All Your VMs are Disconnected: Attacking Hardware Virtualized Network

Zhe Zhou[1,3], Zhou Li[2], Kehuan Zhang[1,3]
Department of Information Engineering, Chinese University of Hong Kong[1]
ACM Member[2]
CUHK Shenzhen Research Institute[3]
{zz113, khzhang}@ie.cuhk.edu.hk[1], lzcarl@gmail.com[2]

ABSTRACT

Single Root I/O Virtualization (SRIOV) allows one physical device to be used by multiple virtual machines simultaneously without the mediation from the hypervisor. Such technique significantly decreases the overhead of I/O virtualization. But according to our latest findings, in the meantime, it introduces a high-risk security issue that enables an adversary-controlled VM to cut off the connectivity of the host machine, given the limited filtering capabilities provided by the SRIOV devices.

As showcase, we demonstrate two attacks against SRIOV NIC by exploiting a vulnerability in the standard network management protocol, OAM. The vulnerability surfaces because SRIOV NICs treat the packets passing through OAM as data-plane packets and allow untrusted VMs to send and receive these packets on behalf of the host. By examining several off-the-shelf SRIOV NICs and switches, we show such attack can easily turn off the network connection within a short period of time. In the end, we propose a defense mechanism which runs on the existing hardware and can be readily deployed.

Keywords

SRIOV; Virtualization; OAM

1. INTRODUCTION

Virtualization techniques are important to today's computing infrastructure like cloud. They enable hardware resources to be shared among different users through running heterogeneous virtual machines (VMs). They also provide secure computing environment to users based on the strong guarantees of isolation. In the beginning, virtualization suffered from significant runtime performance penalty. As one main cause, a lot of computation has to be taken in order to emulate the access to I/O devices, which becomes the major bottleneck. To address this issue, a suite of I/O virtualization techniques were proposed to reduce such unnecessary overhead.

CODASPY'17, March 22-24, 2017, Scottsdale, AZ, USA
© 2017 ACM. ISBN 978-1-4503-4523-1/17/03…$15.00
DOI: http://dx.doi.org/10.1145/3029806.3029810

The initial I/O virtualization design required the participation of the hypervisor who emulates I/O devices and acts as a bridge between VMs and I/O devices [27]. Such design is called para-virtualization and unnecessarily consumes a lot of extra computing resources due to the replication of numerous I/O operations. In recent years, a new technique, Single Root I/O Virtualization (SRIOV) was developed and is quickly gaining tractions. This technique can partition the hardware into multiple compartments and *directly* assign each compartment to a VM [11, 26, 31, 21]. As such, the hypervisor is removed from the I/O process path and most I/O operations cab achieve nearly bare-metal performance [20, 18, 15, 28]. According to VMware's recent research, the SRIOV passthrough mode NIC virtualization achieves 99.8% throughput of a native machine, while less than 50% for para-virtualization. In terms of latency, in average, SRIOV passthrough mode is only 13% higher than a native machine while the number is 107.7% for the para-virtualized mode [30].

Meanwhile, the security of SRIOV was studied since the very beginning. In principle, an SRIOV device should not be directly configured by a VM, as the VM can be easily controlled by an attacker. Therefore, the functions provided by the device are separated into two groups under control-plane and data-plane, and by assigning only data-plane functions to the VM. Control-plane functions are assigned to the hypervisor and all requests from the VM regarding configurations are firstly routed to the hypervisor for sanitization. Under this setting, the security requirements seem to be satisfied.

However, such design is problematic under today's complicated network environment when numerous network protocols have to be supported. In particular, we found one network protocol, Ethernet OAM (Operations, Administration, Maintenance), could cause severe disruption to the network environment if abused by the adversary. To launch the attack, the adversary controlling a VM simply needs to send **3 OAM packets** to a multicast address. Then, the port connected by the host will be closed, causing other VMs and the hypervisor residing on the same host all disconnected. This kind of attack could lead to much worse consequence when numerous VMs are controlled by attackers.

This vulnerability is a result of the combination of multiple factors. First, the switch can be configured to automatically turn off the port when receiving an error signal from the entity at the other end. This design is reasonable when the other end is a single host but questionable when multiple untrusted VMs run on the end host. Second, **the**

OAM packet is considered as a data-plane packet as there is no command in the packet content, which bypasses checks performed by the hypervisor. Though this issue is not limited to SRIOV devices, the design principle of the SRIOV (data transparent to hypervisor) **makes it very difficult to fix**. Third, the existing SRIOV networking devices have rather limited **hardware-based** filtering capabilities regarding the packets sent to multicast addresses.

To demonstrate the severity of this vulnerability, we showed a new attack against SRIOV NIC (Network Interface Card) by abusing a VM on the same host. We overcame several obstacles, including adding the VM's address into the recipient list of multicast packets and finishing the handshake process of OAM, and showed that it is possible to achieve the goal of disconnecting the host. We found that the attack can immediately take effect but recovering from the failure requires considerable manual efforts. Then, we developed a defense mechanism based on our observation that the legitimate VM does not need to initiate OAM conversations. The defense requires a small modification to the NIC driver to prevent VM from adding itself to the recipient list of OAM packet. As a result, the attack is thwarted since the adversary cannot finish the handshake process, a required step before sending valid OAM messages to the switch.

Next, we explored the possibilities of bypassing our basic defense and it turns out the adversary is able to completely skip the handshake process. By thoroughly evaluating the OAM protocol, we found that the possible values indicating the switch configuration are within limited range. In fact, learning the configuration can be done without going through the handshake process. By repeatedly sending OAM packets with possible configuration values, the attack could still succeed. As shown in our evaluation, the switch does not check the contents of the handshake packets, which enables attackers to freely set up OAM connections and launch the attack without receiving any packet. Considering its severity, we reported the issue to the switch manufacture. The manufacture confirmed the vulnerability and promised to fix it soon. In addition, as noted by the manufacture, **the relevant RFC standard does not require handshake validation** and we speculate the issue could exist in other switches as well.

At last, we explored the existing implementations of all entities involved and found that an unused field in OAM packets can be retrofitted to include a secret value that is hard to be guessed by the adversary. We proposed a mitigation approach that only requires small changes to the switch firmware and the test result showed that it is quite effective.

Contributions. The contributions of this work are summarized below:

- We discovered a vulnerability on SRIOV NIC in handling the OAM protocol and the multicast channel. Such vulnerability can result in a severe consequence where untrusted VM can cut off the physical machine's entire network connectivity.

- Based on this finding, we successfully launched attacks in a virtualized environment, leveraging a flaw of the switches shipped by a well-known manufacturer. In fact, the cause of the flaw comes from the rough specification by the RFC OAM standard.

- We proposed a method to defend against the attack without any modification to the physical layer of the existing hardware. The method only requires modification to the switch firmwares and NIC drivers. We also set up a emulation environment to validate the defense method.

Roadmap. Section. 2 introduces two main technologies that are related to this work. Section. 3 describes how attacker can launch the attack. Section. 4 shows a simple defense that only modifies the NIC driver which is later turned out to be not enough. Section. 5 describes an updated attack method that can be used to counter the defense. Section. 6 analyzes the root reason of the vulnerability. Section. 7 includes a practical defense that can truly prevent attackers from launching this kind of attack. Section. 8 describes related works and Section. 9 concludes this paper.

