Understanding the Security Implication of Aborting Live Migration

Xing Gao[1], Jidong Xiao[2], Haining Wang[3], Angelos Stavrou[4]
[1]University of Memphis, [2]Boise State University, [3]University of Delaware, [4]George Mason University
xgao1@memphis.edu, jidongxiao@boisestate.edu, hnw@udel.edu, astavrou@gmu.edu

Abstract—Live migration of Virtual machines (VMs) has become a regular tool for edge and cloud operators to facilitate system maintenance, fault tolerance, and load balancing, with little impact on running instances. However, the potential security risks of live migration of VMs are still obscure. In this paper, we expose a new vulnerability in the existing VM live migration approaches, especially the *post-copy* approach. The entire live migration mechanism relies upon reliable TCP connectivity for the transfer of the VM state. We demonstrate that, if the host server is vulnerable to off-path TCP attacks, the loss of TCP reliability leads to VM live migration failure. We demonstrate that, by intentionally aborting the TCP connection, attackers can cause unrecoverable memory inconsistency for *post-copy*, leading to a significant increase in downtime and performance degradation of the running VM. Additionally, we present detailed techniques to reset the migration connection under heavy networking traffic. We also propose effective defenses to secure the VM live migration. Our experimental results demonstrate that memory inconsistencies could be devastating to some applications, and it only takes a few minutes to reset a heavy migration connection.

Index Terms—Virtual Machine, Live Migration, TCP Reset Attack.

1 Introduction

Virtual machine (VM) live migration is the process of moving a running VM between two different physical machines with minimal disruption to the VM's normal operations. As a powerful tool for system maintenance [6], load balancing [52], fault tolerance [38], energy saving [39], [11], and performance enhancement [5], VM live migration has been widely used in edge computing [51], [42], [4] and mainstream cloud services [32]. The procedure of live migration includes the transfer of all memory pages and hardware state over a pre-established TCP connection. One primary method used by the existing tools is the *pre-copy* approach [10], which first iteratively copies all memory pages as well as those pages dirtied in the previous iteration. The VM is kept running until the last iteration, when *pre-copy* pauses the VM and then finishes copying the rest of the memory pages. The total length of time for the last procedure in *pre-copy*, known as *service downtime*, reaches up to many seconds [57] depending on the copying rate and intensity of memory activities. To reduce service downtime, the *post-copy* approach [23] was recently proposed. The *post-copy* approach immediately suspends the source VM and copies the minimum execution states. The VM is quickly resumed at the destination side, and memory pages are then transferred. Using the *post-copy* approach, a significant amount of service downtime could be saved, and various hypervisors have already integrated this mechanism.

While researchers have made numerous efforts to improve the performance of VM live migration, little attention has been paid to the security implications of live migration. One potential reason is that, currently, live migration is not made available to non-admin users in the cloud or edge environment. Moreover, the confidentiality of the process can be safeguarded using encrypted live migration [65], [43]. This can further secure the migration process, preventing adversaries in the middle of a network to hijack or falsify memory pages.

In this paper, we present a new vulnerability in the existing VM live migration caused by the insecure TCP channel established for the transfer of memory pages. We demonstrate that a vanilla TCP-RST attack could lead to devastating and unrecoverable consequences for today's VM live migrations, especially *post-copy*-based approaches. By inducing a reset of the TCP connection of a *post-copy* live migration, adversaries can force a VM to enter into an inconsistent memory state, causing failures and unpredictable behavior. The reason is that *post-copy* immediately stops the VM on the source and starts running the VM on the destination server without any checks as to the integrity of the transfer process. The consequence of terminating the TCP connection is that the VM on the destination server becomes unresponsive or in many cases crashes. This is due to missing parts of the memory did not transfer properly from the source. Meanwhile, the newly modified memory pages on the destination server are also unavailable to the stopped VM on the source, resulting in memory inconsistencies. Such a vulnerability is caused by the design of *post-copy* rather than the actual implementation in the hypervisor. The lack of protection in existing hypervisors' implementation further exacerbates this problem, making the consequences severe and the remote VM image unrecoverable. Furthermore, the TCP reset attack could also significantly increase the service downtime and degrade the performance of the running VM, for both *pre-copy* and *post-copy* live migration approaches.

