Challenges Towards Protecting VNF With SGX

Juan Wang
Wuhan University
jwang@whu.edu.cn

Shirong Hao
Wuhan University
shirong@whu.edu.cn

Yi Li
Wuhan University
liyi1023@whu.edu.cn

Chengyang Fan
Wuhan University
cyfan@whu.edu.cn

Jie Wang
Wuhan University
iwangjye@gmail.com

Lin Han
Wuhan University
hchina@whu.edu.cn

Zhi Hong
Wuhan University
hongzhi@whu.edu.cn

Hongxin Hu
Clemson University
hongxinhu@clemson.net

ABSTRACT

Network Function Virtualization (NFV) is an emerging technology to implement network functions in software, which reduces equipment costs (CAPEX) and operational cost (OPEX) through decoupling network functions from network dedicated devices and deploying them on high-volume standard servers and running as virtual instances. However, due to running in a shared and open environment and lacking the protection of proprietary hardware, virtual network functions (VNFs) face more security threats than traditional network functions. Hence, it is crucial to build a trusted execution environment to protect VNFs. In this paper, we first analyze the challenges for VNF security protection. We then propose a lightweight and trusted execution environment for securing VNFs based on SGX and Click. To demonstrate the feasibility of our approach, we implement a DDoS defense function on top of our environment and conduct paramilitary evaluations. Our evaluation results show that our system only introduces manageable performance overhead for protecting VNFs.

CCS CONCEPTS

• **Security and privacy → Denial-of-service attacks**; *Trust frameworks*;

KEYWORDS

NFV, VNF, Intel SGX, Click, Trust

ACM Reference Format:
Juan Wang, Shirong Hao, Yi Li, Chengyang Fan, Jie Wang, Lin Han, Zhi Hong, and Hongxin Hu. 2018. Challenges Towards Protecting VNF With SGX. In *SDN-NFV Sec'18: 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, March 19–21, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3180465.3180476

SDN-NFV Sec'18, March 19–21, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5635-0/18/03…$15.00
https://doi.org/10.1145/3180465.3180476

1 INTRODUCTION

Network Function Virtualization (NFV) implements network functions in software through decoupling network function from network dedicated devices and deploying them on high-volume standard servers and running as virtual instances. It can reduce both equipment costs (CAPEX) and operational cost (OPEX) and speed up software-oriented network innovation to bring new services and generate new revenue streams. Recently, more and more companies have started to embrace NFV so as to adapt to increasing complicated network environment and IT virtualization. Despite the benefits, virtual network functions (VNFs) face more serious security challenges than the protection of proprietary hardware[1–3]. Traditional network functions run in a dedicated network appliances that have their own separated CPU and memory, which can provide a relatively closed running environment. However, VNFs run in a shared and open environment and bring more security risks, such as the isolation threat, code and policy being tampered with, network sensitive data leakage, unlimited open ports, and side channel [4]. Hence, it is crucial to build a trusted execution environment for VNFs.

Some research efforts have been devoted to building trusted VNFs. ESTI NFV Security and Trust Guidance [1] proposes to provide trusted protection based on HSM (Hardware Security Module), TPM (Trusted Platform Module), and vTPM (virtual Trusted Platform Module). NetBricks [5] leverages a safe language (Rust) and LLVM [6] to build a zero copy soft isolation. It provides memory isolation software by using the type safe language and achieves high performance by adoption LLVM as an optimization back-end of compilers. NetBricks can also ensure that only a single NF can access a packet so as to guarantee that packet isolation in common cases. OpenNetVM [7] runs NFs in lightweight Docker containers based on the NetVM architecture. It provides NF isolation through container mechanisms, such as namespace and capability. Recently, Intel Software Guard Extensions (SGX) [2, 3, 8, 9] has been proposed to build secure running environment that can be used to protect VNFs. Intel SGX can isolate an application to a hardware sandbox called an enclave so that OS, driver, BIOS or virtual machine monitor (VMM) can not access the code and data in the enclave. However, running existing network functions in SGX enclaves can cause a high performance overhead [10].