2. BACKGROUND

Our study reveals a critical vulnerability underlying SRIOV devices when handling Ethernet OAM (Operations, Administration, Maintenance) packets. In this section, we briefly describe these two technologies.

2.1 SRIOV

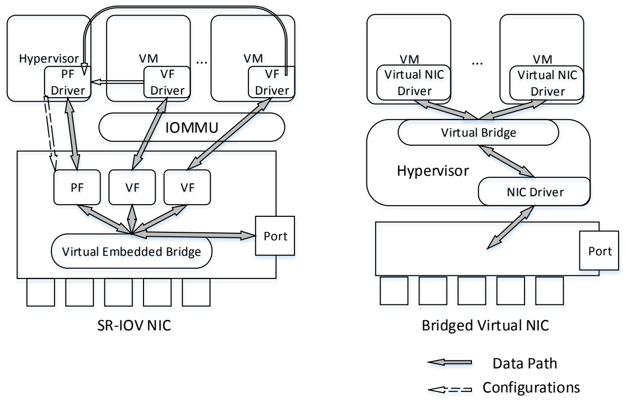

Figure 1: SRIOV NIC workflow.

Design. SRIOV was proposed by the PCI Special Interest Group (PCI-SIG) to allow a PCIe device to appear as multiple separate physical PCIe devices and be manipulated by VMs directly. Under the traditional I/O virtualization model, a hypervisor mediates all the flows between VMs and I/O devices. The flows have to be multiplexed before being forwarded to hardware. The right part of Figure 1 shows an example of the bridged Virtual NIC. In this setting, the hypervisor must virtualize software NICs (including interfaces and ring buffers) for an VM to access network and set up virtualized bridge to connect VM's virtualized NICs to the physical port. Extra CPU and memory resources are consumed by hypervisor to forward the packets to the correct destinations (software-based forwarding). The penalty exacerbates when a huge volume of data has to be processed, e.g., in 10G Ethernet network.

To the contrary, SRIOV directly assigns the logical accessing port of the hardware to VM and makes the whole processing flow *transparent* to hypervisor, achieving nearly bare-metal performance. The left part of Figure 1 illustrates the processing flow through SRIOV capable NIC [1]. Such NIC integrates an efficient hardware unit called *Virtual Embedded Bridge (VEB)* [17, 12, 13]. Flow forwarding is done by VEB and no extra performance penalty is incurred by hypervisor. To notice, VEB is not required by every SRIOV device. But in this NIC case, it is a must-have because an NIC has only one physical port and VEB enables the port sharing among different VMs.

To enable the access from the upper levels, SRIOV device offers *Physical Function (PF)* and *Virtual Function (VF)*. Each device is required to provide at least one PF and multiple VFs.

A PF is a fully featured PCIe function supported by a hardware driver, via which its owner can fully control the hardware device including configure its setting (control plane) as well as performing I/O operations (data plane).

A VF has only a subset of features of PCIe function and is supported by a driver different from PF's. It is only allowed to exchange data with hardware device (data-plane only). VF is designed to be invoked by VMs and a hardware unit IOMMU (Input output memory management unit) is built to map a VM to a specific VF.

Take SRIOV NIC as an example. The interface controller usually provides 63 VFs to serve up to 63 VMs simultaneously. NIC receives all the packets from VF, PF and devices outside of the machine, and then passes them to its VEB. VEB needs to route the packets to their destination, as a result it forwards them to either PF/VF of the same machine or sends them out to other machines through network cable.

Many devices nowadays are SRIOV capable, like storage controller, GPU and NIC. These devices have already been widely deployed by cloud service providers (e.g., Amazon EC2 HPC), enterprise data centers and high performance computers (HPC) [2, 3].

Security model. A major security requirement for SRIOV is to protect the hardware from being tampered by untrusted parties, i.e., VMs controlled by attackers, on the host machine. In the traditional I/O virtualization model, hypervisor is leveraged as the guard to apply filtering rules on the virtual bridge. For example, hypervisor can use `ebtables` to set rules for filtering packets from VM based on the protocol, source MAC address, or destination MAC address. In the SRIOV model, hypervisor is removed from the path, so, as a replacing security mechanism, SRIOV device separates its functionalities into data-plane and control-plane, and group them into PF and VF. Then, it delegates PFs to trusted party, usually hypervisor, and VFs to untrusted parties, usually VMs. If the owner of VF intends to change the hardware configuration, it has to issue a request to the owner of PF (i.e., hypervisor). If the request is approved, PF will be used by its owner to fulfill the requests.

Under this setting, adversary who controls VM can only exchange data through the hardware and configuring hardware is out of her reach. This security model appears to be sound but our attack (see Section 3 and Section 5) shows that such design is not perfect. The problem emerges due to the ill-conceived integration of old management protocol and new virtualized environment, and further complicates due to the lack of finer-grained filtering capabilities on SRIOV devices.

[1]We use the term SRIOV NIC afterwards for brevity.

2.2 Ethernet OAM

To help the network operator diagnose the network, IEEE 802 working group established IEEE 802.3ah (Ethernet in the First Mile, EFM) protocol which supports link layer Ethernet management (Ethernet OAM) and included it into the overall standard 802.3-2008. To date, Ethernet OAM was implemented by the majority of the network device vendors. The features required by OAM include link discovery and monitoring, remote fault detection and remote loopback [8]. Below, we describe the data format of OAM packet and the handshake process within the protocol.

OAM Data Packet. As defined by the EFM protocol family, OAM data packets are exchanged under basic 802.3 slow protocol frames, which are also called *OAMPDU (OAM Protocol Data Units)*. By default, OAMPDUs are transmitted at limited speed, which is up to 10 packets per second. Another restriction on OAMPDU is that it is designed to be **transmitted between two endpoints of a single link**. Even when the receiving endpoint does not support OAM, **OAMPDU cannot be relayed to other entities**. To communicate across links, support from higher-layer applications is required.

Length	Field	Value
6	Destination Address Slow Protocol Address	01-80-c2-00-00-02
6	Source MAC address	any
2	Type	88-09
1	Sub-type	0x03
2	Flags	any
1	Code	any
42-1496	Payload	any
4	Check Sum	any

Table 1: OAMPDU format.

Table 1 elaborates the format of OAMPDU. When the endhost attempts to send OAMPDUs, the destination can only be the device at the other end of the cable, e.g., switch. Therefore, the Rx (Receive) address is a constant value (first row in Table 1) and defined as a *multicast* address, while the Tx (Transmit) address can be any valid value (second row in Table 1). Such setting disables packet relays as required by OAM protocol. In addition, every packet of OAMPDU carries 15 bits of flags representing entity's real-time status, including the OAM connection status (4 bits, 2 bits for local status and the other 2 bits for remote status) and emergency link events (4 bits with each bit referring to `Link Fault`, `Dying Gasp`, `Critical Event` and `Link Loss`).

There are 3 types of OAMPDU defined by EFM: *information OAMPDU*, *event notification OAMPDU* and *loopback control OAMPDU*. Information OAMPDU is used to exchange information between OAM entities, including the process of handshaking and error reporting. In this work, we discovered the vulnerability of SRIOV device when handling information OAMPDU and we introduce its structure here (also illustrated in Figure 2).

