IEICE TRANS. INF. & SYST., VOL.E102–D, NO.9 SEPTEMBER 2019

LETTER

Cross-VM Cache Timing Attacks on Virtualized Network Functions

Youngjoo SHIN[†a)], ***Member***

SUMMARY Network function virtualization (NFV) achieves the flexibility of network service provisioning by using virtualization technology. However, NFV is exposed to a serious security threat known as cross-VM cache timing attacks. In this letter, we look into real security impacts on network virtualization. Specifically, we present two kinds of practical cache timing attacks on virtualized firewalls and routers. We also propose some countermeasures to mitigate such attacks on virtualized network functions.

key words: *cross-VM cache timing attack, network function virtualization, virtualized network function*

1. Introduction

Network function virtualization (NFV) enhances the flexibility of network service provisioning by leveraging virtualization technologies. In NFV architecture (Fig. 1), a variety of virtualized network functions (VNFs) such as a firewall, router and content delivery networks run as an instance of virtual machines (VMs) on commercial general-purpose x86 servers. As computing resources of the host are shared among multiple VMs, *cross-VM cache timing attacks* [1], [2] have emerged as a serious security threat to network virtualization. Although a wide range of security challenges with NFV have been extensively explored in the literature [3], [4], the effectiveness of such attacks has ever not been thoroughly studied.

In this letter, we look into real impacts on cross-VM cache timing attacks against virtualized network functions. Specifically, we realize two concrete attacks to the most common network functions, a firewall and a router.

Firewall, located at the perimeter of networks, aims to prevent ingression of malicious traffic from outside by means of packet filtering. Thus, reconnaissance on the filtering policy of the firewall is essential for an attacker to successfully infiltrate the inside network [5]. Our first attack shows how a co-located attacker can infer filtering rules of virtualized firewalls through a cache timing channel. We evaluate our reconnaissance technique on VyOS, a Linux-based versatile networking operating system.

As a second attack, we show that cache timing channel to virtualized BGP routers can be exploited to hijack benign network traffic. BGPsec, a standardized security extension of BGP, is supposed to be resistant against such traffic hijacking attacks. However, a cache timing attack to a BGPsec implementation is proved to weaken the desired security. Our attack is also evaluated on a BGP-SRx prototype, a reference implementation of BGPsec developed by NIST.

Finally, we propose some countermeasures to mitigate cross-VM cache timing attacks against virtualized network functions.

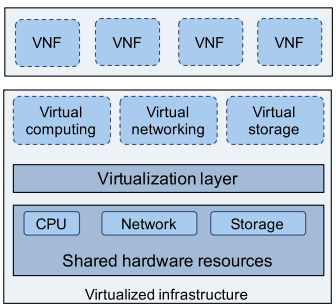


Fig. 1 NFV architectural framework

2. Cross-VM Cache Timing Attacks

Resource sharing between virtual machines results in cross-VM cache timing attacks such as Flush+Reload [1] and Prime+Probe attacks [2].

Flush+Reload attack utilizes memory deduplication technique in hypervisors, which enables sharing of physical pages across multiple VMs. This attack proceeds in three phases. During *Flush* phase, the attacker flushes the desired memory line (shared with the victim) from the entire cache hierarchy using the clflush instruction. Then, in *Wait* phase, the attacker waits for the victim to execute security sensitive operations. Finally, during *Reload* phase, the attacker reloads previously flushed memory line and measures the access time. If the victim has accessed the line during *Wait* phase, it will be reloaded from the cache, which results in lower reload time. If not, then the memory line still resides in the memory therefore resulting in higher reload time.

Flush+Reload attack cannot be used when the page sharing (i.e., memory deduplication) is not available in the hypervisor. In this case, Prime+Probe technique [2] is an alternative to Flush+Reload.