We analyze in depth the possible approaches to launch the TCP reset attack on VM live migration. In particular, we propose a new method to discover the timing of a migration process as well as the destination port number. Attackers can infer that useful information by intentionally initiating fake migration requests. In some cases, attackers are even able to hinder the migration connection from being established. While the process of VM live migration includes heavy networking traffic, we further propose an enhanced algorithm that exploits the challenge ACK vulnerability [7] to reset the migration connection. Experimental results show that our attack can break the migration in a few minutes. In terms of defense strategies, we present two possible mechanisms to mitigate the TCP reset vulnerability from becoming a serious threat for live migration in cloud and edge environments. We implement a prototype by modifying QEMU and the Linux kernel.

The rest of this paper is organized as follows. Section 2 introduces the background of VM live migration and describes the reset threat to TCP connections. Section 3 presents the vulnerability in VM live migration process, and details the method to reset the migration connection within heavy networking traffic. Section 4 shows the evaluation of our attack on VM live migration from different aspects. Section 5 discusses several potential defense mechanisms. Section 6 surveys related work, and finally Section 7 concludes.

2 Background

2.1 VM Live Migration

Clark et al. [10] first proposed VM live migration as a procedure of migrating an entire OS as well as all of its applications as one unit from one host machine to another over a reliable connection (typical a TCP connection). In the rest of the paper, we use TCP as a practical example. All states and system resources of the original VM, including memory, storage, and network connectivity, are transferred from the source to the destination. While the key procedure for VM live migration is the transfer of the main memory state, two major approaches, *pre-copy* [10] and *post-copy* [23], [24], are widely deployed. Based on those two approaches, a body of works [53], [28], [25] further improve the performance of VM live migration.

2.1.1 Pre-copy

Typically, a *pre-copy*-based VM live migration involves four steps. First, the source server and destination server enter into a ready-for-migration mode. In particular, the destination side should be listening at a specified port. The TCP connection is then established on that port once the migration command is issued. After that, the memory of the source VM will be copied in an iterative fashion. It first transfers all the pages to the target server. During this iteration, some pages become dirty. The migration process then continues to repeatedly copy those newly generated dirty pages. After several rounds, if the page dirtying rate is continuously faster than the rate of copying, the iterative copying will not converge. Then, the source VM would be suspended, and the rest of the inconsistent memory is copied. Finally, the VM on the destination side is activated, and the source VM is paused. As we see, the VM must be taken down to finish the migration. The amount of service downtime depends on the running workloads as well as the bandwidth of the TCP connection. The downtime could be huge if there is a large changing rate for dirty pages.

2.1.2 Post-copy

Different from *pre-copy*, which starts with copying memory pages, *post-copy* first stops the source VM. During this downtime, only minimum execution states needed by the VM to start (e.g., the processor state) are copied. After this minimum copying, the VM is immediately resumed on the target side. Then, *post-copy* copies the memory pages from the source to the destination. Several techniques could be applied to fetch memory: (1) *demand paging*, which transfers the pages that page faults are generated by the resumed VM; (2) *active pushing*, which actively pushes the VM's pages to the target; and (3) *prepaging*, which predicts the pages that might be accessed by the running VM and transfers the pages before they are faulted. Compared with the *pre-copy* approach, *post-copy* can significantly reduce service downtime, and thus has recently been implemented in various hypervisors, including Xen and KVM.

2.2 TCP Reset Attack

A TCP connection is defined by a four-tuple `<source IP address; destination IP address; source port number; destination port number>`. Some features like sequence numbers and acknowledgment numbers are utilized to ensure a reliable connection. Normally, attackers need to obtain all four tuples as well as a valid sequence number to interfere with the connection. While the TCP protocol was not initially designed for security concerns, vulnerabilities exist for attackers to infer the four tuples and sequence numbers. One type of those malicious attacks is the TCP reset attack. By forging TCP packets with specific flags, attackers can terminate an existing TCP connection.

