Secure Software Defined Networks Controller Storage using Intel Software Guard Extensions

Qasmaoui Youssef[1]
Hassan First University, Faculty of Sciences and Techniques, Computer Networks, Mobility and Modeling laboratory: IR2M, Settat, Morocco

Maleh Yassine[2]
IEEE Senior Member
Sultan Moulay Slimane University
LaSTI Laboratory, Beni Mellal
Morocco, Settat, Morocco

Abdelkrim Haqiq[3]
IEEE Senior Member
Hassan First University, Faculty of Sciences and Techniques, Computer Networks, Mobility and Modeling laboratory: IR2M, Settat, Morocco

***Abstract*—The SDN controller is the core of the software-defined network (SDN), which provides important network operations that needs to be protected from all type of threats. Many researches have been focusing on different layers of security regarding the SDN controller such as Anti-DDOS system or enforcement of TLS connection between the controller and the Open-vswitches. One of the major security threats targeting any program is the environment execution itself (e.g. Operating system and the hardware itself). Intel's Software Guard Extension (SGX) offers a sloid layer of security applied to applications by creating a Trusted execution environment. SDN controller relay on a storage module to keep sensitive data such as Flow Rules, users' credentials and configuration files. Protecting this side of the SDN controller is a must in term of security. To date, no work has been conducted considering SDN controller storage security using Intel SGX. This paper introduces an SGX enabled SDN controller. The new controller ensures the integrity and the confidentiality in a trusted execution environment by leveraging a recent hardware technology called intel SGX. This technology provides a trusted and secure enclave. Enclaves are sealed and unsealed by intel SGX attestation mechanisms to protect the executed code and data inside live memory and disk from being altered by any unauthorized access. High privileged codes such as the OS itself is kept from altering data inside enclaves. We implemented the Intel SGX using the Floodlight SDN controller running a real enabled Intel SGX hardware. Our evaluation shows that the SGX enabled SDN controller introduces a slightly observable performance overhead to the floodlight controller compared to advantages in term of security.**

Keywords*—Software defined networks; software guard extensions; storage; integrity; confidentiality

I. Introduction

In recent years, the network research community has experienced a period of intense activity that has led to the emergence of different architectures or paradigms such as the SDN. The centralization (logical or physical) of the control plan, had to bring the expected flexibility to the network applications and allow to respond to many concrete use cases.

Software-Defined Network is a new paradigm of network architecture that aims to design a data plane that is fully programmable and separated from the control plane [1]. The Control Plane manages decisions about how and where to transmit network traffic through system configuration, management and exchange of routing table information. The Data/Forwarding Plane manages the actual transmission of packets to the destination network according to the logic of the control plan. Behind this separation, there are three main objectives:

- The separation of network intelligence (control plan) from equipment (data plan).
- The provision of a logically centralized view of the global physical network.
- Providing an abstraction of programmable network equipment using Interfaces of application programming (API).

Among the innovations of this new paradigm is the programmability of network equipment and applications. New network applications can be transparently programmed and deployed using standard APIs. However, its implementation in the data plane remains one of the greatest challenges for research.

Protecting sensitive data from been altered or access gained by any authorized manner is present since the beginning of the programming time, a challenge that many have taken the race to solve it [2]. SDN technology has been introduced to solve the complexity of configuring network hardware. SDN enabled networks relay on a central decision making called the SDN controller to handle request coming from network devices such as switches and routers [3], [4]. Those requests are transported via API's using open flow protocol [5] and optionally secured using TLS.

The SDN controllers represent the most delicate part of the SDN architecture as it consists of the brain of the network, making it vulnerable to all sort of attacks [6], [7]. Security issues may vary depending on the level of interest targeted by a malicious person; it goes from Denial of service to traffic redirection and flow rules modification [8]. The SDN controller software is run on vast untrusted platforms, including operating systems, hypervisors, firmware, and hardware. This large machine base is growing complex and difficult to verify. For e.g., an OS such as Linux has 17 million line of code, however 662 vulnerabilities related to CVE have been recorded in 2019, such as memory corruption, transverse directory, unauthorized code execution. Execution

of normal and security-critical applications running on shared resources controlled by untrusted computing machines raises security threats. Running the SDN programs in such environments represent a considerable threat to its normal operations.

To solve the issue, one of the solutions is Trusted execution environments (TEE). TEE guarantee security by relying on less hardware and software computing base. Hardware is commonly considered to be a stable base since the cost and sophistication of hardware attacks usually are high. This has lead to the development of a secure running environment by industrial hardware companies for a safety-critical application that maintains little reliance or less dependency upon the operating system and hypervisor. Up to today, we found two main technology which are ARM Trust Zone Technology, Intel Software Guard Extensions (SGX) [9][10].

The objective of this work is to propose a secure architecture by programming new modules and adding security functions at the control plan storage based on Intel SGX. Then, evaluate the impact of SDN architectures at the performance level.

