Protecting OpenFlow using Intel SGX

Medina, Jorge; Paladi, Nicolae; Arlos, Patrik

Published in:
IEEE Conference on Network Function Virtualization and Software Defined Networks

DOI:
10.1109/NFV-SDN47374.2019.9039980

2020

Document Version:
Peer reviewed version (aka post-print)

Protecting OpenFlow using Intel SGX

Jorge Medina*, Nicolae Paladi† and Patrik Arlos‡
*Dept. Electrical and Computing Engineering
New Jersey Institute of Technology, New Jersey, USA, Email: jmc237@njit.edu
†Lund University & RISE Research Institutes of Sweden, Stockholm Sweden, Email: nicolae.paladi@ri.se
‡Dept. Computer Science, Blekinge Institute of Technology, Karlskrona Sweden, Email: patrik.arlos@bth.se

***Abstract*—OpenFlow flow tables in Open vSwitch contain valuable information about installed flows, priorities, packet actions and routing policies. Their importance is emphasized when collocated tenants compete for the limited entries available to install flow rules. OpenFlow flow tables are a security asset that requires confidentiality and integrity guarantees. However, commodity software switch implementations - such as Open vSwitch - do not implement protection mechanisms capable to prevent attackers from obtaining information about the installed flows or modifying flow tables. We adopt a novel approach to enabling OpenFlow flow table protection through decomposition. We identify core assets requiring security guarantees, isolate OpenFlow flow tables through decomposition and implement a prototype using Open vSwitch and Software Guard Extensions enclaves. An evaluation of the prototype on a distributed testbed both demonstrates that the approach is practical and indicates directions for further improvements.**

***Index Terms*—Software Defined Networks, Software Guard Extentions, integrity, confidentiality.**

I. Introduction

Flexible and powerful control over network flows is one of the core advantages of Software-Defined Networking (SDN). Flow rules stored in the switch network flow tables contain information on packet processing and routing. Flow rules are stored in memory, within a set of data structure rules, and managed by a classifier in flow tables.

Recent advancements in software programmable networks allow tenants to independently populate switch network flow tables to control communication between endpoints in their respective slices. Network flows are a valuable security asset: they contain information about traffic patterns between the endpoints, while network tenants are competing for the limited entries in flow tables [1].

Commodity software switches do not currently implement confidentiality or integrity protection of flow tables. An attacker can observe or modify installed flows by exploiting software vulnerabilities to access the switch host memory [2]. By *Observing* installed flows, an attacker can learn security-sensitive information such as network topology, flow patterns between the endpoints and flow priority defined by the network administrator. By *modifying* installed flows an attacker can exploit routing loopholes and avoid network defence mechanisms - for example route around a firewall or prevent mirroring packets to an intrusion detection system.

We describe an approach to protecting the confidentiality and integrity of network flows installed on software switches. We modify the network switch architecture to reduce the attack surface by isolating the OpenFlow flow tables and the flow rules from the rest of the code base. We describe the design and use Open vSwitch (OvS) to implement a prototype of the solution [3]. Finally, we evaluate our prototype to measure its performance and identify ways to further improve the approach.

Due to the page limit we exclude the internals of OvS packet forwarding and flow table protection by decomposing OvS; we refer the reader to [4].

The rest of this paper is organized as follows. In Section II we introduce the necessary background information. In Section III we present `OFTinSGX`, a library that enables OvS to allocate its OpenFlow flow tables and forwarding logic inside enclave memory. We evaluate the proposed solution in Sec-

tion IV, describe the current limitations and future work in Section V and conclude in Section VI.

II. Preliminaries

A. Open vSwitch Overview

OvS is an open source programmable switch [5]. It comprises a daemon (*ovs-vswitchd*), a database server (*ovsdb-server*) hosting a configuration database, and a switching module (*data path*) [6].

ovsdb-server hosts a database containing the OvS configuration including bridges, interfaces, tunnels and a list of IP addresses of managers and controllers that communicate with OvS. OvS clients (e.g, ovs-vswitchd, ovs-vsctl) use JSON-RPC[1] to retrieve configuration data from the ovsdb-server.

ovs-vswitchd is a user space daemon that follows the OvS configuration from ovsdb-server to initialize the bridge module, spanning the data path initialization, the flow tables and the translation process; flow table entries determine the actions that the switch executes on unknown incoming packets.

