leverage endpoint hypervisors or flow tables in data plane nodes. These resources can be leveraged to have the initial unproven `hunch` that a pair of flows could be correlated. The monitoring system is

[†]Work done when affiliated to Aalto University, Finland.

978-1-5386-8088-9/19/$31.00 ©2019 IEEE

Fig. 1: *A typical stepping stone attack, where the hacker or attacker hides himself behind N sequential stepping stones.*

then notified, which reacts by enabling targeted exhaustive analysis of the flow pair and/or chain.

Our main contributions include:

- Proposing a monitoring system that can be integrated easily with programmable networks in order to detect stepping stones by correlating flows.
- Presenting initial results which raise scalability concerns.
- Identifying some interesting paths forward, leveraging endpoint hypervisors or data plane flow tables, taking into concern recent programmable network debugging and monitoring advances.

II. Stepping stone detection mechanisms

Early detection techniques relied on content-based detection. For example, Staniford Chen and Heberlein proposed a thumbprint-based detection mechanism [4]. Thumbprints are short summaries of content over a certain period in a flow, which are matched to identify correlations. Random bit errors which are common in real networks lead to a thumbprint value similar to the actual one in contrast to the avalanche-effect seen in cryptographic hashing. Content-based solutions are limited to un-encrypted traffic and hence are ineffective nowadays.

Various transmission characteristic-based detection techniques exist [5]. Wang et al. propose a sleepy watermark tracing [6] technique which relies on injecting watermarks into flows and coordinating with routers in the path. This approach also does not work on encrypted traffic. Yang and Huang propose the use of temporal thumbprints or T-thumbprints [7], which are based on the temporal spacing of packets in a flow, to correlate different flows in a stepping stone chain. This approach works on encrypted interactive traffic.

The timing-based detection technique [8] proposed by Zhang and Paxson also works effectively on encrypted interactive traffic. Their solution is based on the fact that human keystrokes follow a Pareto distribution in an interactive terminal. Hence the network traffic can be divided into ON (data) and OFF (no data) periods. An OFF period starts if there is no data for time T_{idle}. The OFF period ends and ON period starts when the first data packet is next observed. ON periods of two different flows are correlated if the starting times of the ON periods differ by a value $\leq \delta$ seconds. The sink or destination of the first flow also needs to be the source of the next flow. Two flows are correlated if

$$\frac{ON_{1,2}}{min(ON_1, ON_2)} \geq \gamma$$

where $ON_{1,2}$ denotes the number of correlated ON period starts, ON_1 denotes the number of ON periods of the first flow, ON_2 denotes the number of ON periods of the second flow, and γ is a control parameter. The false positive rate is reduced by incorporating causality constraint in the solution.

In this paper, we adapt the timing-based detection mechanism discussed above to today's programmable networks. In the following sections we discuss programmable networks, our experimental setup and initial results. We include discussions on the yet unresolved challenge of scalability, as seen by timing-based detection mechanism, and analyze the solution space based on recent advances in network monitoring.

III. Background on programmable networks

Programmable networks or software-defined networks (SDNs) [9] mark the separation of the control plane and data plane in order to make the network nodes programmable. the network architecture has 3 tiers: the application tier, the control tier, and the infrastructure tier. The control tier consists of the logically centralized controller, which provides a network-wide view of the forwarding elements and their states to the application tier via north-bound interfaces (NBI). Distributed routing protocols are replaced in SDN by algorithms that make use of the global view of the network. The centralized control plane is the single point of configuration for the network administrators. The controller in turn manages the forwarding elements.

The infrastructure tier (data plane) consists of forwarding elements which typically forward packets based on layer-2 and layer-3 headers and are known as switches. The controller communicates with the switches using south-bound interfaces (SBI). The SBI is used by the controller to send instructions to the switches and by the switches to consult the controller when they are not able to make the forwarding decision based on the previous instructions.

sFlow It is a traffic monitoring technique [3] in networks. Low cost sFlow agents are installed in the switches which sample packets and forward sampled data to a data collector for analysis. sFlow defines the