3. Threat Model

In our threat model, an attacker is a co-located VM that re-

Manuscript received March 11, 2019.
Manuscript publicized May 27, 2019.
[†]The author is with School of Computer and Information Engineering, Kwangwoon University, Seoul, Korea.
a) E-mail: yjshin@kw.ac.kr
DOI: 10.1587/transinf.2019EDL8048

Copyright © 2019 The Institute of Electronics, Information and Communication Engineers

sides in the same host with a victim VM, but has different security domain to the victim. The attacker has his own guest VM run on the host with privileged access to the guest OS. Therefore, she has a full-fledged ability to mount cross-VM cache timing attacks against the victim VM. We assume that a hypervisor on the host supports memory deduplication, which is necessary for Flush+Reload attacks. However, our attack is not confined to the specific cache timing technique.

4. Attacks on Virtualized Network Functions

4.1 Firewall Policy Reconnaissance

Firewalls are one of the most critical network appliances that protect inside networks by filtering out adversarial ingress packets. Firewall policy defines which packets are allowed to pass through, usually represented as a collection of filtering rules in the form of ACL (Access Control List). Each rule describes the shape of matched packets (i.e., source, destination address and port number, etc) and the relevant actions (i.e., accept, drop and reject, etc).

Firewall policy reconnaissance is a scanning technique to gather information on packet filtering rules of the target firewall for a preliminary step prior to the network intrusion. In all the proposed techniques [5], [6], attackers typically perform the reconnaissance by observing what kind of response packets are arrived for the specially crafted probe packets.

In this section, we present a new kind of reconnaissance technique for co-located attackers. Instead of observing arrived packets, our method infers filtering rules by learning the cache behavior on a host to the probe packets.

4.1.1 iptables

Most virtualized firewall products such as VyOS, IPFire, OPNSense and Smoothwall are constructed upon Linux kernels, within which a kernel component called iptables provides the main functionality of filtering packets. Hence, we focus on iptables as a target against our reconnaissance technique in this section.

The overall structure of iptables is depicted in Fig. 2. iptables defines a number of tables, among which a filter table is associated with the way packets are filtered. This table contains three chains of rules (i.e., FORWARD, INPUT and OUTPUT) for filtering packets. That is, filtering rules are applied when packets are sent to (from) local processes or forwarded through network interfaces.

iptables is implemented based on Netfilter, a hook-based framework for network operations. Each rule in the filter table basically consists of function hooks, which are invoked when the rule is being matched against packets and the relevant action is triggered for the matched packet.

4.1.2 Attacks

In order to infer filtering rules of the target, our reconnaissance technique exploits cross-VM cache timing attacks. As a probe packet passes through networking stack, it is examined against filtering rules in the filter table. If a matched rule is found, then a relevant action of the rule is triggered for the packet. As shown in Fig. 2, a number of kernel functions are called during this process. For instance, when a probe packet just arrives at a network interface, ip_recv() is called to handle the packet first. At the filter table, ipt_filter_hook() and ipt_do_table() are called sequentially and then the corresponding function for an action (e.g., reject_tg() for reject) is called.

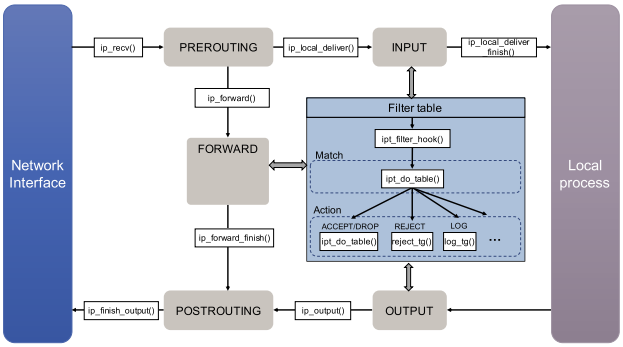


Fig. 2 Structure of iptables

The sequence of function calls leaves unique and distinguishable footprints on a memory and a cache, which can be observed by co-located attackers through a cache timing channel. Specifically, an attacker proceeds with policy reconnaissance as follows.