In this paper, we propose a lightweight and trusted execution environment for protecting VNFs. We leverage the modularity feature provided by Click [11] to design VNFs and only run the security-aware elements of VNFs in SGX enclaves. We also design state management and migration mechinisms to achieve fine-grained state consistency and ensure lose-free and order-preserving for NF scaling. To demonstrate the feasibility of our approach, we use a DDoS defense function as a case study. We have implemented the DDoS defense function on top of our environment and evaluated it in three different configurations, *no-SGX mode*, *SGX in simulation mode*, and *SGX in hardware debug mode*. The results show that our design has the lower performance overhead.

The rest of this paper is organized as follows. In Section 2, we describe the challenges for protecting VNF using SGX. Section 3 presents our method for building trusted VNF based on Click and SGX. Section 4 discusses our system implementation and experimental results. Conclusion is drawn in Section 5.

2 CHALLENGES FOR VNF PROTECTION WITH SGX

SGX provides a new way for securing applications through putting the security-related data and code to a trusted container. Intel calls this container an enclave. SGX assumes that everything on a system, such that OS, driver, BIOS, and virtual machine monitor (VMM), except CPU is untrusted. For an SGX program, an enclave is a trusted execution environment where only the application self can access its data and code. Since SGX provides a run-time secure environment for applications, some existing work has explored to use SGX to protect VNFs. S-NFV [12] tried to use SGX to protect the VNF state, but it ignores other data such as policies. LightBox [13] presented a secure SGX-assisted system, which supports secure and generic middleboxes for network function outsourcing. However, it does not address the issues about state migration and usable SGX-aware APIs. In addition, they directly ported BRO to SGX enclave, which could cause larger performance overhead. Trusted Click [10] proposed to use SGX to enforce data privacy and provide guaranteed computation for network function outsourcing in cloud. Although it provided a framework to secure network function outsourcing, it also did not address some important issues including state migration. To effectively and efficiently protect VNFs using SGX, our study reveals following challenges:

- **Performance Overhead:** We have done a deep analysis of Snort code, and decomposed it into the seven modules including *sniffer*, *pre-processor*, *rules matching*, *state processing*, *policy*, *system interface*, and *library*. Furthermore, we divided them to trusted modules and untrusted modules. In addition, we put the trusted modules, such as *pre-processor*, *rules matching*, *state processing*, and *policy*, to a secure enclave in real SGX device. We found that those trusted modules still caused a very high performance overhead although we have carefully divided Snort modules and only run security-related modules in the enclave. Our evaluation shows that the performance overhead for the state processing in SGX-protected Snort modules is about 10 times than running those modules in *no-SGX mode*. If we run all security-related modules in the SGX enclave, the performance becomes even worse.
- **Function Decomposition:** Taking Snort as an example again, if we want to secure Snort with SGX, we need to divide it to trusted and untrusted modules and only run trusted modules in SGX enclaves. However this is a hard as well as time-consuming task. For example, the policy processing in Snort runs across almost all Snort procedures. Thus, if we want to use SGX to protect the policy and policy processing of Snort, we may need to upload almost all of the Snort code and data to enclaves, which would bring unacceptable performance overhead. Hence, a new approach for designing lightweight VNFs based on SGX is desirable.
- **State Synchronization and Migration:** State management is crucial for VNFs. Previous work [14, 15] has addressed state synchronization and migration mechanism for VNF scaling. However, Those work does not consider the security issue during the procedure of state synchronization and migration, which are based on shared and unencrypted buffers and vulnerable to security threats, such as information theft.
- **Usable APIs:** It is currently a tough task for developers to design secure applications using SGX. The developers need to have specialized knowledge in the SGX platform, principles, and SDKs. Hence, there is another challenge to design easy-to-use APIs for developers who can use SGX in the development of secure VNFs.