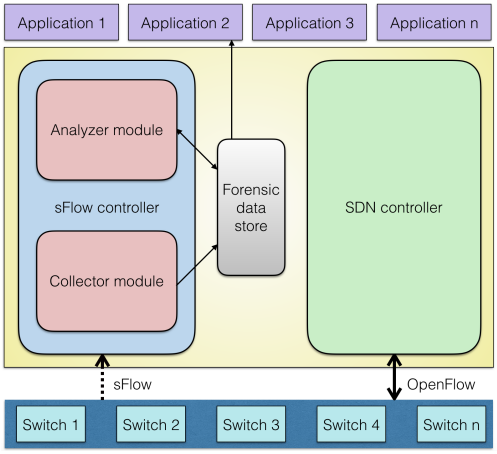

Fig. 2: *A monitoring architecture for programmable networks.*

sampling techniques used in the sFlow agents, the sFlow management information base (MIB) used by the sFlow collector (analyzer) to control the sFlow agents and the format of the data forwarded by the sFlow agents to the collector. Cisco's NetFlow [10] or IPFIX [11] serve similar purpose, albeit at a different granularity. Flow statistics are aggregated at the network nodes and forwarded periodically to the controller. The fact, that sFlow is dumb and blindly forwards packet headers in real-time to the collector for further analyses, has 2-fold benefit — a) it reduces performance overhead in the switches, and b) analysis at the centralized controller can be done in real-time. For our experiments, we select sFlow as the preferred technique for packet monitoring.

OpenFlow [12] and sFlow are complementary technologies. OpenFlow is an SBI for SDN environments which configures the switches by translating user requirements into instructions and installing flow entries into the flow tables of the switches. sFlow, on the other hand, provides an API for network monitoring and opens up the possibility of performance aware network management and provisioning.

IV. Proposed System Design

In our proposed architecture, as shown in Fig. 2, switches interact with 2 controllers — a) a standard SDN controller using OpenFlow as the interface protocol, and b) a monitoring controller we devised that collects and analyzes network monitoring data. The devised controller has 2 modules — a) a collector module which collects monitoring data from the switches (sFlow) and stores collected data in a forensic data store, and b) an analyzer module which retrieves monitoring data from the data store and analyzes data to find out interesting correlation information.

We prefer sFlow to NetFlow/IPFIX as the switches immediately forward the sampled header information without caching. The sFlow sampling rate and polling rate can be configured for the switches. These rates can be adjusted to fit the type of the network (bandwidth, traffic volume, etc.) so that the sFlow collector is not flooded with sFlow datagrams. As different sampling rates are used, the probability of a packet being sampled and reported to the collector varies. For example, if the sampling rate is 1 in s packets, then the probability of a particular packet getting sampled at 1 switch is $1/s$. Hence, if packets have to pass through n switches on average, the probability of a packet being sampled in at least one of the n switches is $1 - ((s-1)/s)^n$ which increases with n for a fixed value of s. On the other hand, for a fixed average number of switches n, the probability of a packet being sampled in at least one of the switches decreases as s increases.

The sFlow agents embedded in switches can be configured to modify the sampling and polling rates. The sFlow agent extracts header information from the sampled packets, marshals the header information into sFlow datagrams, and sends the datagrams immediately to the sFlow collector. The sFlow collector module extracts the header information from the sFlow datagrams and stores the information in a data-store to be used by the data-analysis module in order to correlate flows and detect stepping-stones.

The forensic data store stores monitoring data collected by the collector module for forensic analysis. Although the header information received by the collector module gets transiently stored here, the data becomes stale immediately after being analyzed by the analyzer module and hence can be deleted. The forensic data store only persists interesting correlation information, which can act as evidence in forensic analysis.

The data-analysis module analyzes the monitoring data collected by the data-collection module. It fetches collected data and removes any redundancy introduced by multiple sFlow agents in the path of a packet sending the same header information. Then it matches headers to flows. Once the matching has been done, the header information is used to update the flow data. The individual flow-level information is used to correlate flows. The knowledge gained by this module can be used by an intrusion prevention system (IPS) application, which may instruct the SDN controller to quarantine stepping stones and to restrict traffic generated at these stepping stones.

In our study, we have used Mininet 2.2.1 [13], Open vSwitch 2.0.2 [14], OpenFlow 1.0 [12] and sFlow 5.0 [3]. For the forensic data store we have used MongoDB 3.2.4 [15]. Mininet includes a reference controller which installs flow entries to the flow tables of the switches through the SBI. We require a framework to monitor the network in order to identify the stepping

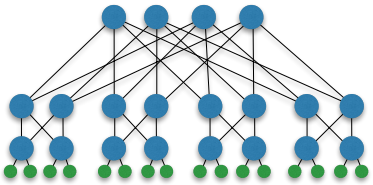

Fig. 3: *A 4-ary fat-tree topology. Large circles represent switches while smaller ones are end hosts.*

stones as OpenFlow counter values do not provide sufficient information. Hence, we implement an sFlow controller to gather traffic-related metadata from the switches (sFlow agents), and the forensic data store.