For an information OAMPDU, the data field in the payload section contains one or more TLV (Type-length-value), which is the container for entity configurations. The TLV can be used to describe local entity, remote entity or be filled with customized information. Each Local/Remote TLV is 16 bytes long and composed of 9 fields.

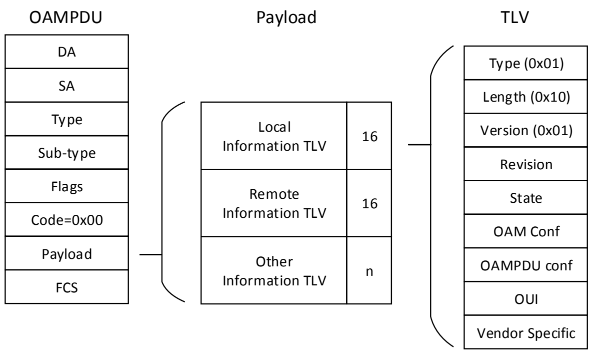

Figure 2: Information OAMPDU format.

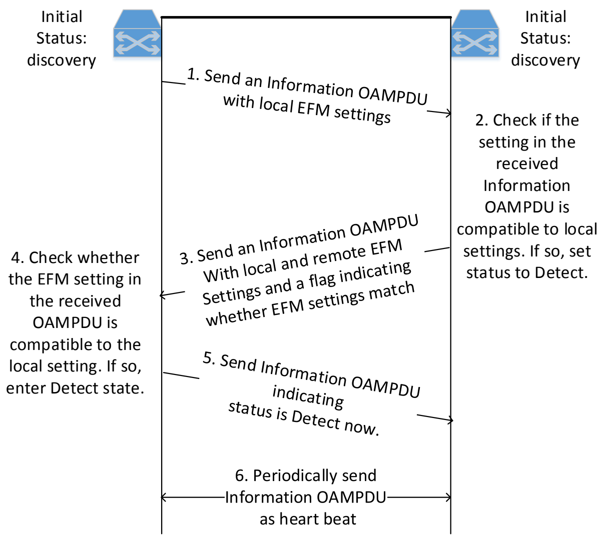
Initial Status: discovery
Initial Status: discovery
1. Send an Information OAMPDU with local EFM settings
2. Check if the setting in the received Information OAMPDU is compatible to local settings. If so, set status to Detect.
3. Send an Information OAMPDU With local and remote EFM Settings and a flag indicating whether EFM settings match
4. Check whether the EFM setting in the received OAMPDU is compatible to the local setting. If so, enter Detect state.
5. Send Information OAMPDU indicating status is Detect now.
6. Periodically send Information OAMPDU as heart beat

Figure 3: OAM handshake flow chart.

Handshake flow of information OAMPDU. Essentially, the handshake process has to be successfully completed before actual conversation of OAM. Figure 3 illustrates this process and the details are elaborated as follows:

1. In the beginning, the entities of the both ends set their status to `Discovery`. Then, either one of them in active mode continuously broadcasts its own configuration settings through OAMPDUs to indicate the intention of setting up connection. Only a local information TLV is contained in OAMPDU.

2. When the remote entity successfully receives the packet, it checks if the received configurations are compatible with its pre-defined format. If the check is passed, it replies to the sender with an Information OAMPDU that carries flags indicating that it is satisfied with the configuration and changes the connection status to `Detect`. The Information OAMPDU embeds both its own information TLV (in local TLV) and the received information (in remote TLV) into the payload.

3. After the responding OAMPDU reaches the sender, the embedded configuration information will be checked as well in the same compatibility checking process. If compatibility requirement is fulfilled and the attached configuration of the remote entity conforms to that of the first packet issued by the sender, the sender's status will be switched to `Detect`, announcing that the connection is successfully set up. Following that, both entities will periodically send Information OAMPDUs as heart-beat signal.

All other OAM functionalities are enabled after the connection is set up, like link quality measurement. The most commonly used feature is error isolation, through which the remote entity turns the interface status to `Error Down` if the entity detects the link failure by itself or receives the error report sent through that interface. This feature is very helpful to accelerate the routing converge of the higher layers and avoid the problematic link being clogged by the network data. However, this feature could be turned into an attack vector if abused by the adversary, i.e., to disconnect other legitimate hosts sharing the same network, and our attack demonstrates this is feasible.

3. ATTACK

In this section, we elaborate how we leverage the OAM protocol to attack the SRIOV device and disconnect all VMs and the hypervisor linked to it. We first describe the adversary model and attacker's motivation. Then, we show the details of the attack and evaluate the attack effectiveness.

3.1 Adversary Model

We assume the adversary is targeting the network infrastructure of a cloud service, which could be either public cloud (like Amazon EC2) or private cloud deployed within an organization. Attacker's goal here is to disrupt the service operation. The computer of the cloud service runs multiple VMs connected to a switch. The NIC of the computer is SRIOV capable and the feature is already enabled. Each VM on the machine is assigned with an NIC VF to allow direct access to network consisting of a switch that supports OAM protocol. .

Comparing to controlling hypervisor or switch for the attack purpose, obtaining access to the VMs is a more attainable goal. In public cloud, the adversary could rent a VM and issue malfeasant configuration requests. In private cloud, attacker could exploit the vulnerability of a less protected VM and take the full control. As described later in Section 3, by just issuing carefully crafted OAMPDU from an attacker-controlled VM to SRIOV NIC of the host, all the network connections between the VMs and the hypervisor to the switch will be terminated, causing network failures hard to recover. Given that a cloud host-machine usually runs many VMs at the same time, the damage to the users or customers could be tremendous. Since SRIOV devices are widely deployed by cloud services and these devices are usually off-the-shelf, the number of cloud services under threat is potentially very large. In addition, our attack does not break the isolation mechanisms of VM-to-VM and VM-to-hypervisor, making prevention very difficult.

We also assume the administrator follows the security guidance of SRIOV, under which PF is controlled by benign hypervisor and VF is delegated to untrusted VMs. While the attacker could request hypervisor to adjust settings that are exposed to her, hypervisor can easily reject such illegal requests by enforcing the standard detection logic. In contrast, our attack only leverages VF to communicate with the host NIC and all the checks from the hypervisor are bypassed.

A recent work by Smolyar *et al.* has shown that SRIOV-capable NIC is vulnerable to DoS attack launched by VM [24]. By sending illegal flow control packets, the performance of the network is significantly dropped, e.g., 250% increase of network latency. Comparing to that attack, ours could cause much more severe consequence - disconnecting the entire network connections.

3.2 Attack Method

Attack Overview. Our attack exploits the error isolation feature provided by OAM protocol. As described in Section 2.2, an entity could send an information OAMPDU packet to the linked switch to indicate if there is a link fault. Receiving this signal, the switch could cut off the connection of the host machine. This feature is innocuous when the sender is a hypervisor: even if the hypervisor is controlled by attacker, sending this packet only disconnects itself. The attack cannot be easily launched if virtualization is realized through software bridges, because hypervisor can easily filter out OAM packets. However, damage could be caused when it is sent by an attacker-controlled VM, as all other VMs and their hypervisor sharing the same NIC would be disconnected while the hypervisor is agnostic.