The rest of the paper is organized as follow. The next section will include a background and related works followed by the proposed model to secure SDN controller storage using Intel SGX, then we present the results of the implementation with discussion. Finally, we conclude this work with a conclusion and perspectives for future work.

II. Background

Trusted Execution Environment (TEE) is a tamper-resistant computing ecosystem that works on a separate kernel. It guarantees the validity of the executed programs, the security of the runtime components (e.g. memory, CPU registers, and critical Input / Outputs) and the secrecy of the executed code, data and runtime states are maintained in non-volatile memory [11]. In addition, the remote certificate shall be given to show its trustworthiness to third parties. The contents of TEE are not static; they can be changed safely. TEE condemns all software-related threats as well as hardware threats against the main memory of the operating system. Attacks leveraging backdoor authentication bugs are futile.

Fig. 1 illustrates the difference between a Trusted execution environment and an ordinary OS.

The most common TEE environments are Intel SGX and ARM TrustZone [12]. Both Intel SGX and ARM TrustZone are hardware TEE environments, but the process behind building a trusted environment with trusted code is distinct. Intel SGX provides a trusted environment for trusted programs that run on top of current untrusted device software. Whereas, ARM TrustZone is building a new, trusted ecosystem for trustworthy applications that operate on trustworthy device software and hardware that are only accessible to the trusted Configuration.

In this paper we focused on intel SGX technology to deploy our secure SDN controller. The Choice of using Intel SGX was taken depending on the much benefit that supersede ARM TrustZone, benefits such as documentation, maturity and the availability of hardware enabled machines. The majority of researchers uses Intel SGX to deploy a trusted execution environment.

Intel Software Guard Extensions (Intel SGX) provides hardware-based memory encryption to isolate portions of code and application-specific data in memory. Intel® SGX allows user-level code to assign private memory regions (called enclaves) designed to be protected against processes running at higher privilege levels. Only the Intel® SGX solution provides such a granular level of control and protection.

Intel SGX has been used to secure flow tables inside Open-Vswitches as mentioned in related works.

Fig. 2 shows the architecture of a typical intel SGX enabled environment.

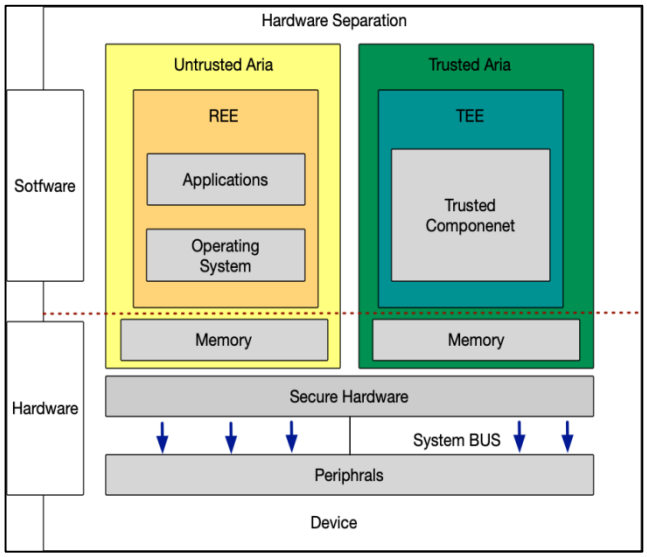

Fig. 1. Trusted Execution Environment.

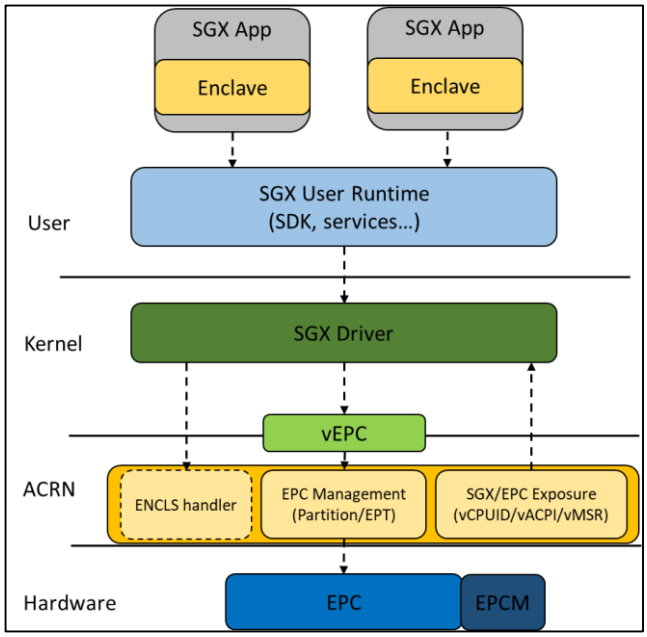

Fig. 2. Intel SGX Architecture.

