S-NFV: Securing NFV states by using SGX

Ming-Wei Shih Mohan Kumar Taesoo Kim Ada Gavrilovska

School of Computer Science
Georgia Institute of Technology
266 Ferst Dr NW
Atlanta, Georgia
{mingwei.shih, mohankumar, taesoo}@gatech.edu, ada@cc.gatech.edu

ABSTRACT

Network Function Virtualization (NFV) applications are stateful. For example, a Content Distribution Network (CDN) caches web contents from remote servers and serves them to clients. Similarly, an Intrusion Detection System (IDS) and an Intrusion Prevention System (IPS) have both per-flow and multi-flow (shared) states to properly react to intrusions. On today's NFV infrastructures, security vulnerabilities many allow attackers to steal and manipulate the internal states of NFV applications that share a physical resource. In this paper, we propose a new protection scheme, S-NFV that incorporates Intel Software Guard Extensions (Intel SGX) to securely isolate the states of NFV applications.

Categories and Subject Descriptors

•Security and privacy → Security in hardware; Virtualization and security; Network security; •Networks → Middle boxes / network appliances;

Keywords

Middlebox; NFV; VNF; Intel SGX

1. INTRODUCTION

Network Function Virtualization (NFV) is a way to package network functions (NFs) traditionally performed by specialized physical appliances into virtual machines that can run on any physical server. The datacenter NFV infrastructure provides the necessary capabilities—computational resources and network paths—to establish the environment in which the virtualized network functions (VNFs) can execute. Many network functions (NFs) tend to create and maintain internal states to enable complex, intricate cross-packet and cross-flow analysis. Such a rich packet processing becomes an essential component in modern NFs that implements a wide range of applications, such as Content Distribution Network (CDN), Intrusion Detection System (IDS), and Intrusion Prevention System (IPS).

In general, NF states encompass private per-flow states (e.g., used for correctly tracking per-flow packets), shared multi-flow states (e.g., used for accounting for or managing packets from a single end-point), and function-wide global states (e.g., like cached data in CDNs). These NF states contain end-user data, such as IP address, end-user host details, cached user content, etc. For instance, CDNs [9] cache objects from origin servers based on client requests. Cached objects are user data, such as profile pictures, which should be protected from malicious access. Moreoever, this protection is a top priority for CDNs, running as VNFs, in datacenter NFV infrastructure.

The ETSI security specification [5] for NFV points out the risks with datacenter NFV infrastructure such as hypervisor introspection, where the NFV confidentiality and integrity are not guarded. This is because hypervisor introspection can enable the ability to view, inject, and/or modify operational state information associated with NFV through direct or indirect methods. Instead of guarding the entire guest OS state, we focus on the end-user states maintained in VNFs and the code accessing these states. Our goal is to provide confidentiality and integrity to the end-user states maintained in the VNFs.

In this paper, we take a first step toward solving this problem by using a yet-to-be commodity security scheme, called Intel SGX. Intel SGX allows a processor to instantiate a secure region of address space known as enclave; it then protects execution of the code within the enclave, even from malicious privileged code or hardware attacks such as memory probes. Haven [7] shows the benefits of Intel SGX for running existing server applications in the cloud with adequate level of trust and security.

Our solution, S-NFV, provides a secure framework for NFV applications. S-NFV provides an interface to move the VNF states and state processing code inside the enclave. Since VNF's functionality is tightly coupled with their states, the framework needs to provide a model to only allow relevant states (which need protection) to be moved to the enclave. Also, for the framework to provide the right abstraction for the different types of states (per-flow, shared, and global), the challenge lies in understanding the state usage model in various NFV applications. Along with securing VNF states, we also propose secured administrator access to configure rules used by these states and to view logs generated by these state processing. We use OpenSGX [10] to demonstrate securing Snort [11] application's per-flow state. Also, we peform the preliminary evaluation using a SGX-equipped machine.