While it is assumed that the sender reports only *after* detecting the link error, such assumption is not guaranteed. VM could *fake* a link fault message and there is no mechanism in place to check its authenticity. Though error reporting is supposed to be the responsibility of the hypervisor, we suspected VM could undertake this task as well. We tested such hypothesis by sending the OAMPDU packet through VM and discovered that VEB treats the packet as a normal multicast packet (data-plane packet) and forwards it to the remote switch. To complete the error reporting process, a handshake has to be completed ahead. Again, this is achievable by VM: VM can register its address as the destination address of the OAM protocol and receive all the OAMPDUs sent by the switch. Below, we elaborate how the attack can be launched.

Attack Implementation. Our attack takes two steps: the first step is to set up an OAM connection and the second step is to send the heart beat packet with the fault flag toggled.

Attacker could set up the OAM connection using *raw socket* and inject crafted packets that accord to the OAM handshaking format. By setting the destination address to a multicast address *01:80:C2:00:00:02* (see Section 2.2), the packet will be sent out by the NIC of the host and reach the switch. However, receiving the packet and completing the handshake process are not trivial for VM: by default, the responding packet is also sent to the same multicast address instead of the Tx address provided by the VM and then transmitted to the NICs of the linked hosts. It turns out the VEB of host NIC normally would refrain from forwarding multicast packet to VMs due to two performance concerns: 1)It consumes a lot of computations unnecessarily as not every VM needs multicast packet. 2) Handling those multicast packets not belonging to the VM costs the VM considerable resources.

Here, we let the VM configure *Multicast Table Array (MTA)* of the host NIC to receive the reply packets. For an Intel SRIOV NIC, when it receives a packet towards a multicast address, it will look up MTA which logs a list of destination addresses and forward the packet to the VMs whose addresses are enlisted. Though MTA has to be updated by the hypervisor and cannot be configured by VM directly, it turns out that VF driver of Intel SRIOV NIC provides API for a VM to ask hypervisor for updating MTA. By default, hypervisor will accept the request and fulfill it automatically without any actions from the administrator. In particular, the attacker could run `IP maddr add` command on the VM to append the address into MTA. After that, the malicious VM is able to receive OAMPDUs and complete the handshake process.

The latter step is rather straightforward. When the attacker wants to cut off the network of the physical machine, she only needs to set the link fault flag in the Information OAMPDU to 1, which represents "critical errors". The switch will shut down the corresponding connection once this OAMPDU is received.

3.3 Evaluation

To test the effectiveness of the attack, we set up a testbed consisting of a SRIOV capable server and an OAM capable switch. We created two VMs supervised by KVM hypervisor and all of them run Ubuntu 14.04.3 LTS OS. Table 2 shows the detailed information of the platform.

We assume one VM (VM_1) has been controlled by the attacker while the other is benign (VM_2). Both of them have been assigned with NIC VF. Meanwhile, VM_1, VM_2 and the host OS are all configured to use static IP addresses belonging to the same class-C IP subnet. The physical NIC port of the machine is connected to a HUAWEI S3328TP-EI switch that connects to a LAN with Internet access through another port. The OAM of the switch is turned on and the fault isolation is enabled, which means the switch will terminate all the connections for the port and turn the status of the port to `Error Down` if an error is received or perceived from the port.

As for the experiment result, after VM_1 set up OAM connection and sent an Information OAMPDU carrying a link fault flag, we found that the switch cut the connection as expected, immediately after the OAMPDU is received, in less than 1 second. Consequently, VM_2 can no longer access the Internet, together with the hypervisor. From the console of the switch, we observed that the status of the port was turned to `Error Down`, and all the services associated with the port were terminated. This result clearly proves our attack is effective and easy to carry out.

We also evaluated the difficulty for an technician to restore the network from such failures.

First, it turns out without the admin privilege, the switch cannot recover the network activities by itself. We detached the cable from the port and attached it to the port again after 30 seconds. The status of the port was not changed and the connection was still blocked. According to the configuration guide of the tested switch [1], traffic will not be resumed even if the faulty link recovers, because the error status of the interface is not reset.

Second, we also tested whether the administrator could use the network manager account to reset the connection. The link status came back to normal after we restarted the interface. However, the restarting process should be conducted carefully, as the guide warns network managers that

Table 2: Platform Information.

	Host	VM
CPU	Intel Xeon E5-2620 V3	Intel Xeon E5-2620 V3 (2 cores)
Memory	16G DDR4	8G
Mother board	ASUS Z10PE-D16	/
NIC	Dual Port Intel Ethernet Controller i350-AM2	Intel Ethernet Controller i350 VF
OS	Ubuntu 14.04.3 LTS	Ubuntu 14.04.3 LTS
NIC driver	igb 5.3.2	igbvf 2.3.5
Hypervisor	KVM + virt-manager	/

they should manually check link quality after switching back the traffic. Therefore, the network manager still has to manually examine the network status or even recover by herself, a very time-consuming task.

The consequence can be much more severe if the network manager configured *EFM Association* [1]. When the switch receives the Information OAMPDU with link fault, firstly it invokes Layer-2 operations (e.g., turn down the interface.) to the port according to the configurations. Besides, it may also trigger other modules or other EFM entities by broadcasting the error. The error broadcast could incur further damage to the network. For example, it could trigger *BFD (Bidirectional Forwarding Detection)* module to change the routing behavior of a upper layer router. It may also trigger another switch to shut down a port, etc. Such snowball effect would cause more machines to be disconnected.

4. BASIC DEFENSE SCHEME

To protect the cloud service from being disrupted by this attack, we propose a defense mechanism which can be easily implemented without any change to the hardware and network infrastructure. We believe the capabilities of the VM in configuring OAM should be controlled. In the meantime, all other communication channels from VMs should not be affected. Below, we elaborate our defense scheme and the evaluation result.

4.1 Method

Shutting down the OAM channel of a VM while ensuring other data communication channels run normally are in fact not easy. Hypervisor has no visibility and control over the data sent by VM, when VM accesses SRIOV NIC. One intuitive idea is to grant hypervisor the ability to block certain data flows between VM and SRIOV NIC, in particular through manipulating the functional registers provided by the NIC, which however requires hardware modification and cannot be readily deployed.

As described in Section 3.2, VM has to enlist the OAMPDU multicast address in MTA of SRIOV NIC to receive the response OAMPDU packet and this registration process has to go through hypervisor, therefore we could ask the hypervisor to supervise the process. In particular, we found NIC is responsible for writing a specific multicast address in MTA [4] and we implemented the checking routine as a module in NIC which blocks the registration request of OAMPDU multicast address.

Before describing the details of our mitigation, we first introduce the process on how address is registered in MTA to motivate our implementation choice.

The functionality of multicast filtering is implemented in NIC. Specifically, the *VM Offload Register (VMOLR)* in the

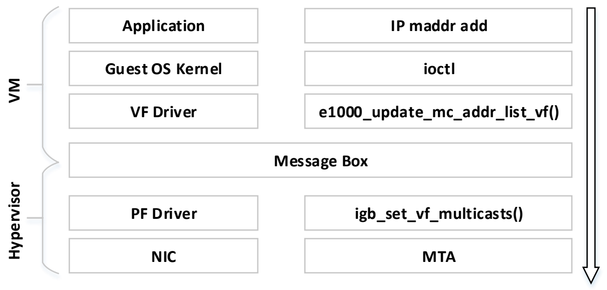

Figure 4: Multicast Addr Registration Call Graph.