SGX is built to be reliable; this is done in a variety of ways, including robust enclave delivery, sealing and attestation. Intel summarizes SGX's protections [13], [14] as follows;

- Memory is secured against observation and modification from outside the enclave, using an in-die Memory Encryption Engine (MEE) [15], with a secret that rotates on every boot. This protection notably works against host hypervisors, other enclaves, and anything running in supervisor mode.
- Enclaves will attest or confirm their identities to a competitor with the aid of a permanent hardware identification key for asymmetric encryption.
- Computer calls are designed to schedule and pass power in and out of the enclave. Arguments are safely mapped according to the concept of a static enclave.
- SGX does not protect itself against reverse engineering or side-channel attacks: to counteract this is the responsibility of the client.

III. Related Works

There are only a few works in the literature that discuss SDN security using SGX. Intel Software Guard Extensions (SGX) has provided the general purpose of the hardware-assisted TEE referred to as Intel SGX. Intel SGX is an expansion of the x86 architecture with a new range of security-related instructions [16]. These instructions are used by security-critical programs to create a hardware-assisted trust environment referred to as an enclave [17]. Intel SGX enclave maintains secrecy through hardware-maintained data layout and honesty tests by encrypting data and code when it is outside the CPU package [18]. Intel SGX is a centralized security architecture, and the trustworthy TCB computing foundation is known to be a CPU package.

TruSDN is a mechanism for bootstrapping confidence in the technology of SDN [19]. Supports the safe supply of switches in SGX enclaves, a protected communication channel between switches and SDN controllers, and secure communication between endpoints.

Trusted Click [20] investigates the viability of network processing in SGX enclaves. Although none of the above methods discusses the credibility and anonymity of OpenFlow flow tables, they can be complemented by OFTinSGX to accomplish this. SCONE allows operators to protect the secrecy and integrity of computing in containers against host root access adversaries [21]. An alternative approach to securing virtual network functions running in containers, which prevents the unnecessary expansion of the trusted computing foundation, is proposed in [22]. Event Controller Eviction mitigates DoS attacks and OpenFlow Application overflow [23]. This framework uses two different frameworks – the learning module and the flow control module – while the case handler system prevents overload and DoS attacks, the OpenFlow flow tables have no security guarantees. OFTinSGX maintains the integrity and confidentiality of OpenFlow tables and the reasoning for forwarding and disposal procedures.

TLSonSGX guarantees that OvS authorities retain communication with SDN controllers and the cryptographic material they use [24]. This methodology can be paired with OFTinSGX to provide broader security assurances for OpenFlow switches. Fig. 3 shows the TLSonSGX system design.

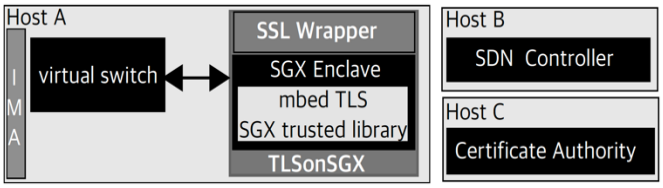

Fig. 3. TLSonSGX System Design.

In recent works OFTinSGX has been proposed by [9], which has four components: SGX OpenFlow table, SGX rule structure, SGX Eviction component, and SGX tables dpif, which helps OvS to delegate its OpenFlow tables and forward logic to enclave memory.

The important limitation of this work is that the abstraction of only the contents of the OpenFlow flow tables does not address all security concerns, as the classifier only includes references to the classification rules. The procedure used to control the OpenFlow flow tables can cover both the contents of the tables and the full description of the rules assigned to the untrusted memory.

Fig. 4 illustrates the OFTinSGX Architecture.

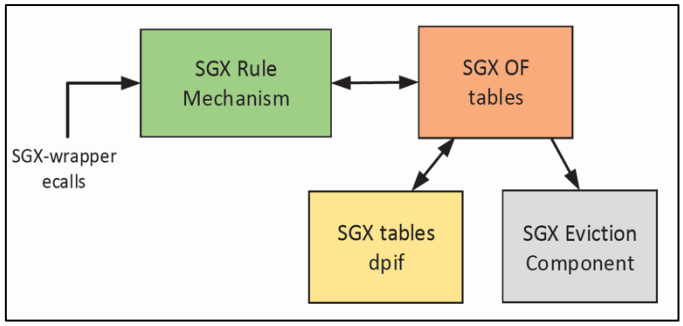

Fig. 4. OFTinSGX Architecture.

IV. The Proposed SGX SDN Controller

A. SDN Storage Module Overview

This section presents the design and the architecture of the proposed model to secure SDN controller storage using Intel SGX.

Generally, the SDN controller is relying on a storage module that handles all sensitive data; controller data are mostly configuration files such as flow rules. In our case, we use the floodlight SDN controller to implement our approach. Fig. 5 illustrates the Floodlight architecture. The proposed model consists of rewriting the storage module code taking into consideration the Intel SGX technology.