A *data path* implements packet forwarding. OvS is a flow-based switch, where clients install flows determining forwarding decisions. Flows are installed in a cache level structure that assists the data path to execute actions on received packets, e.g. allow, drop, learn or resubmit. For each ingress packet, the data path consults its cache and forwards the packet to its destination if matching entries exist. For each cache miss, the data path issues an upcall and forwards the packet to ovs-vswitchd. A data path can be deployed as a kernel module or in user space with additional firmware support [7].

B. Open vSwitch Packet Classification

Packet classification in OvS is computationally expensive, mostly due to the many types of matching fields. Matching is implemented in a hash table of flow rules, with matching fields hashes as keys. OvS uses a modified Tuple Space Search (TSS) algorithm [8] for packet classification. The algorithm searches through the hash map tables based on the maximum entry's priority and terminates after finding the highest priority matching flow rule.

[1]JavaScript Object Notation, Remote Procedure Call, https://www.jsonrpc.org

Early OvS releases implemented OpenFlow processing exclusively as a kernel module. However, the difficulty of developing and updating kernel modules motivated moving packet classification to user space. A multi-level cache structure kernel implementation compensates the resulting performance impact [5]. The cache structure consists of two levels with increasing lookup costs: a *microflow cache* (or Exact Match Cache) and a larger *megaflow cache*. The megaflow cache is key to forwarding performance tuning, since it covers multiple flow matches through wildcards [9].

C. Open vSwitch Forwarding Operation

Figure 1 illustrates the OvS components, utilities and interactions. When an incoming packet reaches the data path from a Network Interface Card (1), the forwarding process runs an exact match search (2). In case of a microflow cache match, the packet is sent to the specific table in the megaflow cache to retrieve the required actions. Otherwise, the forwarding process runs a second search in the next cache line (3). This search is computationally expensive, since the process visits every table in the megaflow cache until it encounters a matching flow. If there is no match in the megaflow, the data path notifies the ovs-vswitchd via upcalls that it lacks information to handle the packet (4).

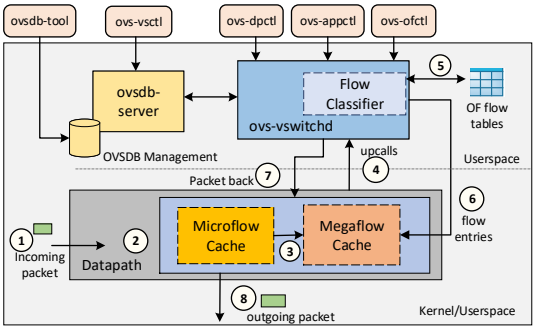

Figure 1: Open vSwitch components

Ovs-vswitchd uses the classification process (5) to obtain a matching rule via its OpenFlow flow

tables[2]. A matching rule contains a priority value, a wildcard mask, and actions for handling the unknown flow (5). Ovs-vswitchd returns to the data path, inserts the new entry in the cache structure (6) and returns the packet to the kernel (7). Finally, the data path forwards the packet to the intended destination (8). If no match is found in the OpenFlow flow tables, ovs-vswitchd sends a `packet_in` request to the controller for get a matching rule.

D. Trusted Execution Environments

We use SGX enclaves to create trusted execution environments (TEEs) during operating system runtime [10]. We use TEEs to allocate and initialize OpenFlow flow tables, together with sensitive information stored in flow rule structures. SGX enclaves rely on a trusted computing base of code and data loaded at enclave creation time, processor firmware and processor hardware. Program execution within an enclave is transparent to the underlying operating system and other mutually distrusting enclaves on the platform. Enclaves operate in the Enclave Page Cache, a range of dynamic random access memory that cannot be accessed by system software or peripherals [11]. The CPU is the root of trust of an enclave; it prevents access to the enclave's memory by the operating system and other enclaves.

E. Threat Model

We consider a powerful adverary, capable of exploiting software vulnerabilities in the host operating system and OvS, reload the OvS its binary, access the host memory and start arbitrary processes on the host. We rely on the confidentiality and integrity guarantees provided by SGX enclaves. Following the attacker model of SGX enclaves [10] we exclude side-channel attacks.

F. Related Work

Protecting the security assets of network elements is a topic of active on-going research. Jacquin proposed an architecture that used a hardware root of trust to remotely attest the integrity of virtualization hosts in SDN infrastructure [12]. Furthermore, commodity TEEs were used in case studies on securing network applications [13]. TruSDN is a framework for bootstrapping trust in an SDN infrastructure [14]. It supports secure provisioning of switches in SGX enclaves, a secure communication channel between switches and SDN controller, and secure communication between endpoints. *Trusted Click* [15] explores the feasibility of performing network processing in SGX enclaves. While none of the approaches above address the integrity and confidentiality of OpenFlow flow tables, they can be complemented with OFTinSGX to achieve this.