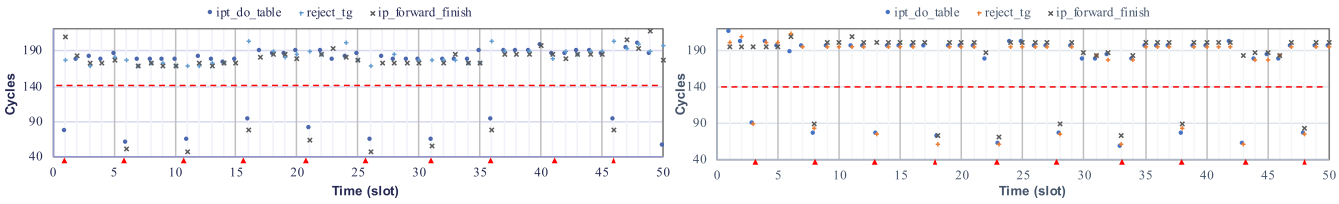
1. (*Flush* phase) chooses memory locations of kernel functions to be monitored and then flushes the desired cache lines on the cache.
2. (*Wait* phase) waits for a fixed time slot.
3. (*Reload* phase) reloads the cache lines and measures the memory access time.

We evaluate our reconnaissance technique by an experiment. The experiment was conducted on a host with Intel Xeon E5-2620v4 CPU and 32GB RAM running KVM (a Linux-based hypervisor). The host runs two guest VMs, one for a victim running VyOS 1.2 and the other for an attacker running an Ubuntu Linux 18.04. VyOS[†] is a Linux-based networking OS that has firewall features based on iptables. We chose three kernel functions on the victim for call tracing, ipt_do_table(), reject_tg() and ip_forward_finish().

For the experiment, two kinds of probe packets are used, one that is accepted by the victim's firewall policy and the other rejected. Those packets were sent repeatedly at every fixed interval (i.e., 500ms) from a remote host to the victim, during which a co-located attacker launches the cache timing attack with a time slot of 100ms.

Figure 3 shows the result of the experiment. Every dot below the threshold (i.e., 140 cycles) in the graph indicates that the corresponding function was called at that time. From the graph, we can clearly identify different types of

[†]https://vyos.io



(a) ACCEPT (b) REJECT

Fig. 3 Traces of kernel function calls (a dashed line represents a threshold and a triangle represents the time at which a probe packet was sent.)

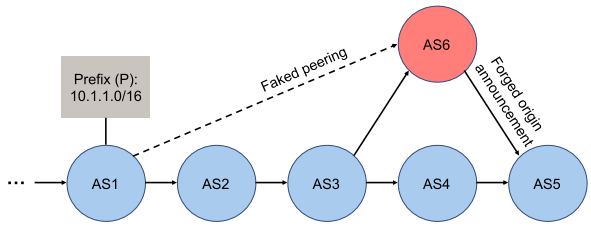


Fig. 4 Forged BGP updates

function call traces between those probe packets, which can be exploited to infer filtering rules.

It is noteworthy to mention that our reconnaissance technique has significant advantage over previous works [5], [6]. That is, our technique allows to conduct reconnaissance even with a crafted probe packet in which its source IP address is spoofed. This property provides two benefits. First, we can achieve evasiveness against intrusion detection since the source of probing is anonymized. Second, we can extend the search space to firewall policy by eliminating the restriction of source address on crafting probe packets.

4.2 Attacking BGPsec

In this section, we present a cross-VM cache timing attack on virtualized BGP routers with BGPsec extension.