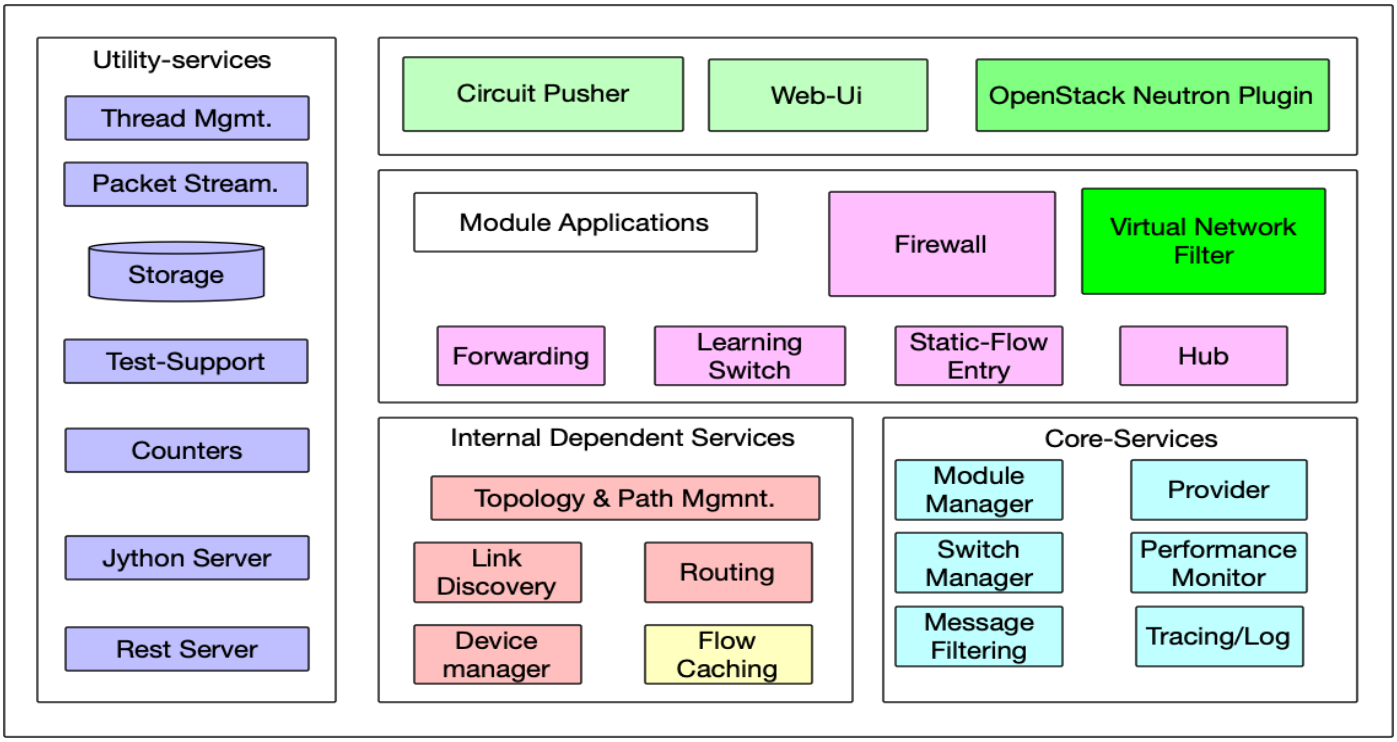

Fig. 5. Floodlight Architecture.

The storage module consists of four main modules: link discovery module(responsible for discovering and maintaining the status of links in the OpenFlow network), Device manager module (tracks devices as they move around a network and defines the destination device for a new flow), Static flow entry pusher (allows a user to manually insert flows and groups into an OpenFlow network) module and QOS module(give a user a way to simply push QoS state to switches that support these features) interacting with the primary storage module which is mainly a NoSQL database residing in the random-access memory. Fig. 6 shows the interaction between the four-module and the storage module.

A compromised host can present a huge issue to the NoSQL database making it vulnerable to several attacks such memory dump or memory exhaustion attacks. Our method consists of hardening all the space used by the NoSQL database by implementing the Intel SGX technology to prevent any damage to the RAM area used by the SDN controller.

B. SDN Enabled SGX Architecture

SDN enabled SGX model operates as an intermediate system. Specifically, it executes a process daemon that intercepts all the call made to the ordinary storage module by the controller application. These calls are translated into corresponding functions of the SDN enabled SGX model enclave. For instance, when a new open flow rules need to be inserted, the call is made via our interface and call the corresponding function inside the enclave via a JNI (Java Native Interface). The code residing in the enclave return the right value depending on the result of the function via the interface. In this design, all the data structures related to the file system are continuously kept in the Enclave Page Cache(EPC), which is a subset of DRAM that cannot be directly accessed by other software, including system software and SMM code. The CPU's included memory controllers also reject DMA transmissions targeting the EPC, thus protecting it from access by other peripherals.