SCONE enables operators to protect confidentiality and integrity of computation in containers against adversaries with host root access [16]. An alternative approach to protecting virtual network functions running in containers, that avoids the excessive expansion of the trusted computing base is presented in [17].

Event Handler Eviction mitigates DoS attacks and overflow of OpenFlow flow table [18]. The mechanism uses two independent modules - learning module and flow checking module - although the event handler mechanism reduces overflow of flow tables and the risk of DoS attacks, it does not provide security guarantees to the OpenFlow flow tables. OFTinSGX protects both the integrity and confidentiality of the OpenFlow flow tables, as well as the forwarding logic and eviction process.

TLSonSGX protects the cryptographic material used by OvS instances to protect communication with SDN controllers [19]. This approach can be combined with OFTinSGX to enable wider security guarantees for OpenFlow switches.

III. OFTinSGX

We now present the design and implementation of OFTinSGX, a security mechanism allowing OvS to allocate its OpenFlow flow tables and forwarding logic inside enclave memory. OFTinSGX has four components: the *SGX OpenFlow tables*, the *SGX rule mechanism*, the *SGX Eviction Component* and the *SGX Tables dpif*. We illustrate the interaction of these components in Figure 2.

[2]We use "OpenFlow tables" referring to OpenFlow flow tables *in user space*, different from flow tables *in the data path*

A. OFTinSGX Components

SGX OFtables implement the OpenFlow flow tables running in the enclave memory. *SGX OFtables* host the forwarding logic to match classification rules for unknown incoming flows. The classification rules are subsequently translated to instructions installed in the data path.

The *SGX rule mechanism* is an interface implementing the logic mapping rules allocated in untrusted memory to classification rules in the enclave for matches in the SGX OFtables. It ensures that only rules created by an authenticated SDN controller are used to install entries in the data path. The SGX rule mechanism translates the SGX-wrapper calls to direct operations on the OpenFlow flow tables. It is structured as a hash map table (*SGX_cls_table*), enclosing SGX rules (*SGX_cls_rule*). We used the *hmap* OvS library to implement the SGX hash map table.

The SGX Eviction Component implements the rule eviction process. Based on given criteria, it removes a classification rule from the classifier when a table reaches the maximum number of allowed flows. The SGX Eviction Component also contains the eviction groups (struct `eviction_group`). The dpif SGX tables implement struct `table_dpif` to maintain updated information about the existing rules in the OpenFlow flow tables after flow update routines. This is necessary to monitor entries in the data path and maintain up-to-date statistics.

B. Workflow Summary

Prior to sending OpenFlow requests to OvS, a remote SDN controller attests the integrity of the enclave where the OpenFlow flow tables are hosted (1) [10]. Upon a successful attestation, the controller sends OpenFlow requests to operate the OpenFlow flow tables (2). If the requested operation is a rule addition, a rule is first allocated in untrusted memory and an SGX rule containing a classification rule is allocated in the enclave memory. The SGX Rule Mechanism receives the rule operation requests and handles the requests inside the enclave by calling the services of the respective processing component (3): lookup, eviction or flow revalidation. The SGX Rule Mechanism contacts the OpenFlow flow tables when the data path does not find instructions to handle an incoming packet. In this scenario, an upcall request for a rule lookup is passed to user space, containing the incoming packet the data path cannot handle (4). If a matching classification rule is available, the SGX Rule mechanism returns the address of the corresponding untrusted rule and the essential information about the matching classification rule to install an entry in the data path cache for the incoming packet, containing a flow and a mask (5). The entry in the data path is installed once the rule allocated in untrusted memory was identified and the actions to be executed on that packet were provided (6). The OpenFlow flow tables deployed in SGX enclaves are protected from malicious tampering (7). The SDN controller maintains a hash of the enclave and private key used to sign the enclave, and later uses them to establish an authenticated channel with the enclave. As a result, only an authenticated SDN controller can operate on the OpenFlow flow tables.