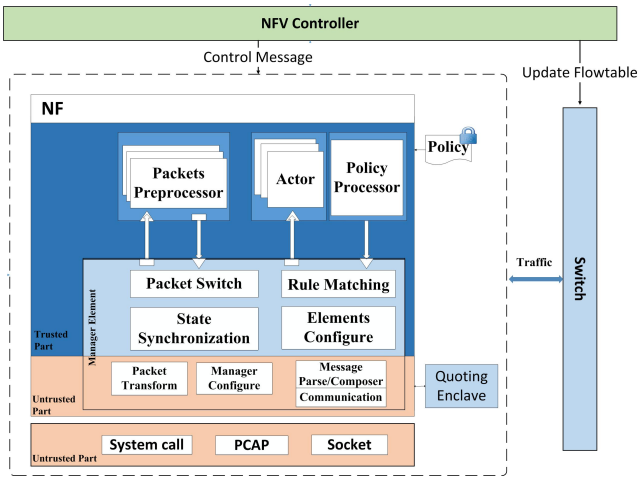

Figure 1: Architecture Overview

3 OVERVIEW OF ARCHITECTURE

Our goal is to design a lightweight and trusted execution environment for VNFs based on SGX and Click. Click is a flexible, modular software architecture for building network applications. Building trusted VNFs based on Click and SGX has three advantages. First, the modularity of Click allows us to compose different elements for easily building different VNFs. Second, the modular VNFs enable more fine-grained NF scaling. Third, modular VNFs protected by SGX can only lead to a low performance overhead.

Figure 1 shows the overview architecture of our trusted environment for VNFs. The architecture is divided into two parts: a trusted part and an untrusted part. The trusted part protects the packets preprocessor, policy resolution, decision actor and the state synchronization modules. The untrusted part implements an interface to system calls and provides the support for enclaves, such as command line and configuration file parsing and so on.

The trusted part contains the key elements, such as *Packets Preprocessors*, *Actors*, *Policy Processor*, *State Synchronization*, *Elements Configure*, and *Packet Switch*, which are protected by SGX enclaves. *Packets Preprocessors* may include many different elements which perform some preprocessing operations on the fetched packets, such as assembling, decoding, filtering and so on. *Policy Processor* is used to decrypt the sealed policy with SGX. *Actors* can also have many different elements which make decisions for packets processing such as discarding, rate limiting, logging, outputting and so on. *Packet Switch* is responsible for sending packets to the corresponding processing ports through searching rule tables. *State Synchronization* is responsible for state synchronization and migration. The *Rule Matching* element is responsible for searching rule tables, matching the rules with the corresponding processing action. After the action of matching is successful, packets will be sent to the corresponding action processing module. *Elements Configure* is responsible for initializing and building the process of detection and execution.

The untrusted part contains *Packet Transform*, *Manager Configuration*, *Communication*, and *Message Parser/Composer* elements. *Packet Transform* element transfers packets to the trusted part. *Manager Configure* initializes the implementation of various modules. *Communication* element is responsible for creating the connection with the controller. *Message Parser/Composer* element parses the messages sent by the controller, and then sends messages to the trusted part.

The *Quoting Enclave* is used to make attestation for the virtual network function which is protected with SGX.

Through the modular design for VNF, we can easily put the small and security-related elements to SGX enclaves. Meanwhile we can flexibly extend the designed virtual functions and elastically scale the virtual network functions.

3.1 State Synchronization

In NFV environment, when the traffic can not be proceeded by a single instance, it is necessary that the controller divides network traffic to two or more instances by updating the flow tables. Hence, state migration [14, 16, 17] is a vital problem for VNFs. Our approach achieves fine-grained state consistency and ensures lose-free and order-preserving for VNF scaling. Loss-free means that all packets should be processed without any packet lose. Order-preserving indicates that packets should be processed according to the original order when they are forwarded to the new NF instances. An overview of the state synchronization process is illustrated in Figure 2, which consists of the following steps:

① We assume there is an existing NF instance A. When the traffic of A is too large, the controller creates a new NF instance B.

② The controller copies the configuration of the instance A to the instance B. According to the configuration, the instance B completes initialization and begins to run. The instance B waits to synchronize with the instance A, caching the packet F2 forwarded by the switch.

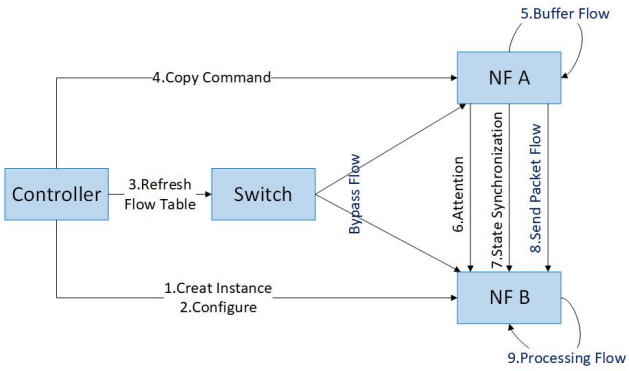

Figure 2: State Synchronization Process

③ The controller updates the flow tables, and the switch forwards part of the data flows to the instance B.