The NoSQL database files are maintained by the enclave and protected by the encryption mechanisms of the SGX technology, making a dump of the memory useless as its encrypted and cannot be read.

Data is decrypted by the SGX unsealing mechanism: this process occurs while entering the enclave and is thus secured by the CPU borders. Consequently, a dump of the EPC (enclave page cache) would be captured and blocked by the SGX protection mechanisms.

Fig. 7 shows the proposed SDN enabled SGX model.

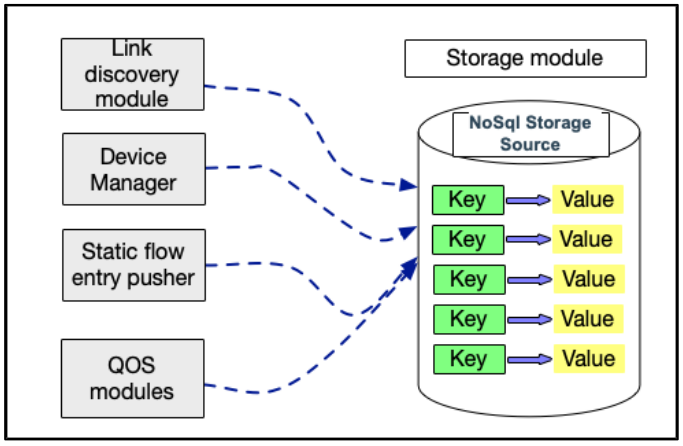

Fig. 6. The Proposed Storage Module.

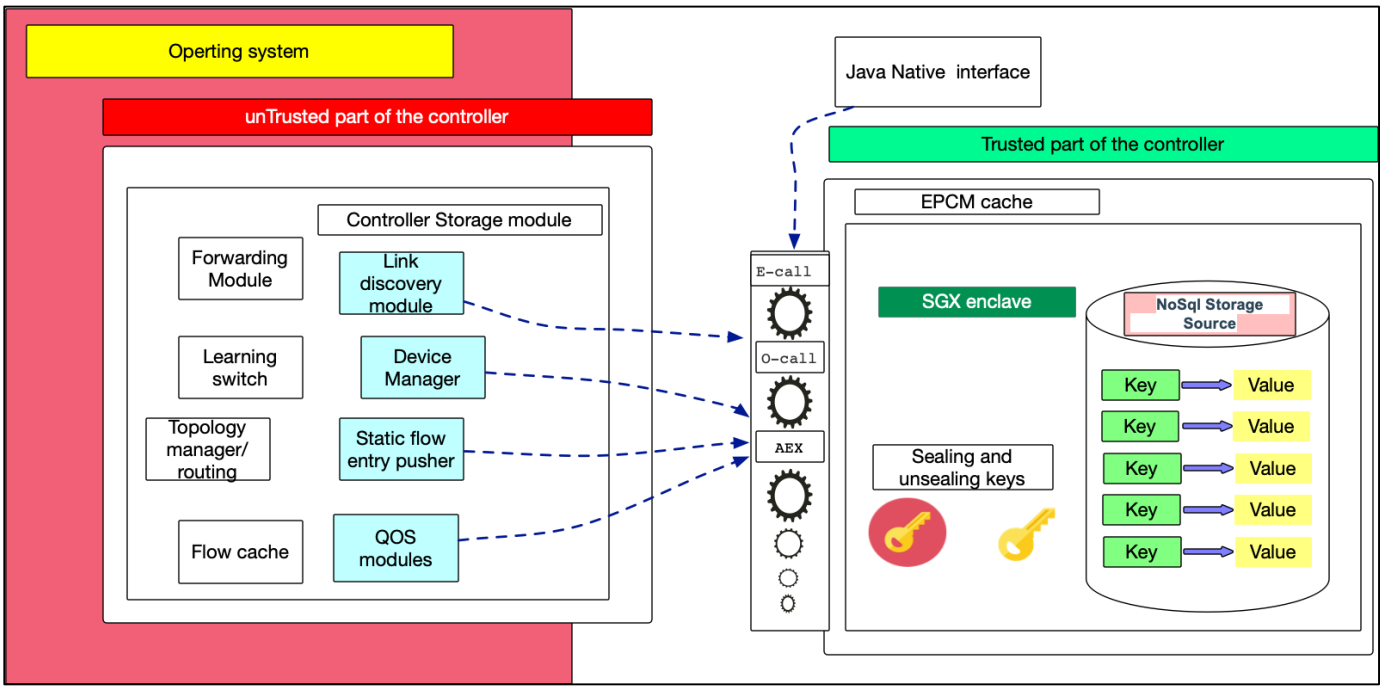

Fig. 7. The Proposed SDN Enabled SGX Model.