4.2.1 BGPsec

BGP (Border Gateway Protocol) is a routing protocol that operates for an autonomous system (AS) on the top of the Internet hierarchy. Due to a lack of authentication and integrity for messages, BGP is vulnerable to traffic hijacking attacks. Suppose that an autonomous system AS1 originates an address prefix (P) 10.1.1.0/16 as illustrated in Fig. 4. In order to announce the prefix, a BGP router in AS1 sends an update message containing (P, AS1) to its peer in AS2, which in turn adds its AS number to the message and forwards it to the next peer in A3, and so on. This way, a router in AS5 finally receives the message containing (P,AS1,AS2, ..., AS4), which becomes an AS path to the prefix P. However, if a malicious router in AS6 sends a forged update message (P, AS1, AS6) to AS5, there is no way to check whether the message is valid. Hence, all traffic destined for P will be sent to AS6, not to AS4, since the rogue AS path via AS6 is shorter than the benign AS path.

BGPsec is a security extension of BGP defined in RFC 8205, published in 2017, which aims to prevent forgery of BGP update messages. In BGPsec, each router has a pair of public and private key for a signature algorithm. All messages are signed with a private key prior to being forwarded to peers, and the integrity of received messages are verified with a sender's public key. Thus, forgery of messages by AS6 in the above example can be prevented. BGPsec uses ECDSA (Elliptic-Curve Digital Signature Algorithm) with NIST P-256 curve as its signature algorithm [7].

4.2.2 Attacks

Suppose that a victim is a virtualized external BGP (eBGP) router in a transit AS, equipped with BGPsec functionality. We show a cross-VM cache timing attack against a BGPsec implementation of the victim. In our attack, the goal of attackers is to extract an ECDSA private key of the co-located BGP router.

Our attack is realized on a BGP-SRx prototype[†], a reference implementation of BGPsec developed by NIST. This prototype fully relies on an OpenSSL library for the implementation of cryptographic operations such as ECDSA signing and verification of BGP update messages.

OpenSSL is a well-known cryptographic library, thus has been targeted by a variety of cache timing attacks for a long time. Recently, García et al. discovered a cache timing attack against ECDSA implementation with NIST P-256 curve in an OpenSSL library [8]. When a victim is executing ECDSA signing operations, this attack tries to capture ephemeral keys by using Flush+Reload technique. Once a sufficient number of ephemeral keys (i.e., as few as 50 keys) are gathered through the side-channel, then a full private key of the victim can be computed from the obtained information with lattice methods [8].

We make use of García et al.'s technique for realizing our attack on the BGP-SRx implementation. Let us assume that an attacker is able to eavesdrop BGP update messages containing BGPsec_PATH attribute sent from the victim router. This attribute includes an AS path as well as its signature generated by the victim in unencrypted form.

Our attack proceeds as follows. Initially, a co-located attacker chooses memory locations of secret-dependent branch instructions in ECDSA routines (i.e., BN_rshift1() and BN_usub()) of an OpenSSL library. Then, she keeps

[†]https://www.nist.gov/services-resources/software/bgp-secure-routing-extension-bgp-srx-prototype

Flush+Reload probing at these locations while waiting for a new BGP update message being received. On the receipt of the message, the victim router should forward it to a next hop after signing the message, during which the attacker can obtain an ephemeral key of the signature through the cache timing channel. This way, she repeats collecting ephemeral keys as well as their signature captured over the network until sufficient information is gathered to recover a signing private key.

Note that our attack considers passive adversary model. This is because according to BGPsec protocol, only an authenticated BGP peer is allowed to send valid BGP update messages to the victim. Hence, the attacker should wait for the victim to receive update messages from its peer so that signing operations are triggered. Recent surveys report that an eBGP router in real ASes on average receives 170,000 update messages per day from its peers[†]. Therefore, the attacker is able to collect a sufficient number of ephemeral keys (i.e., at least 50 keys) within just 30 seconds to mount García et al.'s attack.

We evaluate our BGPsec attack on the same experimental environment as in the previous section except that a victim guest VM runs a BGP-SRx prototype. For the experiment, we set up an another host located remotely over the network, which plays a role of BGP peers. It sends BGP update messages to the victim at the rate of 120 messages per minute, which conforms to the rate in real ASes.