2. OVERVIEW

2.1 Network Functions and Their States

To understand the protection that a solution like S-NFV must provide, we first analyze the state requirements associated with typical NFs. For this, we hand-picked five representative NFV ap-

SDN-NFVSec'16, March 11 2016, New Orleans, LA, USA

DOI: http://dx.doi.org/10.1145/2876019.2876032

Function	Description	Per-Flow State	Shared/Multi-Flow State	Global/All-Flows State
NAT64	NAT64 function performs in-line address translation from IPv6 to IPv4 address and vice-versa	**Struct:** nat64_binding **Desc:** NAT IP binding **Size:** 38 Bytes	None	None
Snort	Snort is an Intrusion Detection/Prevention System.	**Struct:** TagNode **Desc:** Track packets per flow **Size:** 51 Bytes	**Struct:** TagNode **Desc:** Track packets per host **Size:** 51 Bytes	Blacklisted IPs, URI, Files etc
PCEF	Policy Control and Enforcement Function (PCEF) is a telecom node used to provide policy and charging functionality by performing DPI.	**Struct:** NA **Desc:** Per flow charging **Size:** NA	**Struct:** NA **Desc:** Per mobile subscriber charging - multiple flows from the subscriber **Size:** NA	None
HAProxy	HAProxy can be configured as a HTTP proxy and Load-balancer for servers.	**Struct:** session, connection **Desc:** Each session represents 2 connection states **Size:** session-208 Bytes, connection-148 Bytes	None	Struct Server - contains server information to load balance
Squid	Squid is a Web caching proxy with global states.	None	None	Web Object - shared by multiple flows and different VNFs

Table 1: The internal state of NFV applications can be securely protected by S-NFV through SGX.

plications, including packet processing, IDS and CDN (see Table 1). We observe that these network functions vastly differ in the state access behavior with respect to their per-flow, shared and global state requirements, as categorized in Table 1.

1. NAT64 [2] contains only per-flow states that bind an IPv4 address to another IPv6 address. The binding is created when the first packet for the flow is processed by NAT64 and persists for the lifetime of that flow. Any malicious modification to the NAT state, in the middle of packet processing, will disrupt the flow from the server or client side.
2. Snort [11] can be configured as an online Intrusion Prevention System (IPS). In this mode, Snort maintains the per-flow state to prevent malicious packets by inspecting incoming and outgoing traffic. A malicious access to the Snort state may allow untrusted parties to read end-user details like an IP address.
3. Policy Control and Enforcement Function (PCEF) is a telecommunication node that maintains users' accounting states, comprising per-flow bytes and packet count. In our study, we focus only on the accounting state for subscribers in PCEF. According to the 3GPP specification [1], the policy for charging can be based either on volume or time, meaning that the billing state can be sent to the offline billing function after a certain time (e.g., data accounted in every 600 seconds) or after a data limit (e.g., every 10 KB data accounted). Malicious modifications to these states will create incorrect billing records for the subscribers.
4. HAProxy [3] maintains a per-session state (i.e., two connection states): one from a client to HAProxy and another from HAProxy to the backend server. The session state contains the information for the current HTTP transaction associated with the connection. Access to the existing HTTP transaction allows intercepting HTTP transaction details. Modifying the HTTP transaction state results in disrupting the HTTP transactions.
5. SQUID [4] is a web caching proxy that dynamically caches web pages based on client requests. The cached content are the NFV states for SQUID. The size of the cached content is arbitrary, based on the size of the cached web page. The cached web pages are similar to CDNs, where malicious access to the web page results in accessing end-user data like profile pictures, end-user accounts, etc.

The different states imply that there may be opportunities for controlling and managing fine-grained NFV application state protection. Based on the NFV application usage of these states, their protection requirements would also differ. For our initial exploration, we use Snort as a driving example and analyze the feasibility and costs associated with protecting Snort states (TagNode) using OpenSGX.

2.2 Intel SGX

Intel SGX provides two main security features, namely, isolation and remote attestation. In this section, we introduce one of such features, *isolation* that S-NFV mainly utilizes to protect NFV application states. Then, we give a brief overview of OpenSGX [10], an open platform for Intel SGX, that S-NFV relies on to implement and evaluate the design of the proposed idea.