We consider four different topologies in our experiments — single switch or star, linear, tree, and clos (4-ary fat-tree topology). The average number of switches between two end hosts varies between the topologies. If the sFlow packet sampling rate is set to 1 in s packets and the average number of switches between two end hosts is n, then the probability of a packet getting sampled in at least one of the switches is $1-((s-1)/s)^n$. In this paper, however, we only discuss results for the 4-ary fat-tree topology, as shown in Fig. 3, as it is the most realistic one and is extensively studied in the context of data centers. There are 5 switches between any 2 end hosts in different pods in a 4-ary fat-tree. Hence, the probability of a packet, sent along such an inter-pod path, getting sampled is $1-((s-1)/s)^5$ when the packet sampling rate is set to 1 in s packets.

sFlow security The deployment of network monitoring raises a number of security issues which need to be addressed. sFlow does not have any security mechanism and relies on proper deployment and configuration. sFlow traffic is sent unencrypted to the collector. Casual eavesdropping as well as spoofing of datagrams are possible. To eliminate these issues, sFlow datagrams should be sent through an isolated channel. VLAN or VPN tunnels can be used to create these secure isolated channels. The solution is deployment specific and in our experiments we have simply forwarded the unencrypted traffic through the network to the collector. However, in our implementation, we check the sequence numbers of the headers encapsulated in the sFlow datagrams to remove possible redundancy or spoofed packets. Analysis of the sFlow datagrams can reveal sensitive information about the network activities of a user. Although sampling of packets at the switches and limiting the number of header bytes encapsulated by the sFlow datagram prevents leakage of sensitive information to some extent, only the network administrators with proper rights should be allowed to access the forensic data store for forensic analysis. Nevertheless, network monitoring itself makes the network more robust and less vulnerable to attacks.

V. Initial Results

We generated interactive attack traffic due to lack of real data consisting of ON/OFF periods. Volunteers (Computer Science students in their 20s) were asked to type in a selected set of unix commands via nested `ssh` terminals in order to generate attack traffic. This attack traffic was mixed with normal traffic consisting of various Web requests and response. Each experiment, with varying packet sampling rates, is run 100 times in order to capture the non-deterministic behaviour at the switches (different packets are sampled across runs) and we let the collector and analyzer modules correlate flows. The interactive attack traffic uses 2 consecutive stepping stones in 2 different pods in a 4-ary fat-tree. Hence, each packet traverses exactly 5 switches while traveling through the network due to shortest-path routing.

A. Identification of ON Periods

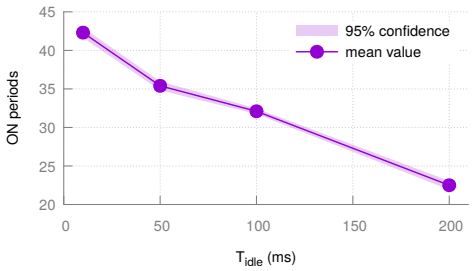

Fig. 4: *Effect of varying T_{idle} on the identification of ON periods. Packet sampling rate is 1 in 10.*

Fig. 4 shows that the number of ON periods decreases as T_{idle} (check §II for definition) increases for a fixed packet sampling rate of 1 in 10. This is along the lines of expectation, as human typing patterns in interactive terminals follow Pareto distribution [8]. For the same traffic trace, 15 ON periods were reported when we set the sFlow packet sampling rate to 1 in 1 and T_{idle} to 200 ms (plot omitted). The increase in the number of ON periods when sampling rate is 1 in 10 is due to the fact that some of the packets within a longer ON period were not sampled which split it to multiple shorter ON periods. Note that a sampling rate of 1 in 10 does not necessarily capture only 1 in every 10 packets. Different switches along the path of a flow might capture and report different packets thus making the monitoring system more fine-grained than it seems.

B. Correlation of ON Periods

The monitoring system correlates ON periods of two consecutive flows in a chain of stepping stones. As the malicious traffic hops from one pod to the other, the system correlates ON periods of successive flows.

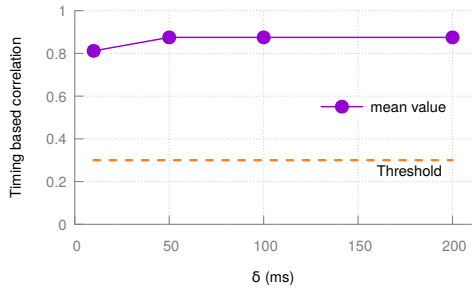

Fig. 5: *Effect of varying δ on the timing based correlation score of consecutive flows in a stepping stone chain. Packet sampling rate is* 1 *in* 1.