V. Implementation and Evaluation

A. Implementation Setup

We implemented the SDN controller enabled SGX on a Linux platform holing an SDN controller named floodlight developed using JAVA programming language. Since intel SGX SDK support only C/C++ language, we used the Java Native Interface (JNI) to make the right call to C functions to handle intel SGX operations. The next scenario demonstrates the different phases of the implementation, this scenario is almost generic to any Intel SGX enabled case:

- Step 1: An enclave is created by the untrusted part of the application.
- Step 2: The enclave must be initialized via a launch token, fetched and provided by Intel's launch enclave.
- Step 3: Access to the LE and other architectural enclaves, e.g., the quoting enclave (QE) and the provisioning enclave (PE), is provided by the Intel application enclave service manager (AESM). SGX libraries provide an abstraction layer for communicating with the AESM.
- Step 4: Execution of a trusted function which executes an ECALL.
- Step 5: The ECALL goes through the SGX call interface to bring the executing thread inside the enclave.
- Step 6: An OCALL is executed once the execution in the trusted environment completes.
- Step 7: Finally giving the control back to the caller.

The program run on HP Proliant gen10 server with the following Configuration:

- Intel XEON E-2174G with intel SGX support.
- 32GB of RAM.

To generate traffic we use mininet simulator [25], mininet was setup and configured on five different computers to be able to generate 2000 request by each one.

The written code mainly focuses on the Storage Source Service, which is the main interface of the storage module of floodlight controller, also some of the dependencies were taken into accounts such as IDebugCounterService and IRestApiService.

Table I shows a summary of the code changes made to the storage module inside the floodlight controller. The most significant part of the added code consists of new (JAVA NATIVE INTERFACE) JNI interface and make files, floodlight storage also has been modified to accept the call from OCALL I/O peripherals.

B. Performance Analysis

Our evaluation is based on the scenario that involves a number of nodes making calls to the SDN controller. In this case, the network devices receive various request to dispatch packets from a script that run on several separated machines. Decision making is sent from the data layer represented by the network devices. Next, the controller responds with the corresponding flow rule. Rules present inside the Open-vswitches are deleted, so the network devices are forced to make calls to the controller. The script sends over 10000 requests to several separated switches. The generated traffic sent to the controller allows us to take statistics and compare them to the normal scenario.

TABLE I. SUMMARY OF THE CODE CHANGES

Lines of code of SGX enabled SDN controller		
component	***Lines of original code***	***% changed***
Floodlight storage module	3688	9.05% (350 lines)
JNI interface	1203	100% (1203 line)
Other(e.g. make files)	200	100% (200 lines)

A normal scenario consists of the same use case but with non-modified controller. Table II, shows the collected measures and a relative Overhead. Overhead is between 7 and 10% in the first run. The overhead is calculated as shown in the following equation:

$$OH = (E[SGX] - E[normal])/E[normal] * 100$$

The increase of the overhead is caused by the use of SGX enclaves. For each ECALL to the storage module to gather flow rules, the CPU switch to the enclave mode, resulting to an increase of execution time. Overhead increases linearly with the number of e-call invocations. The state of the controller gets stabilized after a while since the flow rules are cached inside the open flow tables within the open vswitches.

TABLE II. COLLECTED MEASURES AND OVERHEAD

Controller response time			
Number of requests to the SDN controller	***Normal scenario***	***Intel SGX enabled SDN controller***	***Overhead***
10000 (first run)	3300 us	3630	10 %
10000 (second run)	2170 us	2320	7 %
10000 (third run no flow flashing)	300 us	313 us	4.33 %

Table II shows the Overhead interval between two test cases. The same parameters are kept during the two tests. The deference between the overhead in both runs is due to CPU consumption by the OS itself and other floodlight operations.

The third run on the scenario was conducted in a special case where the open-vswitches are not flushed so flow rules are kept inside the switches flow tables, open flow switches call the SDN controller only if there is no entry matching the upcoming packets. There was a slit increase in time execution in this case and that was because of normal operations executed by the controller itself.

VI. CONCLUSION

The concept of SDNs or Software Defined Networks is an architecture that facilitates network management and control, and allows rapid introduction of network services through programming and separation of the control plane from the data plane. With this new architecture, administrators can manage the network in a unified way from the control plane, and can introduce or eliminate any service through the application plane without changing the physical infrastructure. New network applications can be transparently programmed and deployed using standard APIs. Most modern SDN controllers can run on any OS However, its implementation in the data domain remains one of the biggest challenges for storage security at the control plane level.

In this work, we proposed using Intel SGX to provide additional security to the general intent of the Execution Environment of applications.

In order to evaluate the performance impact of our SDN enabled Intel SGX architecture we implemented our SDN controller model enabled SGX on a Linux platform holing an SDN controller named floodlight. The results of our model implementation show the efficiency of our model without any major cost in term of performance.