Isolation. SGX protects the confidentiality and integrity of an enclave's memory, where NFV applications are loaded and operate on. Enclave memory management is divided into two parts: address space allocation and memory commitment. The address space allocation is a specification of the range of logical addresses that the enclave may use. This range is called the ELRANGE. No actual resources are committed to this region. Memory commitment is the assignment of actual memory resources (as pages) within the allocated address space. This two-phase technique allows flexibility for enclaves to control their memory usage and adjust dynamically without overusing memory resources when enclave needs are low. Commitment adds physical pages to the enclave. An operating system may support separate allocate and commit operations. Further details of Intel SGX are found in the programming reference [6].

OpenSGX. It is an open platform that provides the hardware emulation of Intel SGX and an ecosystem such as operating system interfaces and user library for easy development of enclave programs. OpenSGX is implemented on top of QEMU's user-mode emulation by leveraging its binary translation. It provides a rich development environment, thereby allowing the research community to easily emulate a program running inside an enclave, without SGX-enabled hardware, licenses, and keys [10].

2.3 Threat Model

We assume that NFV applications are deployed by service providers in an untrusted datacenter environment, where the service providers do not have any control over the datacenter infrastructure. The

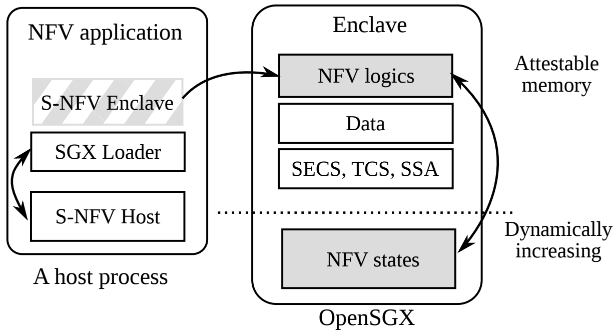

Figure 1: NFV Applications with SGX Framework

control infrastructure in such environment, i.e., the hypervisor or services running in privileged mode, is able to arbitrarily view and modify the NFV application states. As a result, an attacker who takes control over such environment can maliciously access and modify end-user states, such as profile pictures in CDN. The worst impact could be when such malicious access results in reconfiguration of firewall, routing policies and modifying tenant network address space, etc. In such setting, the memory region protected by Intel SGX is considered as the only trusted computing base (TCB). Also, the service providers are able to verify the evidence of a NFV application running in an SGX-equipped environment. Note, attackers can still launch attacks such as denial-of-service to break NFV application's functionality under the protection of SGX, but such kind of scenarios are out of scope in this work.

3. DESIGN

Given the strong threat model where the host environment is untrusted, S-NFV aims to provide a secure system design for NFV applications that can protect (i.e., isolate) their states from the potentially malicious underlying software stack.

3.1 S-NFV Software Architecture

In the S-NFV architecture, the original NFV application is split into two parts: S-NFV enclave and host. The S-NFV enclave, which contains the states and the state processing code, is decoupled from the original NFV application and protected by the SGX enclave, while the S-NFV host includes the rest of the processing as shown in Figure 1. To effectively leverage Intel SGX to ensure the integrity and confidentiality of states, there are two design requirements for the S-NFV enclave to guard against possible attacks [8]: 1) clear isolation against S-NFV and the underlying environment (e.g., operating system)—that is, the code and data of S-NFV enclave should not rely on memory outside the enclave; and 2) safe APIs to enable only a limited yet necessary set of interactions. S-NFV enclave also serves as the trusted anchor to support secure the S-NFV application bootstrap and establish the secure channel between S-NFV application and the service provider's trusted management node.

3.2 S-NFV Enclave Design

To extract a piece of code from the original program as a separated module and meet the aforementioned requirements are non-trivial. Here, we present details on the design of the S-NFV enclave in terms of each requirement.

Clear isolation. An NFV application is usually developed as a whole, which results in many dependencies among the code and data. For example, there could be several global variables used in different pieces of code. A single piece of code inside the enclave that relies on memory that resides outside the enclave can introduce additional attack surface and further break the security guarantee of SGX [12]. Therefore, to provide clear isolation, we need to carefully examine and decouple the target code (in Intel SGX) from the original application program.