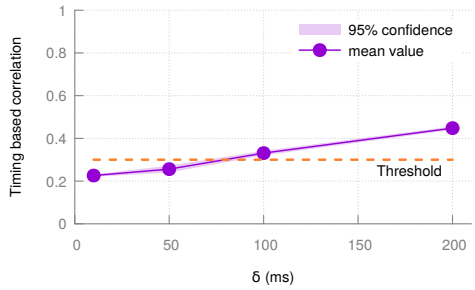

Fig. 6: *Effect of varying δ on the timing based correlation score of consecutive flows in a stepping stone chain. Packet sampling rate is* 1 *in* 10.

ON periods of two different flows are correlated if the starting times of the ON periods differ by a value $\leq \delta$ seconds. When every packet is monitored, most of the correlated ON periods could be identified even for low values of δ ($<$ 10 ms). When packet sampling rate is 1 in 10, the number of correlated ON periods are less. Nevertheless, the number of correctly identified correlations increase with δ and is sufficiently large for values higher than 100 ms.

C. Timing-Based Correlation

Fig. 5 and 6 show the impact of selected δ and packet sampling rates on the timing-based correlation scores of actually correlated flows. Recall from §II that *timing-based correlation score* of two connections is given by

$$\frac{ON_{1,2}}{min(ON_1, ON_2)}$$

where $ON_{1,2}$ denotes the number of correlated ON period starts, ON_1 denotes the number of ON periods of the first flow, and ON_2 denotes the number of ON periods of the second flow. The threshold for marking pairs of flows as correlated, γ_{timing}, is set to 0.3 following the proposal by Zhang *et al.* [8]. When every packet is monitored, consecutive flows have very high correlation scores even for low values of δ in the range of 10–100 ms. For δ value of 50 ms, the correlation score saturates with a high value of 0.875.

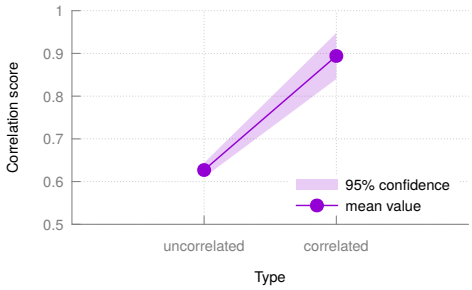

Fig. 7: *Content-size based correlation score for correlated and uncorrelated flow pairs. Note that the y-axis starts at* 0.5.

When packet sampling rate is 1 in 10, effective timing-based correlation is possible for δ values of 100 ms or higher. For lower values of δ, timing-based correlation score falls below the threshold (γ_{timing}) value of 0.3. Although timing-based correlation scores increase with δ, higher values of δ might increase the false positive rate and hence should be avoided.

D. Content-Size Based Correlation

Content-size based correlation is an additional step to reduce the false positive rate. Even if timing-based correlation score for two flows is higher than the threshold, the monitoring system investigates whether an increase in content-size from one correlated ON period to the next of one flow is followed by an increase in the content-size of the consecutive flow, and vice versa. For this purpose, we only consider those pairs of ON periods which are correlated. The score is given by

$$\frac{\textit{Matches found}}{\textit{No. of correlated ON period pairs} - 1}$$

Fig. 7 shows that the scores differ significantly in the two cases. When the flow pairs are uncorrelated, the average content-size based correlation score is 0.627 ± 0.017 in the 95% confidence interval. When the flow pairs are correlated, the average score is 0.894 ± 0.054 in the 95% confidence interval. We conjecture 0.8 as the content-size based correlation threshold – $\gamma_{content-size}$. Two flows are marked as correlated if and only if both the timing-based correlation score and the content-size based correlation score lie above the threshold values.

Legitimate stepping stones Our setup and results do not differentiate between legitimate and illegitimate stepping stones. The network, ideally, should not raise alarm when a legitimate stepping stone is used. A user, for example, might use an intermediate proxy to open an ssh connection to a server. The flow from the user to the proxy is correlated to the flow from the proxy to the server; but, the traffic is not malicious. Our setup identifies correlated flows. Mapping a flow to an attack is completely orthogonal to this study. Also, identifying stepping stones, both legitimate and illegitimate, is key

to detection of stepping stone attacks. Identification leads to quarantine followed by further investigation.