VII. FUTURE WORKS

As a perspective, we would like to test our model in large test platforms to estimate its capacity and limitations and compare it to the TEE provided by ARM named ARM Trustzone.

Other evolution will be conducted on other point of views such as using intel SGX to build the entire application instead of the storage module alone.

REFERENCES

[1] M. F. H. and M. A. Ismail, “Distributed Shadow Controllers based Moving Target Defense Framework for Control Plane Security,” Int. J. Adv. Comput. Sci. Appl., vol. 10, no. 12, pp. 150–156, 2019, doi: 10.14569/IJACSA.2019.0101221.

[2] Safaa MAHRACH and Abdelkrim HAQIQ, “DDoS Flooding Attack Mitigation in Software Defined Networks,” Int. J. Adv. Comput. Sci. Appl., vol. 11, no. 1, pp. 693–700, 2020, doi: 10.14569/IJACSA.2020.0110185.

[3] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, “A Survey on the Security of Stateful SDN Data Planes,” IEEE Commun. Surv. Tutorials, 2017.

[4] J. Son and R. Buyya, “A Taxonomy of Software-Defined Networking (SDN)-Enabled Cloud Computing,” ACM Comput. Surv., vol. 51, no. 3, p. 59, 2018.

[5] N. McKeown et al., “OpenFlow: enabling innovation in campus networks,” ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, 2008.

[6] K. Bhushan and B. B. Gupta, “Distributed denial of service (DDoS) attack mitigation in software defined network (SDN)-based cloud computing environment,” J. Ambient Intell. Humaniz. Comput., pp. 1–13, 2018.

[7] H. D’Cruze, P. Wang, R. O. Sbeit, and A. Ray, “A Software-Defined Networking (SDN) Approach to Mitigating DDoS Attacks,” in Information Technology-New Generations, Springer, 2018, pp. 141–145.

[8] A. QASMAOUI, Y., & HAQIQ, “Enhanced Solid-Flow: An Enhanced Flow Rules Security Mechanism for SDN,” IAENG Int. J. Comput. Sci., vol. 47, no. 3, 2020.

[9] J. Medina, N. Paladiy, and P. Arlosz, “Protecting OpenFlow using Intel SGX,” in 2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Nov. 2019, pp. 1–6, doi: 10.1109/NFV-SDN47374.2019.9039980.

[10] S. Costan, V., & Devadas, “Intel SGX Explained,” IACR Cryptol. ePrint Arch., vol. 86, pp. 1–118, 2016.

[11] V. Lefebvre, G. Santinelli, T. Müller, and J. Götzfried, “Universal Trusted Execution Environments for Securing SDN/NFV Operations,” 2018, doi: 10.1145/3230833.3233256.

[12] J. Winter, “Trusted Computing Building Blocks for Embedded Linux-Based ARM Trustzone Platforms,” in Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, 2008, pp. 21–30, doi: 10.1145/1456455.1456460.

[13] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han, “A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications,” 2015, doi: 10.1145/2834050.2834100.

[14] A. Baumann, M. Peinado, and G. Hunt, "Shielding Applications from an Untrusted Cloud with Haven," ACM Trans. Comput. Syst., vol. 33, no. 3, Aug. 2015, doi: 10.1145/2799647.

[15] S. Gueron, "A Memory Encryption Engine Suitable for General Purpose Processors," Cryptol. ePrint Arch. Rep. 2016/204, 2016.

[16] Intel®, "Intel® Sofware Guard Extensions Programming Reference," (Cited pages 13 14.), 2014.

[17] I. Graydon, E. Beatty, S. Paul, M. N. Us, and J. A. Hauck, "Method and apparatus to provide secure application execution," 2006.

[18] F. X. Simon P Johnson, Uday R Savagaonkar, Vincent R Scarlata and and C. V. R. McKeen, "Technique for supporting multiple secure enclaves," US Patent 8,972,746., 2015.

[19] N. Paladi and C. Gehrmann, "TruSDN: Bootstrapping Trust in Cloud Network Infrastructure," in International Conference on Security and Privacy in Communication Systems (pp. 104-124), 2017, pp. 104–124.

[20] M. Coughlin, E. Keller, and E. Wustrow, "Trusted Click: Overcoming Security Issues of NFV in the Cloud," in Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, 2017, pp. 31–36, doi: 10.1145/3040992.3040994.

[21] S. Arnautov et al., "{SCONE}: Secure Linux Containers with Intel {SGX}," in 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), Nov. 2016, pp. 689–703, [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov.

[22] D. Girtler and N. Paladi, "Component integrity guarantees in software-defined networking infrastructure," in 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Nov. 2017, pp. 292–296, doi: 10.1109/NFV-SDN.2017.8169858.