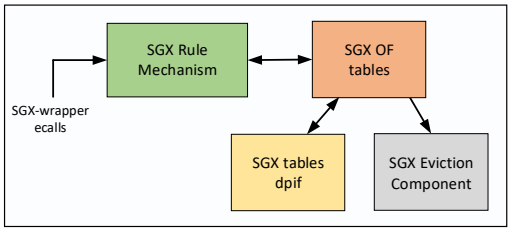

Figure 2: *OFTinSGX* architecture

IV. Evaluation

To evaluate the prototype implementation of the solution, we deployed a testbed that uses a Network Test Automation Systems (NTAS) instance. The testbed is based on the Distributed Passive Measurement Infrastructure (DPMI) [20] with Endace/Emulex DAG 3.6 cards [21] with accuracy of 60 ns and GPS to ensure high quality and rigour on the collected data. In the testbed, NTAS orchestrates the experimentation while DPMI collects and distributes packet data. In addition, packets are mirrored by full duplex measurement points (MPs)

that collect and store the relevant packets at specific locations for further analysis.

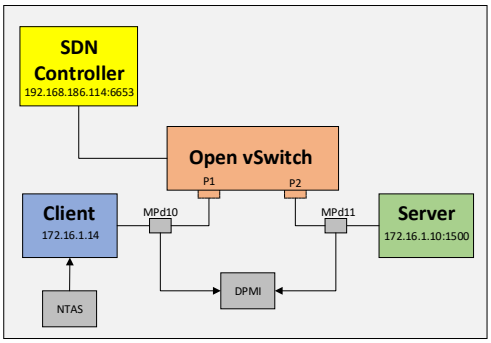

Figure 3: Testbed setup

The testbed includes four components (see Figure 3): a client (packet source); a server (packet sink); an instance of OvS; an SDN controller. Packets flowing in our testbed are mirrored by two MPs - from the client to OvS and from OvS to the server by MPd10 and MPd11. All aforementioned components run on separate hardware; We use the Ryu[3], SDN controller, version 2.8.

A. Performance Analysis

Our evaluation is based on the scenario that involves user space and interaction with an SDN controller. In this scenario, the client packet is sent from user space to the controller. Next, the controller installs a rule in the OpenFlow flow tables, and returns the packet to user space. We (1) flush the installed rules using *ovs-ofctl*, (2) send one packet from client to server, return to 1 and repeat over 10000 times. The generated flow corresponds to UDP traffic with a unique identifier to easily identify ingress and egress packets of the OvS instance, and to obtain its service time (T_i), i.e. delay through the device [22].

Table I shows the collected statistics and a relative overhead (rOH) between 10 and 13% in the first incoming packet delay. The relative overhead is calculated as $rOH = (E[SGX] - E[Plain])/E[Plain]*100$. This overhead is caused by the use of SGX enclaves: for each ecall to OpenFlow flow tables, the CPU transitions to the enclave mode, with a performance degradation. Overhead increases linearly with the number of ecall invocations. It is advisable to keep the number of ecalls as low as possible. In this scenario ecalls are invoked by methods for rule lookup, addition and deletion of rules.

[3]https://github.com/osrg/ryu

Table I: Summary of first incoming packet delay

PacketSize [Bytes]	OvS version	Mean [μs]	Std [μs]	Min [μs]	Max [μs]	rOH [%]
64	SGX	3367	493	1473	12511	12.8
	Plain	2986	413	1234	5086	
128	SGX	3356	490	1446	14602	12.6
	Plain	2981	435	1236	6668	
256	SGX	3393	471	1522	6784	11.7
	Plain	3038	440	1267	9493	
512	SGX	3431	494	1547	11740	10.4
	Plain	3108	442	1345	8000	
1024	SGX	3579	496	1727	11072	11.8
	Plain	3202	442	1461	5975	

V. Limitations and Future Work

A mechanism to protect the OpenFlow flow tables should span both the tables contents and complete representation of the rules allocated in untrusted memory. Isolating only the contents of OpenFlow flow tables does not address all security risks, since a classifier keeps only references of the classification rules. Classification rules are allocated in the struct `rule`, which is pointed by a struct `rule_dpif`, in untrusted memory. We limited the implementation scope to porting the contents of the classification rule to enclave memory and leave porting classification rules for future work.

VI. Conclusion

Flow tables in software switches carry security-sensitive data. Its integrity and confidentiality is essential for SDN security: an adversary capable to modify the flow tables can control the topology of the respective network deployment. Commodity

software switches have weak or no security mechanisms to protect the integrity and confidentiality of flow tables. We presented an approach to protect the flow tables through decomposition. We reduced the attack surface of software switches by isolating the flow table processing and management logic. While we implement a prototype using OvS and Intel SGX enclaves, the approach is generalizable to other switch implementations and isolated execution environments. The prototype evaluation results show the approach is practical for commodity platforms.