As a result of the experiment, we successfully obtained execution traces for each ephemeral key through the cache timing channel but with 30% loss of the key on average due to system noise. The loss of information was supplemented by increasing the number of ephemeral keys to be collected. With at most 80 (partial) ephemeral keys, we completely recovered the ECDSA private key of the victim.

5. Countermeasures

We introduce countermeasures to mitigate cross-VM cache timing attacks against virtualized network functions.

Resource isolation. The root cause that enables cache timing attacks is that hardware resources such as memory page and cache are shared between a victim and a co-located attacker. Page sharing across VMs (i.e., memory deduplication), which inherently allows Flush+Reload attacks, is one of main features in most hypervisors (e.g., Transparent Page Sharing in VMware vSphere and Kernel Same-page Merging in KVM). Thus, it should be disabled at the hypervisor to prevent cache timing attacks.

In order to achieve strict cache isolation, we propose to use hardware-assisted solutions. For instance, Intel Cache Allocation Technology (CAT), equipped in Xeon processors, divides L3 cache and allocates a distinct portion of the cache to each VM. By using CAT, we can make cache timing attacks against virtualized network functions infeasible.

Constant-time implementation Resource isolation-based solutions described above requires modification of low-level system components such as a hypervisor, which becomes major obstacle to be adopted with ease. Constant-time technique is another option for application-level countermeasures. It is an implementation technique that has algorithms (especially, cryptographic algorithms) run without incurring any secret-dependent memory access patterns at the granularity of a cache line. Scatter-gather implementation of a RSA algorithm in recent versions of an OpenSSL library is an instance of constant-time technique. Virtualized network functions equipped with cryptographic algorithms (e.g., BGPsec) can achieve its security against cache timing attacks by employing constant-time implementation technique.

[†]https://blog.apnic.net/2018/01/10/bgp-in-2017/

6. Conclusion

Virtualization technology provides the flexibility of network service provisioning in NFV, but exposes a serious security threat known as cross-VM cache timing attacks. In this letter, we investigated real security impacts by presenting practical attacks on the most common virtualized network functions, a firewall and a router. In order to mitigate such attacks, we also proposed several countermeasures through resource isolation and the use of constant-time technique.

Acknowledgments

This research has been conducted by the Research Grant of Kwangwoon University in 2019 and supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No.2017R1C1B505045).

References

[1] Y. Yarom and K. Falkner, "Flush + reload: A high resolution, low noise, L3 cache side-channel attack," Proceedings of the 23th USENIX Security Symposium, pp.719–732, 2014.
[2] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R.B. Lee, "Last-level cache side-channel attacks are practical," Proceedings of 2015 IEEE Symposium on Security and Privacy, pp.605–622, 2015.
[3] M. Pattaranantakul, R. He, Q. Song, Z. Zhang, and A. Meddahi, "NFV Security Survey: From Use Case Driven Threat Analysis to State-of-the-Art Countermeasures," IEEE Communications Surveys and Tutorials, vol.20, no.4, pp.3330–3368, 2018.
[4] M. Daghmehchi Firoozjaei, J.P. Jeong, H. Ko, and H. Kim, "Security challenges with network functions virtualization," Future Generation Computer Systems, vol.67, pp.315–324, 2017.
[5] M.Q. Ali, E. Al-Shaer, and T. Samak, "Firewall policy reconnaissance: Techniques and analysis," IEEE Transactions on Information Forensics and Security, vol.9, no.2, pp.296–308, 2014.
[6] A.R. Khakpour, J.W. Hulst, Z. Ge, A.X. Liu, D. Pei, and J. Wang, "Firewall fingerprinting," The 31st Annual IEEE International Conference on Computer Communications (IEEE INFOCOM 2012), pp.1728–1736, 2012.
[7] S. Turner and O. Borchert, "BGPsec Algorithms, Key Formats, and Signature Formats," IETF RFC 8208, pp.1–19, 2017.
[8] C.P. García and B.B. Brumley, "Constant-time callees with variable-time callers," Proceedings of the 26th USENIX Security Symposium, pp.83–98, 2017.