[23] Ying Qian, Wanqing You, and Kai Qian, "OpenFlow flow table overflow attacks and countermeasures," in 2016 European Conference on Networks and Communications (EuCNC), Jun. 2016, pp. 205–209, doi: 10.1109/EuCNC.2016.7561033.

[24] N. Paladi, L. Karlsson, and K. Elbashir, "Trust Anchors in Software Defined Networks," in Computer Security, 2018, pp. 485–504.

[25] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using Mininet for emulation and prototyping Software-Defined Networks," in 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), 2014, pp. 1–6, doi: 10.1109/ColCom Con.2014.6860404.

AUTHORS' PROFILE

Youssef Qasmaoui received his Master degree in Networks and IT Security and his Bachelor degree in Networks and IT Systems, respectively in 2012 and 2010 from the Faculty of Sciences and Techniques (FST), Settat – Morocco. He also received a second Master degree in Information System Engineering at the University of Western Brittany at Brest – France. He is also doing his Ph.D. thesis at the FST, Settat - Morocco. His research interests include Software Defined Networks, Virtual Laboratory and Networks Security.

Yassine MALEH is a cybersecurity professor and practitioner with industry and academic experience. He is a Ph.D. degree in Computer Sciences. Since 2019, He working as a professor of cybersecurity at Sultan Moulay Slimane University, Morocco. He was working for the National Port agency (ANP) in Morocco as an IT Security Manager from 2012 to 2019. He has published more than 60 research papers. This includes 7 books, 20 book chapters, 14 peer-reviewed journal articles, and 20 peer-reviewed conference manuscripts.

He has served on Program Committees of more than 20 conferences and events and has organized many Symposiums/Workshops as a General Chair. He is an editor of a number of journals including Editor in Chief: International Journal of Smart Security Technologies (IJSST). Associate Editor: IEEE Access (Impact Factor: 4.09), International Journal of Digital Crime and Forensics (IJDCF) and International Journal of Information Security and Privacy (IJISP). He was also a Guest Editor of a special issue on Recent Advances on Cyber Security and Privacy for Cloud-of-Things of the International Journal of Digital Crime and Forensics (IJDCF), Volume 10, Issue 3, July-September 2019. He served and continues to serve as a reviewer of numerous prestigious journals such as Elsevier Ad Hoc Networks, IEEE Network Magazine, IEEE Sensor Journal, ICT Express, and Springer Cluster Computing, etc…

Prof. Abdelkrim HAQIQ has a High Study Degree (Diplôme des Etudes Supérieures de troisième cycle) and a PhD (Doctorat d'Etat), both in the field of modeling and performance evaluation of computer communication networks, from Mohammed V University, Faculty of Sciences, Rabat, Morocco. Since September 1995 he has been working as a Professor at the department of Applied Mathematics and Computer at the Faculty of Sciences and Techniques, Settat, Morocco. He is the Director of Computer, Networks, Mobility and Modeling laboratory: IR2M. He is an IEEE senior member and an IEEE Communications Society member. He is also a member of Machine Intelligence Research Labs (MIR Labs), Washington, USA. He was a co-director of a NATO Multi-Year project entitled "Cyber Security Analysis and Assurance using Cloud-Based Security Measurement system", having the code: SPS-984425. Prof. Abdelkrim HAQIQ's interests lie in the areas of modeling and performance evaluation of communication networks, mobile communications networks, cloud computing and security, emergent technologies, Markov chains and queueing theory, Markov decision processes theory, and game theory. He is the author and co-author of more than 170 papers (international journals and conferences/workshops). He supervised 15 PhD thesis and co-supervised 3 PhD thesis. Actually, he is supervising and co-supervising other PhD thesis. He is an associate editor of the International Journal of Computer International Systems and Industrial Management Applications (IJCISM), an editorial board member of the International Journal of Intelligent Engineering Informatics (IJIEI) and of the International Journal of Blockchains and Cryptocurrencies (IJBC), an international advisory board member of the International Journal of Smart Security Technologies (IJSST) and of the International Journal of Applied Research on Smart Surveillance Technologies and Society (IJARSSTS). He is also an editorial review board of the International Journal of Fog Computing (IJFC) and of the International Journal of Digital Crime and Forensics (IJDCF). Prof. Abdelkrim HAQIQ was a chair and a technical program committee chair/member of many international conferences and scientific events. He was also a Guest Editor and Co-Editor of special issues of some journals, books and international conference proceedings. From January 1999 to December 1999 he had a Post-Doctoral Research appointment at the department of Systems and Computers Engineering at Carleton University in Canada. He also has held visiting positions at the High National School of Telecommunications of Paris, the Universities of Dijon, Versailles St-Quentin-en-Yvelines and LAAS CNRS, Toulouse in France, the University of Ottawa in Canada, the FUCAM in Belgium, the National Engineering School of Sfax, Tunisia, the University of Naples Federico II, Italy and the University of Algarve, Portugal.