NIC has a field *ROMPE* (the 25th bit) that can be used to enable or disable multicast forwarding (enabled by default). NIC will forward a multicast packet if the destination address of the packet is in the MTA when multicast forwarding is enabled. Therefore, VM has to write the multicast address ahead to MTA to let NIC forward those packets, which is accomplished by calling PF upon the request of VF invoked by VM.

Figure 4 shows the important functions invoked during multicast registration. On VM's side, command `IP` parses the request from user and invokes the relevant system calls based on the input parameters. When Linux kernel receives those calls, it checks the privilege of the invoking process and then forward the request to VF drivers' corresponding interface, i.e., function `e1000_update_mc_ad dr_list_vf()` of the igbvf driver, which is a linux base driver for Intel network connection[7]. The function computes the hash values for at most 30 input multicast addresses (the remaining ones will be discarded). Then this function composes a message using the least significant 12 bits of every hash value and sends the message to PF through the *mailbox system* which exchanges messages between VM and hypervisor. Mailbox system routes the message to the handler function `igb_set_vf_multicasts()` in the igb driver (PF driver) owned by the hypervisor. The PF function parses the message to get the hash values and stores them to the MTA for that VF. After that, the hypervisor replies `ACK` (if success) or `NACK` (if fail) to VF through the mail box. When these steps are completed, NIC forwards the multicast packets to VF if the destination address is in the VF's MTA.

Since the VM could be fully controlled by the adversary, we added a filtering module to the MTA updating functions in the PF driver code, i.e., within `igb_set_vf_multicasts()` (see Figure. 5) of the NIC, to prevent NIC from writing the OAM protocol address to the MTA for any VM. And all other parts of the driver are kept unchanged. In other words, a VM could still receive multicast packets from NIC except the ones related to OAM. Because the address will never appear in the MTA, forwarding unit in the NIC would not forward the OAMPDUs to VMs. As a result, VM can no longer set up the OAM connection with the switch to further launch the attack.

```
 1 static int igb_set_vf_multicasts(...){
     /*get number of maddr from msg;*/
 3   int n = ...;

 5   /*get pointer of hash list from msg*/
     u16 *hash_list = ...;
 7
     /*Add Filter Here*/
 9   for(i = 0; i < n && i < 30; i++)
       if(hash_list[i] == hash(OAMAddr))
11       hash_list[i] = dummyAddrHash;
     /*End of the Filter*/
13   ......
     /*Sanitized hash_list is written*/
15   for (i = 0; i < n; i++)
       vf_data->vf_mc_hashes[i]=
17       hash_list[i];
     ......
19 }
```

Figure 5: Code illustration of the filter. Line 9 to 11 are added while other codes are kept unchanged.

4.2 Evaluation

We modified the PF driver to add our code blacklisting the OAMPDU multicast address and compiled the customized driver. Then, we updated the driver of NIC and relaunched our attack. As expected, the adversarial VM can no longer complete the handshake process with the switch because it cannot receive the replies from the switch, thwarting our attack. The performance cost is negligible to hypervisor and VMs, since the check only takes place during address registration process. However, as shown in the next section, this check can be bypassed by an improved version of our attack.

5. UPDATED ATTACK

In this section, we propose an updated version of the attack scheme which can circumvent the defense scheme proposed in Section 4. The regular OAM communication process requires the handshake step to be completed before using other OAM functionalities. While our last defense could prevent adversarial VM from receiving OAMPDU packets from the switch, it does not stop adversary sending out packet to the switch. As such, it is possible for the adversary to *guess* the right parameters of the outbound OAMPDU packets to complete handshake process. Below, we elaborate the new attack scheme.

5.1 Attack without OAM replies

For a successful OAM handshake, VM needs to receive the TLV information returned from the switch. Without knowing that TLV, what the attacker can do is just to enumerate all the possible value and send the link fault signal with the guessed value to the switch. However, this is doable in reasonable time as shown later. We assume the switch works in passive mode to constantly receive and process the packets from the adversary. Even if the switch works in active mode, it will be switched to passive mode when the counterpart of the channel is in active mode, which can be set by the adversarial VM. The attack works as follows:

1. First, the attacker sends an Information OAMPDU with its own configuration contained in the local information TLV, like the regular process.

2. When the switch receives this packet, it replies an Information OAMPDU with switch's configuration in local information TLV field and the attacker's configuration in remote information TLV field. Local Discovery status and Remote Discovery status of the OAMPDU are set to be `Satisfied` and `Discovery` respectively according to the EFM standard. The reply packet however cannot be received by the adversary this time when our basic defense is deployed.

3. The adversary has to respond to the switch with a valid OAMPDU to finish the handshake process. In particular, the adversary has to correctly fill Local Discovery status, Remote Discovery status, Local TLV and Remote TLV. The first three fields are easy to fill: the adversary can assign `Satisfied` to both Discovery status fields and set the same local TLV value of the first packet. For remote TLV, the attacker enumerates the possible configuration setting of the remote switch and fills the corresponding value.

4. The attacker sends the OAMPDU packet to switch using the above crafted values. If the connection is terminated, the attack is successful. Otherwise, the attacker should go back to step 1 for another round and try a new remote TLV at step 3.

One would worry that the attack is not practical as a huge number of TLV has to be enumerated and the attack would be discovered before succeeded. Interestingly, **the range of valid TLV is quite narrow**, making the guess attempts ends in short time. Figure 6 shows the format of a TLV which can represents either local configuration information or remote configuration information. Some of the fields can be fixed with constant value: `Remote Information` for the information type field, `0x10` (16 bits) for Information length field (the length of a TLV), `0x01` for OAM version and `0x0000` for Revision. The valid values for the remaining fields are described below.

- **State.** There are only 3 possible values for Parser Action (forwarding, looping back, discarding) and 2 values (0 or 1) for Mux.
- **OAM configuration.** This segment has 5 fields while each one is only 1-bit long, adding up to $2^5 = 32$ possible values in total.
- **Max OAMPDU Size.** Maximum OAMPDU size is often set to the default value 128.
- **Vendor Identifier.** Organizationally Unique Identifier is assigned per manufacturer. According to the

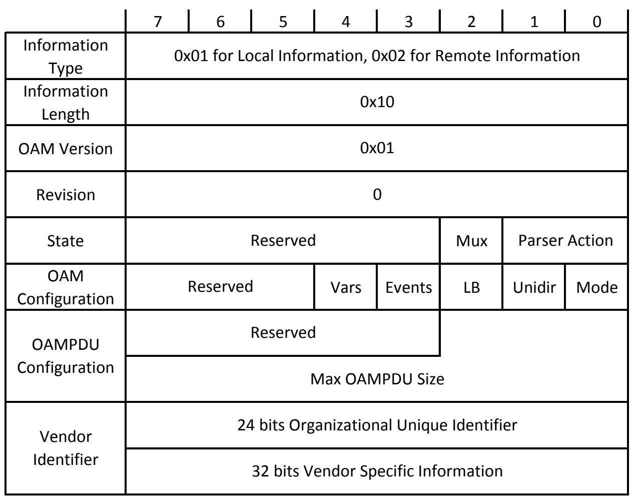

Figure 6: Information OAMPDU TLV format

statistics from IDC [9], the top 5 Ethernet switch vendors have taken the lion's share of the market, as shown in Table 3. In other words, using 5 values corresponding to the top vendors is sufficient in most cases. Vendor Specific Information is often set to all 0 as we examined a large number of the switch documents of mainstream switches.