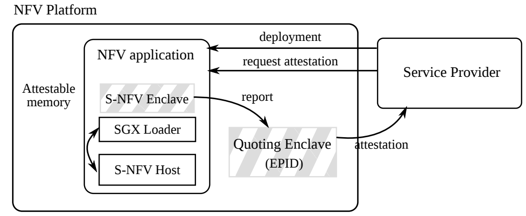

Figure 2: S-NFV Remote Attestation.

Safe APIs. Once we can ensure the clear isolation of S-NFV, the S-NFV enclave APIs are considered as the only attack surfaces exposed to the host environment. Similar to the concerns in clear isolation, the APIs also need to be carefully designed such that a complicated data structure that contains data or code pointers cannot be directly taken as arguments. Also, since attackers are able to observe all data to enter or leave the S-NFV enclave, we need to ensure that state-related data does not enter or leave the enclave in plain-text. Otherwise, the confidentiality of the states can be easily broken. Such cryptographic setting is done by the secure channel establishment explained in the following section.

3.3 Secure Launch

S-NFV applications leverage the SGX remote attestation feature to allow service providers to configure the S-NFV application and monitor S-NFV applications remotely. Based on the remote attestation primitive, as shown in Figure 2, S-NFV can further support:

Secure Bootstrap. During the S-NFV bootstrap, S-NFV enclave collects both memory states on different stages as well as hardware states to generate a valid measurement. The measurement is verified by the service provider through the remote attestation to determined whether the S-NFV application is correctly booted in the trusted environment. Also, the S-NFV is securely configured from the service provider.

Establishing Secure channel. After securely bootstrapping, the service provider can further establish a secure communication channel with S-NFV enclave based on remote attestation. One direct way is to use Diffie-Hellman key exchange, i.e., the key pair is generated within S-NFV enclave and the public key is sent along with remote attestation protocol.

The secure channel can guarantee the confidentiality of both the data sent to or sent from S-NFV enclave against the untrusted environment. The S-NFV configuration data is sent from the service provider to S-NFV enclave and the S-NFV application logs are sent from S-NFV enclave to the service provider.

4. IMPLEMENTATION

We use Snort, a candidate NFV application, as a proof-of-concept to demonstrate the S-NFV design on top of OpenSGX. In our implementation, we target at protecting the tag state (TagNode), which helps with packet processing once tag rules are specified in configuration. Tag contains the flow identifier (Src IP, Dest IP, Src Port, Dest Port and Protocol), with metrics like packet count, and logs the end-user IP address associated with flows.

According to S-NFV, we decouple Snort into Snort enclave (S-NFV enclave) and Snort host (S-NFV host). Tag operation, such as tag creation and tag rules querying, is put inside the Snort enclave, while the remaining part of the original Snort is taken as Snort host.

Isolating tag operation. To meet the isolation requirement in S-NFV enclave, we manually extract tag operation code from the original Snort and take out all links (e.g., global variables). To verify the isolation, we split the Snort host into two parts and run in

Component	LoC Changes	Total LoC
Snort Enclave	161	6,660
Snort Host (Server)	159	159
Snort Host (Client)	169	36,2330
Total	489	369,149

Table 2: The lines of code for Snort over S-NFV.

separate processes. Similar to a client-server model, the majority of Snort host code runs in the client process. The server process contains the Snort enclave and takes charge of handling tag operation requests from the client process. Whenever Snort host intends to execute a tag operation, the client side sends a request to the server side through socket. The server process then executes the tag operation through tag operation API and sends a result back to the client process. The result shows the server side LoC (Snort host server + Snort enclave) is only 6k, which is extremely small compared to the original Snort code base (360k), as shown in Table 2.

API for tag operations. Snort originally provides a set of tag operation APIs, as shown in Table 3. We examine the arguments used in each API and modify or remove the potentially risky arguments. For example, we modify the `Packet *`, which is a complicated data structure containing several code and data pointers, in `CheckTagList()` to `char *`, which is a pure buffer containing necessary packet information. Similarly, `OptTreeNode *` in `SetTags()` is replaced with `TagData *`. Note, all pointer arguments are passed through memory copying into enclave instead of passing by address similar to OpenSGX's trampoline.