④ The controller sends the command of copying instance to the instance A.

⑤ The NF instance A runs the classifier, and caches the data packets that should have been sent to the instance B. The data packets are actually sent to the NF instance A, because the flow table has not taken effect due to the time delay.

⑥ Then, the instance A and the instance B perform remote authentication and exchange keys.

⑦ The instance A sends the encrypted internal states of the stream, which needs be synchronized to the instance B until state synchronization is completed.

⑧ The instance A sends buffered packets to the instance B.

⑨ The instance B processes the packets from the instance A, and then processes the latest packets.

4 IMPLEMENTATION AND EVALUATION

To evaluate our design, we design a virtual DDoS defense function [18, 19] with the support of SGX and Click. We implement several elements including *FromDevice*, *Manager*, *ToDevice*, *AggregateIPFlows*, *CheckLength*, *IPFilter*, *Packet Switch*, *UDP Flood*, *SYN Flood*, *DNS Amplify*, *Discard* and *Delay*. The element of *FromDevice* reads packets from network devices and then the packets flow into the *Manager* element. Furthermore, the packets are processed in SGX enclaves. When it has finished, *ToDevice* element sends benign packets to network devices.

The main elements of our system are run in SGX enclaves. The *AggregateIPFlows* element uses source and destination addresses and source and destination ports to distinguish flows, and set the aggregate annotation on every passing packets to a flow number. The *CheckLength* element checks every packet's length. The *IPFilter* element filters IP packets by contents. The *UDP Flood* element, *SYN Flood* element and *DNS Amplify* element can detect different DDoS attacks. The *Packet Switch* element develops corresponding strategies according to the flow table delivered by NFV controller. Then Packet Switch element sends packets to action elements through

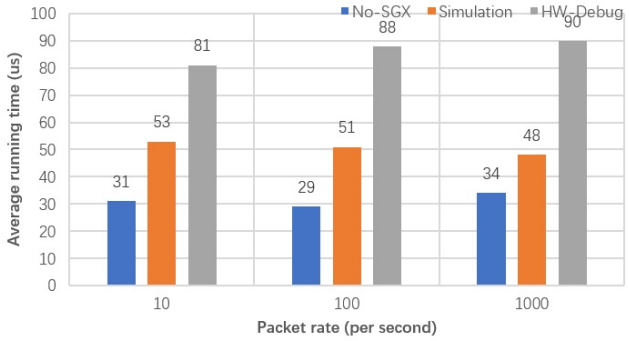

Figure 3: Performance Overhead with SGX and without SGX

different outgoing ports. Packets can be sent to the *Discard* element, the *Delay* element, or the *Todevice* element to make different decision processing. In addition, we create state tables storing packet information (such as source IP and destination IP) for state synchronization. Furthermore, we develop OCALL (e.g., ocall_print) and ECAL Linterfaces (e.g., ecall_generat_router) in the *Manager* element between enclave and outside elements as enclaves cannot directly use system calls.

To measure the performance of our design, we have conducted experiments on two DELL Inspiron 7567 laptops with Intel Core i7-7700HQ 2.80GHz CPU and 8 GB memory, which run a 64-bit Ubuntu Linux and the SGX SDK 2.0.

We performed a simple experiment to evaluate the overhead of our system in three different configurations: *no-SGX mode*, *SGX in simulation mode*, and *SGX in hardware debug mode*. In order to simulate DDoS attacks, we use *Scapy* on another computer to send different rates and types of DDoS packets to our system. In our experiments, the rates of packets is 10, 100 and 1000 per second, and the type of packets is UDP and TCP.

Our evaluation results are summarized in Figure 3, regardless of the rates of receiving packets, the average time of packets processing in the same mode is almost the same. In *no-SGX mode*, the average time is about 31.3us. In *simulation mode* and *hardware debug mode*, the processing time is about 50.7us and 86.3us respectively. On the whole, the DDoS detection system running on top of SGX in *simulation mode* gives 62% additional running time delays, and 176% additional running time delays in *hardware debug mode* compared to *no-SGX mode*.