Vendor	Cisco	HP	Huawei	Juniper	Arista	Others
Revenue	3827	592	393	243	217	1190

Table 3: Market share of switch vendors ($M).

Hence, an attacker only needs to guess at most $3 * 2 * 32 * 5 = 960$ TLV values till finding the right TLV. As one optimization, the adversary could start from the most popular configurations, e.g., the top switch vendor identifier, to reach the correct answer more quickly.

Next, we compute the time required for trying the 960 possible TLV values. An end host can send at most 10 OAMPDU packets per second, while a switch replies 1 OAMPDU per second by default. Our attack requires VM to send 3 OAMPDU packets and wait the switch to reply with 1 OAMPDU packet, so the attacker can try a TLV value in less than 1.5-second interval, assuming that the switch works at the default setting. In the end, an attacker can exhausts all possibilities in 24 minutes at most.

Under the real-world settings, the time overhead could be reduced to a much lower amount. Network managers tend to modify only several fields and keep others unchanged. Assuming that at most 2 bits among the 5 OAMPDU configuration bits are modified by the network manager, attacker needs to try $(C_5^2 + C_5^1 + C_5^0) * 5 = 80$ times maximally to find the right TLV, which means in average, the attacker only needs to guess 40 times, taking around 1 minute.

5.2 Evaluation

We tested the updated attack scheme on the same platform mentioned in Section 3.3. The basic defense described in Section 4 is deployed to prevent VMs from receiving OAMPDUs. We discussed with an experienced network administrator and modified two configuration item of the switch to non-default values and kept other configurations unchanged to simulate the real-world settings.

Though based on the theoretical analysis, in average tens of guesses have to be made to pass the check from switch, surprisingly it took only one guess to succeed no matter what value we chose for the two configuration items. It turns out the root reason is that the switch does not compare the remote TLV within the received OAMPDU packet with its own configuration at all. We consider this is a severe vulnerability and have reported it to the product security team (PSIRT) of the vendor (Huawei). The team confirmed the vulnerability and committed to fix it in the later versions. The team also replied to us that **the switch does not check the TLV because the RFC standard does not specify the checking process**. Hence, they considered it a generic problem they plan to report to IETF. We foresee it would take a long time before the standard is mended, but even it happens, our attack is still viable as the attacker is able to guess the right configuration in short time.

The consequence is similar: the attacker can successfully disconnect the physical machine and the recovery process is also painful. The network manager must use her account to login the switch and reset the interface to wake the port up from `Error Down`.

6. LIMITATIONS OF SRIOV NIC

For the updated attack, leveraging hypervisor to terminate the handshake process between VM and switch is impossible as hypervisor is not on the data plane. The only countermeasure available is letting the NIC to inspect the outbound packets but unfortunately there is no way to easily implement the filtering mechanism. We read through the document of the I350 NIC we experimented with [6] and other documents of the advanced models (e.g., Intel 82599 10GbE NIC [5]). It turns out there is no programmable internal filter inside the NIC, or VEB more specifically, that we can use to switch packets between VMs and external LAN, handle broadcast or multicast packet forwarding.

The outbound packets could be controlled by the Tx switching to some extent. There are some filters like L2 filters and VLAN filters implemented in the original SRIOV NIC which redirect the packets to the virtual pool and decide which one should be sent out through a series of matching process. However, there is one exception of Tx packet filtering [2]: **multicast and broadcast packets are always sent to external LAN**. That means there is no way for an existing NIC to stop VMs from sending multicast or broadcast packets. Routing multicast packet to its destinations is implemented in MTA but MTA cannot be used for filtering. Intel does provide 3 outbound security mechanisms: MAC anti-spoofing, VLAN tag-validation and VLAN anti-spoofing. However, these mechanisms are all targeting spoofed and invalid Tx address which cannot be leveraged for defense, since the Tx address used for attack is valid. In fact, the SRIOV NIC design overlooked the special usage of multicast protocol and simplified the multicast forwarding procedure such that all ethernet control protocols based on multicast run mistakenly over the virtualized environment.

In addition, the OAM vulnerability not only impacts machines equipping SRIOV NIC, but also neighboring machines

[2]Rx switching does not have this exception.

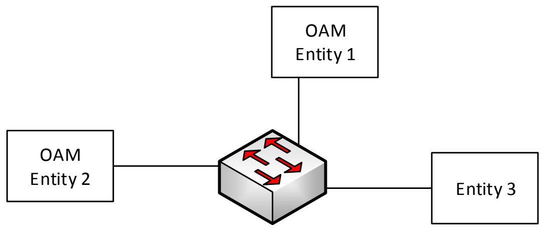

Figure 7: An incorrect network structure.

sharing the same network, when the network is configured incorrectly. We show one such network structure in Figure. 7, where 2 OAM entities (`Entity 1` and `Entity 2`) and an OAM incapable entity (`Entity 3`) are connected to an OAM incapable switch. As the switch cannot handle OAM packets, it will *always forward* them to other connected entities. In this case, when `Entity 1` reports link error, the link between `Entity 2` and the switch will be shut down because the OAMPDU is relayed to `Entity 2` who assumes the error is reported by the connected switch. Subsequently, the link between `Entity 3` and `Entity 2` will be cut off. As such, an attacker controlling one entity could force disconnections of other innocent entities. We tested this attack following the paradigm shown in Figure. 7 and found the attack always succeeded. To mitigate this issue, the network administrator should keep the devices consistent in OAM handling (on/off for all devices).

7. UPDATED DEFENSE

Filtering outbound multicast packets from VM on the NIC side is not a feasible defense solution due to the limited filtering capabilities provided by the current hardwares. As an alternative way, we look for the defense schemes on the switch side which aims to identify whether the OAMPDU is from VM. Below we describe a practical defense method and also discuss other solutions.

7.1 Defense through Distinguishing VMs

Our updated attack could succeed because the amount of valid TLV values in the inbound OAMPDU to NIC is too small. While the handshake process is designed to prevent a random entity from reporting error message through OAM, it is bypassed under this problematic implementation. Intuitively, we could fix this vulnerability through introducing more randomness into the OAMPDU.

Extending the OAM protocol to add an additional field bearing a random number could serve the defense purpose but might cause compatibility issues. Fortunately, we found the 32-bit Vendor Specific Information (VSI) field could be used (see last row of Figure 6). It is always set to all 0 and unused by switch. Therefore, we design the updated defense by filling a random number into this field as a secret in the outbound OAMPDU and checks if the same value is observed in the follow-up inbound OAMPDU. This modification occupies the VSI that is designed for vendor specific purpose. If a vendor has already occupied this field for its private functionalities, the private functionalities can be implemented in Organization Specific TLV that is also designed for vendor specific purpose and has better extensibility resulted from a larger space.