API	Modification
`void InitTag(void)`	-
`void CleanupTag(void)`	-

`int CheckTagList(`
`char*,`
`Event*,`
`void*)`
	
	`Packet*` → `char*`
	
	`void**` → `void*`

`void SetTags(`
`char*,`
`TagData*,`
`-,`
`uint16_t)`
	
	`Packet*` → `char*`
	`OptTreeNode*` → `TagData*`
	`RuleTreeNode*` → `-`

`void TagCacheReset(void)`	-

Table 3: Tag operation API of Snort Enclave.

5. EVALUATION

To evaluate the S-NFV performance on real hardware, we further port the Snort enclave on Windows using SGX SDK [1]. We perform experiments on two main Tag operations, `CheckTagList()` and `SetTags()`, in both with and without SGX scenarios. In each experiment run, the target operation is repeatly executed 10000 times and the total execution time is measured.

Based on the experiment results shown in Table 4, `CheckTagList()` and `SetTags()` in the SGX-enabled case are 11.39 and 8.79 times slower than the SGX-disabled case, correspondingly. We observe that there seems to be a constant time overhead added to SGX-enabled implementation, which is likely introduced by the memory encrpytion and decrpytion performed by the memory encryption engine. Note that the performace overhead of tag operations do not represent the performace overhead of the whole Snort application, which depends the freqency of each operation during the whole Snort execution.

[1]Intel officially releases the Windows version SGX SDK that enables developers to write SGX applications on a SGX-equipped machine as of December 2015.

Tag Operation	SGX-enabled	SGX-disabled	Overhead
`CheckTagList()`	13.357 μs	1.172 μs	11.39x
`SetTags()`	13.426 μs	1.533 μs	8.79x

Table 4: The performace evaluation of Snort Enclave. Each Tag operation is repeatly executed 10000 times.

6. CONCLUSION

In this paper, we took a first step toward protecting the internal states of NFV applications against malicious hosts and buggy applications. By leveraging Intel SGX's isolation, we demonstrated the state protection of Snort's internal state (TagNode) and its state processing, by moving them into an enclave. We also perform the preliminary evaluation on state processing operations using real SGX hardware. In the future, we will extend this approach to a wider range of NFV applications, and analyze the impact of our approach (i.e., performance overheads and security trade-offs) by quantifying various aspects of the SGX hardware architecture.

7. ACKNOWLEDGMENT

We thank the anonymous reviewers for their helpful feedback, as well as our operations staff for their proofreading efforts. This research was supported by the NSF award DGE-1500084; by the ONR under grant N000141512162; by the DARPA Transparent Computing program under contract No. DARPA-15-15-TC-FP-006 and by the ETRI MSIP/IITP[B0101-15-0644].

References

[1] 3gpp policy and charging control architecture : Technical specification. http://www.qtc.jp/3GPP/Specs/23203-811.pdf.

[2] Nat64. http://http://ecdysis.viagenie.ca/whynat64.html.

[3] Haproxy: The reliable, high performance tcp/http load balancer. http://www.haproxy.org/.

[4] squid: Optimising web delivery. http://www.squid-cache.org/.

[5] Etsi nfv security specification, 2014. http://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/003/01.01.01_60/gs_NFV-SEC003v010101p.pdf.

[6] IntelÂő software guard extensions programming reference, 2014. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[7] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. In *Proceedings of the 11th Symposium on Operating Systems Design and Implementation (OSDI)*, Broomfield, Colorado, Oct. 2014.

[8] S. Checkoway and H. Shacham. Iago attacks: Why the system call api is a bad untrusted rpc interface. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Houston, TX, Mar. 2013.

[9] M. J. Freedman. Experiences with coralcdn: A five-year operational view. In *Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, Apr. 2010.

[10] P. Jain, S. Desai, S. Kim, M.-W. Shih, J. Lee, C. Choi, Y. Shin, T. Kim, B. B. Kang, and D. Han. OpenSGX: An Open Platform for SGX Research. In *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, Feb. 2016.

[11] M. Roesch. Snort lightweight intrusion detection for networks. In *Proceedings of LISA*, Seattle, Wa, Nov. 1999.

[12] R. Sinha, S. Rajamani, S. Seshia, and K. Vaswani. Moat: Verifying confidentiality of enclave programs. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, Denver, Colorado, Oct. 2015.