5 CONCLUSIONS

In this paper, we have analyzed the challenges of using the SGX to protect VNFs. We have also proposed a lightweight and trusted framework for protecting VNFs based on SGX and Click. Taking the DDoS defense function as a use case, we have implemented the DDoS defense function based on our framework. We have also evaluated our system and our evaluation results show that our system only introduce very low performance overhead for securing VNFs.

ACKNOWLEDGMENTS

This work was partially sponsored by grants from National Natural Science Foundation of China (No. 61402342 and No. 61772384), National Basic Research Program of China (973 Program) (No. 2014CB340600). This work was also partially supported by grants from National Science Foundation (NSF-OAC-1642143, NSF-CNS-1700499, and NSF-DGE-1723663). We are also very grateful to Gu Jianjun from Intel and Liu Xin from Nationalz for their generous help to this work. They have made some valuable comments and suggestions on our work.

REFERENCES

[1] Bob Briscoe. Network functions virtualisation (nfv); security; problem statement. 2014.
[2] B Jaeger. Security orchestrator: Introducing a security orchestrator in the context of the etsi nfv reference architecture. In *IEEE Trustcom/bigdatase/ispa*, pages 1255–1260, 2015.
[3] Yeping Liu, Zhigang Guo, Guochu Shou, and Yihong Hu. To achieve a security service chain by integration of nfv and sdn. In *Sixth International Conference on Instrumentation Measurement, Computer, Communication and Control*, pages 974–977, 2016.
[4] Mahdi Daghmehchi Firoozjaei, Jaehoon Jeong, Hoon Ko, and Hyoungshick Kim. Security challenges with network functions virtualization. *Future Generation Computer Systems*, 2016.
[5] Cesare Alippi, Romolo Camplani, Manuel Roveri, and Gabriele Viscardi. Netbrick: A high-performance, low-power hardware platform for wireless and hybrid sensor networks. In *IEEE International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 111–117, 2012.
[6] Adve V. Lattner C. Llvm: A compilation framework for lifelong program analysis transformation. 2014.
[7] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, K. K. Ramakrishnan, and Timothy Wood. Opennetvm: A platform for high performance network service chains. In *The Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, pages 26–31, 2016.
[8] Prerit Jain, Soham Desai, Seongmin Kim, Ming Wei Shih, Jae Hyuk Lee, Changho Choi, Youjung Shin, Taesoo Kim, Brent Byunghoon Kang, and Dongsu Han. Opensgx: An open platform for sgx research. In *NDSS*, 2016.
[9] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. Vc3: Trustworthy data analytics in the cloud using sgx. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 38–54. IEEE, 2015.
[10] Michael Coughlin, Eric Keller, and Eric Wustrow. Trusted click: Overcoming security issues of nfv in the cloud. In *ACM International Workshop on Security in Software Defined Networks Network Function Virtualization*, pages 31–36, 2017.
[11] Eddie Kohler. *The click modular router*. Massachusetts Institute of Technology, 2001.
[12] Ming Wei Shih, Mohan Kumar, Taesoo Kim, and Ada Gavrilovska. S-nfv: Securing nfv states by using sgx. pages 45–48, 2016.
[13] Wang C Duan H, Yuan X. Sgx-assisted secure network functions at near-native speed. pages 1–14, 2017.
[14] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: enabling innovation in network function control. In *ACM Conference on SIGCOMM*, pages 163–174, 2014.
[15] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *NSDI*, volume 13, pages 227–240, 2013.
[16] Aaron Gember-Jacobson and Aditya Akella. Improving the safety, scalability, and efficiency of network function state transfers. In *ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, pages 43–48, 2015.
[17] A H M Jakaria, Wei Yang, Bahman Rashidi, Carol Fung, and M. Ashiqur Rahman. Vfence: A defense against distributed denial of service attacks using network function virtualization. In *Computer Software and Applications Conference*, pages 431–436, 2016.
[18] Seyed K Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: flexible and elastic ddos defense. In *Usenix Conference on Security Symposium*, pages 817–832, 2015.
[19] Bahman Rashidi and Carol Fung. Cofence: A collaborative ddos defence using network function virtualization. In *International Conference on Network and Service Management*, 2017.