Specifically, we could command switch to generate a random 32-bit number and fill it into the VSI field whenever sending reply OAMPDUs to the remote entity, as shown in Figure.8. A valid entity would see the random number and the response will automatically include the same number (in remote TLV) according to the EFM standard. For an adversary, since the handshake reply never reaches VM, she has to guess the number, which lies in a much larger range. On the side of switch, if VSI field of the inbound reply OAMPDU packet has an unmatched value, the packet is dropped.

We further analyze this scheme in three aspects below.

- **Effectiveness.** Attacker has to explore the configuration space of 2^{32} values to launch one successful attack. There is a 1.5-second interval between two attempts. As the result, it takes the attacker over 200 years in worst case (100 years in average for uniformly generated random number) to succeed. Without this defense, an attacker could succeed under a minute. Therefore, our defense substantially raises the bar for the attack.
- **Cost.** The only change to OAM protocol is adding a random number to OAMPDU, which requires modification to the switch firmware for random number generation and received value verification. The NIC vendors need to provide a patch to the driver and notify the customers to upgrade their NIC driver to deploy the basic defense scheme described in Section 4. The raised cost is moderate to them. For hypervisor and VMs, there is no extra cost.
- **Performance.** The penalty is incurred when exchanging OAM packets, while all other components are not impacted. Given that OAM is a slow protocol (0.1-second interval between communications at most), the increased time delay is very small comparing to the overall overhead.

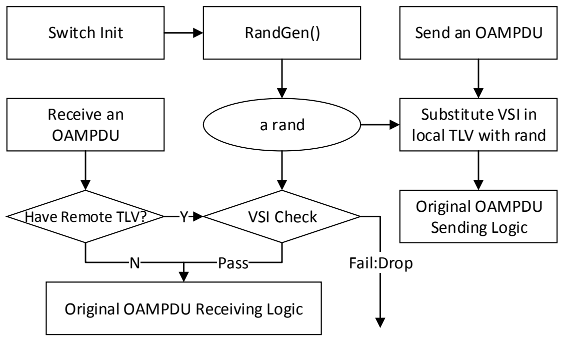
Switch Init
RandGen()
Send an OAMPDU
Receive an OAMPDU
a rand
Substitute VSI in local TLV with rand
Have Remote TLV?
Y
VSI Check
Original OAMPDU Sending Logic
N
Pass
Fail:Drop
Original OAMPDU Receiving Logic

Figure 8: Flow chat of the defense scheme.

Evaluation. The ideal scenario for evaluating our new defense scheme is to modify the switch firmware and test under the same settings as Section 5.2. This is very difficult since we cannot get the source code from the switch vendor. As an alternative solution, we implemented an emulated switch that runs the modified OAM protocol using a PC (we call it switch PC) and connected the virtualized server to the switch PC. The virtualized server follows all the steps required by the OAM standard and the additional routines for randomness.

We let the VM to repeatedly guess the vendor specific information and send out OAMPDU to the switch PC. In the meantime, we check if any such packet can successfully arrive at the destination. After running the experiment for 24 hours, no OAMPDU packet sent from VM can be observed, suggesting our defense is effective.

7.2 Other Approaches

7.2.1 Checking MAC Address

If the switch could differentiate VMs and hypervisor, our attacks will be easily thwarted. The most straightforward way to achieve this goal is to examine the source MAC address of OAMPDU. Network manager could compile a whitelist of hypervisors' MAC addresses and integrate the list into the switch. When the switch receives an OAMPDU packet from an unlisted sender, it drops the packet. Because VMs cannot fake a source MAC address, OAMPDUs from VMs are always dropped.

The major advantage of this defense scheme is that it requires only changes to the receiving logic of the switch. The program logic to read the MAC address of the remote OAM entity has been built into the switch we tested (the MAC address of remote OAM entity is displayed in the management interface of the switch). So, switch vendors could just add a small piece of code to check if the address is in the whilelist. Besides, the scheme has negligible performance penalty because all changes are made in control plane and date plane is kept unchanged.

This scheme has a drawback that a lot of manual works have to be introduced when the network is quite dynamic. For instance, when a new network device is connected to the switch, manager must use her credential to login to the switch and add its MAC address to the whitelist.

7.2.2 Filtering by Hypervisor

We also consider filtering out the OAMPDUs by a software based switch instead of the NIC or physical switch. Smolyar *et al.* proposed a filtering technique to mitigate DoS attack in virtualized environment [24]. The main idea is to let one Ethernet controller forward all packets it received to the hypervisor by utilizing a software bridge. The bridge removes all the illegal packets and forwards the remaining packets to another Ethernet controller that is connected to the switch. We evaluated the feasibility of this defense here as well and concluded it is not practical for defending our attack.

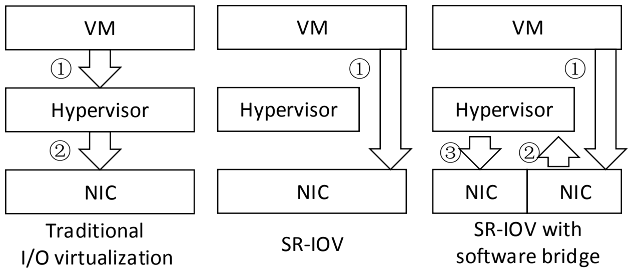

Figure 9: Comparison of the 3 other approaches.

This method seems to be effective against our attacks but it dims the benefits of high performance SRIOV NICs: transparent data path to hypervisors.

First, obviously, two Ethernet controllers (one for inbound and one for outbound) are required which means there is one Ethernet controller not available to the user.

More importantly, the performance penalty is also non-negligible, because all the communications on the data path need to go through hypervisor, which violates the fundamental goal of SRIOV. As shown in Figure. 9, for the SRIOV scheme, a packet sent by VM is transferred from VM's memory directly to NIC's buffer. For traditional I/O virtualization technique, the packet is copied from VM's memory to hypervisor's memory through a virtual NIC first, and then a software bridge in the hypervisor forwards it to physical NIC, which incurs 2 times I/O overhead compared to SRIOV scheme. For SRIOV with defense scheme from [24], packets are sent from VM to NIC and then copied back from NIC to hypervisor's memory for filtering and finally sent with another NIC, which incurs overall 3 times I/O overhead. Such heavy performance penalty makes the adoption questionable, because it is even worse than the traditional I/O virtualization.

7.2.3 Updating NIC Hardware

NIC vendors are able to update their next generation products to provide enough outbound filtering capabilities and open interfaces to hypervisor to defend this attack. This mitigation could resolve all the security issues discovered by us, but the cost could be considerable. In the meantime, the old models are still vulnerable.

8. RELATED WORKS

Network interface virtualization is a hot research area and the related techniques have been widely adopted by many organizations. Sheta *et al.* surveyed the widely used NIC virtualization techniques and evaluated them with different metrics [23]. Yassour *et al.* evaluated the SRIOV method for the network interface [31]. Dong *et al.* elaborated the SRIOV networking architecture in the Xen hypervisor [14].