Lack of real data Our experiments and results lack analyses of real network traffic. We could not find network traces which can be interesting to analyze. Facebook recently published an anonymized data set [16] containing sampled network traffic from 3 production clusters (database, Web server and Hadoop) in their Altoona Data Center. Nevertheless, the sampling rate of 1 in every 30,000 packets renders the whole data set uninteresting to analyze in this context.

Scalability concerns In §V-B and §V-C we hint on the scalability issues with this stepping stone monitoring system. As this detection mechanism relies heavily on identifying ON/OFF patterns, sampling packets for monitoring might fail to identify this pattern correctly. As less and less packets are sampled, the accuracy and effectiveness of this mechanism diminishes very fast. As is evident from Fig. 6, for a sampling rate of 1 in 10, this mechanism is unable to correlate consecutive flows in a stepping stone for low values of δ. In real networks, more exhaustive sampling leads to undesirable overheads. A significant fraction of resources, both compute and bandwidth, are consumed for monitoring the network thus reducing the capability to serve real traffic. Sampling rates of 1 in every 100, 1,000 or even lower are expected to be the norm. So, where do we go from here? Is this an unavoidable deadlock? Can we leverage recently proposed network debugging and monitoring techniques to propose lightweight yet effective solutions? In the following section, we discuss how these recent tools can be used to detect stepping stones.

VI. Recent monitoring techniques in programmable networks

Various network monitoring and debugging techniques, both at network nodes and end-hosts, aimed to solve scalability issues have been recently proposed. Our study of SSD is incomplete without discussing some of these techniques and analyzing how stepping stones can be detected in SDN using these newly proposed tools.

Trumpet This monitoring technique [17] pushes the per-packet monitoring scalability issues to end-host hypervisor leveraging the powerful CPUs at end-hosts. User defined events are translated to triggers at a logically centralized controller. This controller then pushes the triggers to end-hosts. The triggers test for conditions being met and appropriate signals are then forwarded to the controller. The controller aggregates these signals to identify network-wide events. At the end-host, Trumpet works in 2 distinct phases — a) a `match phase` when each packet is matched to flow based on 5-tuple and per flow statistics is maintained, and b) a `gather and report phase` where statistics corresponding to each trigger is gathered and reports are forwarded to the central controller when triggers are satisfied. Trumpet has provisions for NIC-offloading [18] in order to save end-host CPU cycles. NIC-offloading also helps monitor flows which bypass end-host processing (for example, RDMA [19]). There is also provision for mirroring packets to the controller when specific events are triggered.

Trumpet can be used to effectively detect stepping stones at scale. In cloud data centers, for example, hypervisors can be used to correlate flows coming to and going from a specific VM based on the ON/OFF patterns. Even if the VM is compromised, this does not restrict the hypervisor from reporting possible stepping stone to the centralized controller. The controller can put reported flows in quarantine and capture every packet corresponding to those flows in order to correlate multiple stepping stones in chain to detect intrusion. This proactive reporting by the hypervisor followed by the reactive exhaustive packet capturing by the switches in the network might be scalable in large networks. In non-VM environments, Intel SGX [20] can be used to monitor flows at the end-hosts in order to deal with compromised higher privilege applications.

PathDump This tool [21] combines in-network and end-host monitoring and debugging techniques. While the network nodes attach node IDs to packet headers so that end-hosts can reconstruct the path taken by individual packets, the end-hosts aggregate information per 5-tuple flow which includes number of packets and bytes and forward the information to aggregators. On observing events of interest, end-hosts raise alarms. On receiving an alarm, the aggregator can issue further commands to the end-hosts and programmable network nodes. Rules can be set so that end-hosts raise alarms on identifying 2 consecutive flows of a stepping stone chain by observing and correlating the rates of transfer of bytes and packets per flow. A possible correlation raises an alarm and allows the aggregator to enable exhaustive monitoring of packets at specific network nodes in the paths of the suspicious flows.

