Protecting Privacy of NF States in NFV Management Systems with Trusted Executed Environments

Cong Liu, Yong Cui, Wenfei Wu and Jianping Wu
Tsinghua University

1. INTRODUCTION

With the wide deployment of virtualized network functions (NF) in multi-tenant networks [6], NFV management system and its standards are proposed (e.g., OpenNF, OPNFV [4, 7, 3]). These NFV management systems allow operators to flexibly deploy NF instances in networks to achieve better utilization and security. However, as pointed by Baumann et al.[2], tenants are in risk to trust the NFV operators not "peeking" their information in their NF instances. For example, it is possible that NFV operators can steal contents of a web cache by accessing the cache's memory; they can also get statistics information from NFs such as IPS, load balancers.

To eliminate the risk of privacy violation, several valuable enhancements to existing NFV managements are proposed. mcTLS [6] assumes data plane traffic is encrypted by Transport Layer Security (TLS) protocol and integrates NFs into TLS to allow partial processing of network traffic. S-NFV [8] proposed to run NF instances in trusted execution environment to prevent operators stealing NF states.

While these works can protect tenants' information during data transmission and inside individual NFs, there are still challenges regarding to *NF state protection* within the scope whole system. In this paper, we explore *how to protect NF states globally within an NFV management system* and address the challenges to design such a system with existing security technologies.

The first challenge is that NF states are not always statically stored in NF instances, may appear at different locations in the whole NFV management system, and thus are hard to protect. For example, an NF may have logging systems to record some states, NF states may be migrated between NF instances for performance reasons (e.g., load balancing). Thus, we need a complete anatomy of NF state management in NFV frameworks to seal all possible appearances of NF states from NFV operators.

The second challenge is from the contradiction between the limited trusted execution environment (e.g., trusted memory size) and the amount of NF states that need protection (e.g., the cache of a content may be large). This limitation drives us to carefully design mechanisms to store states in both trusted and untrusted zone in the system (including memory and file systems). This state management process would need a careful division of functionalities as well as cryptographic key management.

2. BACKGROUND

NFV state management. In multi-tenant networks, a tenant proposes the network requirements (including end hosts, topology, NFs on paths), and the network operator creates concrete deployment in a physical infrastructure. We also assume the tenant provides NF images to process network traffic (for the reasons of performance, customized logic, or security), and the operator would clone images to NF instances and deploy them in the physical network (with a goal of global optimization, e.g., maximizing infrastructure utilization). The tenant NF also provides state management interfaces (e.g., get/put flow states from/to NF instances) to the operator or controller, so that the operator can flexilbly manage NF placement and state migration [4].

Trusted execution environment. Intel SGX [1] is a trusted execution environment shipped with Intel CPU. With SGX, we are able to run sensitive software in an isolated environment called *enclave*. The enclave memory will be protected by CPU and cannot be read by privileged software including OS. The protected software can be attested from a remote server on the correctness and integrity. SGX gives us the opportunity to protect sensitive NF states in NF instances [8].

Threat model. We focus on protecting the privacy of NF data including the configuration and state. On the NFV platform, we assume the CPU hardware with Intel SGX extension is trusted, but the other hardware parts (e.g., memory and I/O devices) and the host OS are compromised to collect private user data. The controllers (both NFV and SDN) and switches are also compromised so we should not pass any sensitive data to the controller for management reason. The controller can watch the external status in NFs, e.g., the 5-tuple of active flows, CPU and memory usage, etc. We do not consider deny of service. We trust the NFV platform will provide NF-runnable environment, otherwise the customer will not buy the service.

3. DESIGN

Anatomy of NF states in NFV management systems. In a typical NF packet processing loop [9], a packet would be

read from networks, processed and transformed, and sent to networks, and then states are updated. In addition to their appearance during the packet processing, they may also be used for management, including northbound APIs (e.g., get/put flow states) and local logs.

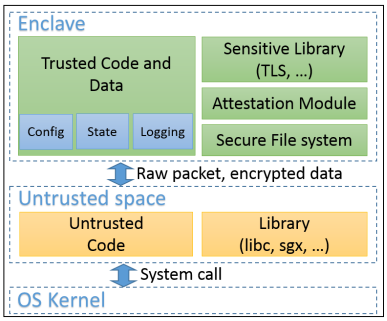

Figure 1: Protection on NFV software

Protecting states within packet processing. The key issue to protect states during packet processing is to find the boundary between the sensitive states processing and non-sensitive ones, and then the state-sensitive code snippet should be put in enclave.