The security problems of network interface virtualization are also studied. Recently, Richter *et al.* and Smolyar *et al.* proposed two different DoS attacks [22, 24] against virtualized network interface. Pek *et al.* discovered a series of attacks that can be launched under a direct device assignment settings [19], using an automatic tool to discover hardware-level vulnerabilities. Different from their works, our attacks can totally cut off the network connectivity of the machine, resulting in much worse consequences. Besides causing connectivity issues, SRIOV NIC can be exploited as a side channel for detecting co-located VMs in the cloud, as shown in [10].

The I/O virtualization technologies can be used to launch or defend against other attacks. Wojtczuk *et al.* showed a Message Signalled Interrupt(MSI) based attack on x86-64 machine, allowing a virtual machine to run privileged code in Xen environment [29]. Stewin *et al.* studied how malware can attack the host machine stealthily [25]. As an example, they implemented a key logger that attacks both Windows and Linux platforms. They used IOMMU to defend that kind of attacks. Still, attacker can modify the hardware-level data structure or configuration table to bypass the IOMMU isolation, as stated in [16].

9. CONCLUSION

We discovered a new vulnerability in SRIOV NIC. The root cause is that SR-IOV NIC does not provide sufficiently fine-grained outbound traffic filtering capability and the hypervisor is bypassed in SRIOV model. Exploiting such vulnerability, an attacker can fake OAM packets to break the connection between the host machine and the switch. As the result, the network connectivity of all VMs and hypervisor can be cut off.

The key to the successful attack is to handshake with the switch through SRIOV NIC. We show one attack scheme which asks help from MTA in the SRIOV NIC, and another which could even succeed without the help of MTA. The result suggests that both attacks are effective and practical.

In the end, we propose an effective defense scheme by using an implied challenge response. Since SRIOV devices are largely deployed only in recent years, we urge the community to thoroughly examine the attack surface against SRIOV devices and the vendors to actively fix the relevant security issues.

Acknowledgment

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 61572415 and Hong Kong S.A.R. Research Grants Council (RGC) under Early Career Scheme No. 24207815.

10. REFERENCES

[1] Configuration guide - reliability. http://support.huawei.com/enterprise/docinforeader.action?contentId=DOC1000019450&partNo=10062.

[2] Enabling enhanced networking on linux instances in a vpc. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html.

[3] How to: Enable single root i/o virtualization in exalogic elastic cloud. https://blogs.oracle.com/opscenter/entry/how_to_enable_single_root.

[4] Intel 82576 sr-iov driver: Companion guide. http://www.intel.com/content/www/us/en/embedded/products/networking/82576-sr-iov-driver-companion-guide.html.

[5] Intel 82599 10 gbe controller datasheet. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/82599-10-gbe-controller-datasheet.pdf.

[6] Intel ethernet controller i350: Datasheet. http://www.intel.com/content/www/us/en/embedded/products/networking/ethernet-controller-i350-datasheet.html.

[7] Intel linux based igb driver. http://www.intel.com/content/www/us/en/support/network-and-i-o/ethernet-products/000005767.html.

[8] Oam introduction from cisco. http://www.cisco.com/c/en/us/td/docs/ios/12_2sr/12_2sra/feature/guide/srethoam.html.

[9] Worldwide ethernet switch market and enterprise and service provider router market. http://www.idc.com/getdoc.jsp?containerId=prUS41061316.

[10] A. Bates, B. Mood, J. Pletcher, H. Pruse, M. Valafar, and K. Butler. Detecting co-residency with active traffic analysis techniques. In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*, CCSW '12, pages 1–12, New York, NY, USA, 2012. ACM.

[11] M. Ben-Yehuda, J. Mason, J. Xenidis, O. Krieger, L. Van Doorn, J. Nakajima, A. Mallick, and E. Wahlig. Utilizing iommus for virtualization in linux and xen. In *OLSâĂŹ06: The 2006 Ottawa Linux Symposium*, pages 71–86. Citeseer, 2006.

[12] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan. High performance network virtualization with sr-iov. *Journal of Parallel and Distributed Computing*, 72(11):1471–1480, 2012.

[13] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan. High performance network virtualization with sr-iov. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pages 1–10, 2010.

[14] Y. Dong, Z. Yu, and G. Rose. Sr-iov networking in xen: Architecture, design and implementation. In *Workshop on I/O Virtualization*, 2008.

[15] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir. Eli: bare-metal performance for i/o virtualization. *ACM SIGPLAN Notices*, 47(4):411–422, 2012.

[16] O. Isfort, K. Müller, D. Münch, and M. Paulitsch. Decreasing system availability on an avionic multicore processor using directly assigned pci express devices. In *European Workshop on System Security (EUROSEC2013)*. Czech Republic Prague, 2013.

[17] P. Kutch. Pci-sig sr-iov primer: An introduction to sr-iov technology. *Intel application note*, pages 321211–002, 2011.

[18] J. Liu. Evaluating standard-based self-virtualizing devices: A performance study on 10 gbe nics with sr-iov support. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[19] G. Pék, A. Lanzi, A. Srivastava, D. Balzarotti, A. Francillon, and C. Neumann. On the feasibility of software attacks on commodity virtual machine monitors via direct device assignment. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 305–316. ACM, 2014.

[20] H. Raj and K. Schwan. High performance and scalable i/o virtualization via self-virtualized devices. In *Proceedings of the 16th international symposium on High performance distributed computing*, pages 179–188. ACM, 2007.

[21] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner. Achieving 10 gb/s using safe and transparent network interface virtualization. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 61–70. ACM, 2009.

[22] A. Richter, C. Herber, T. Wild, and A. Herkersdorf. Denial-of-service attacks on pci passthrough devices: Demonstrating the impact on network-and storage-i/o performance. *Journal of Systems Architecture*, 61(10):592–599, 2015.

[23] R. Shea and J. Liu. Network interface virtualization: challenges and solutions. *Network, IEEE*, 26(5):28–34, 2012.

[24] I. Smolyar, M. Ben-Yehuda, and D. Tsafrir. Securing self-virtualizing ethernet devices. In *Proceedings of the 24th USENIX Conference on Security Symposium*, pages 335–350. USENIX Association, 2015.

[25] P. Stewin and I. Bystrov. Understanding dma malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 21–41. Springer, 2013.

[26] J. L. V. U. J. Stoess and S. Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, 2004.

[27] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *USENIX Annual Technical Conference, General Track*, pages 1–14, 2001.

[28] P. Willmann, J. Shafer, D. Carr, S. Rixner, A. L. Cox, and W. Zwaenepoel. Concurrent direct network access for virtual machine monitors. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 306–317. IEEE, 2007.

[29] R. Wojtczuk and J. Rutkowska. Following the white rabbit: Software attacks against intel vt-d technology. *ITL: http://invisiblethingslab.com/resources/2011/Software\%20Attacks\%20on\%20Intel\%20VT-d.pdf*, 2011.

[30] X. Xu and B. Davda. Srvm: Hypervisor support for live migration with passthrough sr-iov network devices. In *Proceedings of the12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 65–77. ACM, 2016.

[31] B.-A. Yassour, M. Ben-Yehuda, and O. Wasserman. Direct device assignment for untrusted fully-virtualized virtual machines. Technical report, Technical Report H-0263, IBM Research, 2008.