VII. Acknowledgments

This work was financially supported in part by the Swedish Foundation for Strategic Research, grant RIT17-0035 and EU H2020 project ASCLEPIOS, grant 826093. Jorge Medina was supported by a scholarship from the Swedish Institute.

References

[1] H. Cui, G. O. Karame, F. Klaedtke, and R. Bifulco, "On the fingerprinting of software-defined networks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 10, pp. 2160–2173, Oct 2016.

[2] K. Thimmaraju, B. Shastry, T. Fiebig, F. Hetzelt, J.-P. Seifert, A. Feldmann, and S. Schmid, "Taking Control of SDN-based Cloud Systems via the Data Plane," in *Proc. of the Symp. on SDN Research*, ser. SOSR '18. ACM, 2018, pp. 1:1–1:15.

[3] N. Paladi, J. Svenningsson, J. Medina, and P. Arlos, "Protecting OpenFlow Flow Tables with Intel SGX," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, ser. SIGCOMM Posters and Demos '19. New York, NY, USA: ACM, 2019, pp. 146–147.

[4] J. A. Medina Chirinos, "Deconstructing open vswitch for isolated enclaves : A security enabler for sdn data plane," Master's thesis, , Department of Computer Science and Engineering, 2018.

[5] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *Proc. 12th USENIX Conf. on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 117–130.

[6] B. Pfaff and E. B. Davie, "The Open vSwitch Database Management Protocol," Internet Requests for Comments, RFC Editor, RFC 7047, December 2013.

[7] D. Intel, "Data plane development kit," 2014.

[8] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *Proc. of the Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '99. ACM, 1999, pp. 135–146.

[9] O. vSwitch, "Tutorials Open vSwitch 2.9.90 documentation, performance," http://docs.openvswitch.org/en/latest/tutorials/, accessed: 2018-03-16.

[10] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. ACM, June 2013, p. 10.

[11] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative Instructions and Software Model for Isolated Execution," in *Proc. 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. ACM, June 2013, pp. 10:1–10:1.

[12] L. Jacquin, A. L. Shaw, and C. Dalton, "Towards trusted software-defined networks using a hardware-based Integrity Measurement Architecture," in *Proc. 1st IEEE Conf. Network Softwarization*, ser. NetSoft'15, April 2015.

[13] M.-W. Shih, M. Kumar, T. Kim, and A. Gavrilovska, "S-NFV: Securing NFV States by Using SGX," in *Proc. 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFV Security '16. ACM, March 2016, pp. 45–48.

[14] N. Paladi and C. Gehrmann, "TruSDN: Bootstrapping Trust in Cloud Network Infrastructure," in *Proc. 12th International Conf. on Security and Privacy in Communication Networks*, ser. SecureComm'16. Springer, Oct. 2016.

[15] M. Coughlin, E. Keller, and E. Wustrow, "Trusted Click: Overcoming Security Issues of NFV in the Cloud," in *Proc. ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFVSec '17. ACM, March 2017, pp. 31–36.

[16] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure Linux Containers with Intel SGX," in *Proc. 12th USENIX Conf. on Operating Systems Design and Implementation*, ser. OSDI'16. USENIX, November 2016, pp. 689–703.

[17] D. Girtler and N. Paladi, "Component integrity guarantees in software-defined networking infrastructure," in *2017 IEEE Conf. on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2017, p. 292.

[18] Y. Qian, W. You, and K. Qian, "Openflow flow table overflow attacks and countermeasures," in *2016 European Conf. on Networks and Communications (EuCNC)*, June 2016, pp. 205–209.

[19] N. Paladi, L. Karlsson, and K. Elbashir, "Trust anchors in software defined networks," in *Computer Security*, J. Lopez, J. Zhou, and M. Soriano, Eds. Cham: Springer International Publishing, 2018, pp. 485–504.

[20] P. Arlos, M. Fiedler, and A. A. Nilsson, "A distributed passive measurement infrastructure," in *International Workshop on Passive and Active Network Measurement*. Springer, Berlin, Heidelberg, 2005, pp. 215–227.

[21] Endace Measurement Systems, "Endace," 2019. [Online]. Available: https://www.endace.com/endace-high-speed-packet-capture-solutions/oem/dag/

[22] P. Carlsson, D. Constantinescu, A. Popescu, M. Fiedler, and A. A. Nilsson, "Delay performance in ip routers," in *2nd International Working Conf. (HET-NETs '04)*, 2004.