EverFlow Microsoft uses Everflow [22] to monitor network traffic and debug its network in realtime. It relies on mirroring at the switches which let collector modules gather useful packet-level information to be analyzed either online or offline. This framework is similar to our proposed monitoring system, as shown in Fig. 2, although it does not use sFlow but rely on mirroring techniques. Most of this paper implicitly assumes single-tenant programmable networks, while Microsoft, in practice, deals with multiple tenants using the same

physical network. Hence, they have recently come up with Azure Network Watcher [23] which lets the customers look deeper into the virtual tenant network. The customers can proactively or reactively monitor their network like they could do for on-premise network. They can also detect anomalies and take reactive actions without involving the cloud network provider, Microsoft in this case, in the loop. These monitoring techniques give more power to the hands of customers in multi-tenant networks, who can have their own stepping stone detection mechanism in place without involving the provider, of course leveraging the tools and techniques exposed by the provider.

Programmable data plane Recent advances in programmable data plane, like Tofino (programmable switch ASIC) [24] coupled with P4 [25], open up the possibility to gather telemetry data with minimal overhead. It is possible to answer, without generating additional packets, interesting questions like a) which path did a packet follow, or b) when did a packet arrive at a forwarding element? We keep the analysis of SSD in this context to future work.

VII. Conclusion

Stepping stone attack is an example of advanced persistent threat and is possibly behind the recent Panama Papers leak that shook the world in 2016. We propose a monitoring system for programmable networks in order to detect stepping stone attacks. We identify scalability issues and hence explore recent advances in network monitoring and debugging to address the same. These explorations aim to bridge the gap between network and network security research in the context of stepping stone detection.

VIII. Acknowledgement

A. Gurtov was supported by the Center for Industrial Information Technology (CENIIT), project 17.01.

References

[1] ICIJ, "The Panama Papers: Exposing the Rogue Offshore Finance Industry," https://www.icij.org/investigations/panama-papers/, [Online; accessed 28-July-2018].

[2] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[3] P. Phaal, S. Panchen, and N. McKee, "InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks," Tech. Rep., 2001.

[4] S. Staniford-Chen and L. T. Heberlein, "Holding intruders accountable on the internet," in *Security and Privacy, 1995. Proceedings., 1995 IEEE Symposium on*. IEEE, 1995, pp. 39–49.

[5] R. Kumar and B. Gupta, "Stepping stone detection techniques: Classification and state-of-the-art," in *Proceedings of the international conference on recent cognizance in wireless communication & image processing*. Springer, 2016, pp. 523–533.

[6] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill, "Sleepy watermark tracing: An active network-based intrusion response framework," in *IFIP International Information Security Conference*. Springer, 2001, pp. 369–384.

[7] J. Yang and S.-H. S. Huang, "Correlating temporal thumbprint for tracing intruders," in *Proceedings of 3rd International Conference on Computing, Communications and Control Technologies*, 2005, pp. 236–241.

[8] Y. Zhang and V. Paxson, "Detecting stepping stones." in *USENIX Security Symposium*, vol. 171, 2000, p. 184.

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

[10] M. Fullmer and S. Romig, "The OSU flowtools package and CISCO NetFlow logs," in *Proceedings of the 2000 USENIX LISA Conference*, 2000.

[11] B. Claise, "Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information," Tech. Rep., 2008.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[13] R. L. S. De Oliveira, A. A. Shinoda, C. M. Schweitzer, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*. IEEE, 2014, pp. 1–6.

[14] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The Design and Implementation of Open vSwitch." in *NSDI*, vol. 15, 2015, pp. 117–130.

[15] K. Banker, *MongoDB in action*. Manning Publications Co., 2011.

[16] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the Social Network's (Datacenter) Network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 123–137. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787472

[17] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Trumpet: Timely and precise triggers in data centers," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 129–143.

[18] J. C. Mogul, "TCP Offload Is a Dumb Idea Whose Time Has Come." in *HotOS*, 2003, pp. 25–30.

[19] J. Liu, J. Wu, and D. K. Panda, "High performance RDMA-based MPI implementation over InfiniBand," *International Journal of Parallel Programming*, vol. 32, no. 3, pp. 167–198, 2004.

[20] V. Costan and S. Devadas, "Intel SGX Explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[21] P. Tammana, R. Agarwal, and M. Lee, "Simplifying Datacenter Network Debugging with PathDump." in *OSDI*, 2016, pp. 233–248.

[22] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao *et al.*, "Packet-level telemetry in large datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 479–491.

[23] Microsoft Azure, "Network Watcher," https://azure.microsoft.com/en-us/services/network-watcher/, [Online; accessed 28-July-2018].

[24] Barefoot Networks, "Product Brief Tofino Page," https://www.barefootnetworks.com/products/brief-tofino/, [Online; accessed 12-February-2019].

[25] The P4 Language Consortium, "P4," https://p4.org/, [Online; accessed 12-February-2019].