Inspired by S-NFV and mcTLS [8, 6], we first categorize NFs, and define the boundary for each category. The first category of NFs protects their network traffic (using encryptions such TLS and HTTPs) as well as NF states. For example, a web cache would like to protect the cached object without letting the operator to know its content. The second category of NF does not mind exposing traffic but tries to protect the processing logic. For example, a customized DDoS NF would make judgment according to statistics of traversing packets, and the NF vendor would like to protect its smart recognition patterns (or rules). Referring to the NF state anatomy, state variables and state update should always in enclave, while for the first category, per-packet processing should also be put into enclave.

In implementation, the functions inside and outside the enclave should run with limited data exchange. A design with narrow interfaces between the functions (only passing raw packets and encrypted data) and safety check (e.g., return value) is needed. A challenge in implementation is that the state may be large in size, and cannot be put into SGX enclave completely (< 128MB). To solve this problem, we propose to encrypt states within enclave and store the encrypted states out of the enclave.

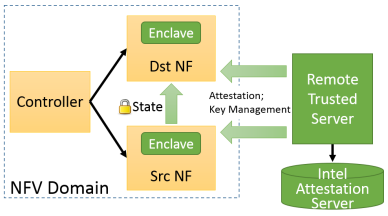

Figure 2: Protection on state movement

Protecting states out of packet processing. The general idea is to encrypt state data inside enclave and only allow decrypting the state in another enclave that is owned by the same tenant. In order to do this, the remote trusted server is involved in the state movement protocol to do the verification and key management as shown in Figure 2.

When the NFV controller issues a state movement operation, the source NF sends a request to the remove server for the state key. Using the allocated key, the source NF encrypts all the state data before sending out of the enclave. For per-flow state, each entry is encrypted individually and the flow keys are kept. The ciphertext is associated with a transaction ID. When received encrypted state data, the destination NF sends a request to the remote server. The remote server decides whether to provider the key to the requesting NF. The key management is independent to the state management protocol, so we can apply our key management to different control mechanisms, such as controller based management in OpenNF [4] and peer-to-peer state management in DiST [5].

Similarly, we can protect sensitive data in logging systems. States-related logs are first encrypted and then stored in log files with the same key provided by the remote server.

Protecting states from forged NF instances. The NF image was delivered to the NFV provider before and is free to be viewed. The NFV controller can decided when and how many instances to be created. Once created, the controller informs a remote server deployed by the NF tenant. The trusted remote server then initiates remote attestation to the NF instance to validate if the software is running correctly in SGX enclave and has not been modified. After the attestation, the remote server creates a trusted communication path with the NF and provisions the initial configuration data. The trusted path is maintained for state key management.

4. REFERENCES

[1] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for cpu based attestation and sealing. In *HASP*, 2013.

[2] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, pages 267–283. USENIX Association, 2014.

[3] ETSI. Network function virtualization (NFV) Use cases, 2013. `http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf`.

[4] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. Opennf: Enabling innovation in network function control. In *Proceedings of SIGCOMM*, Chicago, Illinois, Aug. 2014.

[5] B. Kothandaraman, M. Du, and P. Sköldström. Centrally controlled distributed vnf state management. In *Proc. of HotMiddlebox'15*, pages 37–42. ACM, 2015.

[6] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-context tls (mctls): Enabling secure in-network functionality in tls. In *ACM SIGCOMM CCR*, volume 45, pages 199–212. ACM, 2015.

[7] C. Price and S. Rivera. Opnfv: An open platform to accelerate nfv. *White Paper*, 2012.

[8] M.-W. Shih, M. Kumar, T. Kim, and A. Gavrilovska. S-nfv: securing nfv states by using sgx. In *Proc. of SDNNFV Security*, pages 45–48. ACM, 2016.

[9] W. Wu, Y. Zhang, and S. Banerjee. Automatic synthesis of nf models by program analysis. In *Proc. of HotNets*, pages 29–35. ACM, 2016.