2.2.1 Blind Reset Attack

RFC 5961 [49] defines two types of blind reset attacks: using SYN bit or RST bit. Before RFC 5961, if the sequence number of an incoming SYN or RST packet is in the valid receive window, the receiver would reset this connection. The blind in-window reset attack was first described by Watson [58]: with the knowledge of the four tuples, attackers can reset the connection if the sequence number of the crafted packet is in the window. As a result, a brute-force attacker can simply send one packet in each possible window. A TCP connection uses a 32-bit number recording the next expected in-order sequence number, and another 32-bit number for the last accepted sequence number. A 16-bit number is used to report the receive window size ($wsize$), which has the largest value, 2^{16}. The total number of packets required for the attack is $2^{32}/wsize$, which could be conducted within several seconds in a high-speed networking environment.

RFC 5961 suggests a tight handling of incoming reset packets. For an SYN packet, regardless of the sequence number, the receiver sends back an ACK packet (known as a *challenge ACK*) to request an RST packet with the correct sequence number from the sender to terminate the connection. For an RST packet, the receiver would simply drop the packet if the sequence number is outside of the valid receive window. For an in-window RST packet, an ACK packet with the correct sequence number would be sent to confirm the loss of connection. With the implementation following RFC 5961, a blind attacker can only terminate the TCP connection with a matching packet. The possibility of a blind attack is significantly reduced to $1/2^{32}$.

2.2.2 Side-channel TCP Attacks

Even when strictly following RFC 5961, previous works have shown that off-path attacks are still possible by leveraging various information leaked from specific side-channels [45], [46], [21], [14], [13]. While those works rely on a form of malware on the client side, a more recent work [7] uncovers a new vulnerability ([CVE-2016-5696]) and allows blind off-path reset and data injection attacks.

The vulnerability is raised because of the challenge ACK. As introduced above, RFC 5961 handles incoming packets in a strict way. A challenge ACK is sent in various situations. Moreover, to save system and bandwidth resources for sending ACK packets, RFC 5961 introduces a challenge ACK throttling mechanism. A global maximum number of challenge ACKs (shared by all TCP connections in the server) is set to 100 by default in a 1-second interval (on Linux). To exploit the vulnerability, an attacker can abuse a regular TCP connection to measure the remaining challenge ACK counters, while simultaneously sending probe packets. If the crafted packet triggers a challenge ACK, the number of remaining challenge ACKs would be less than 100. The work [7] shows that attackers can infer the four tuples as well as the sequence and ACK numbers in an idle or slow TCP connection.

Linux kernel maintainers soon released several patches to defend against the challenge ACK vulnerability. For instance, the limit is raised from 100 to 1,000 by setting sysctl knobs. However, huge amounts of servers are still left unpatched. Previous work [47] shows that less than 40% of servers of Alexa Top 1 million websites are patched after six months of the releasing of the patch. Also, such a vulnerability were exploited on TCP connections with light or even idle traffics. Heavy connections, such as VM live migration, are believed to be safe under such exploitation.

3 Threats in VM Live Migration

The TCP-RST attack is a DoS (Denial-of-Service) attack attempts to terminate an active TCP session. It has long ignored by cybercrime organizations or secure-services providers due to its limited effects. According to a recent report from Akamai [1], TCP reset attacks only account for about 1 percent of all DoS attack activities. For most applications, a successful TCP-RST attack causes a lost connection, but a simple re-establishment of the connection could solve the issue. On the other side, while numerous research efforts have been made to improve the performance of VM live migration, little attention has been paid to the security aspects. The reason is that live migration cannot be initiated by normal users in the cloud or edge environment. Hence, it is generally believed that the existing approach is secure enough, and many protection mechanisms are not taken seriously.

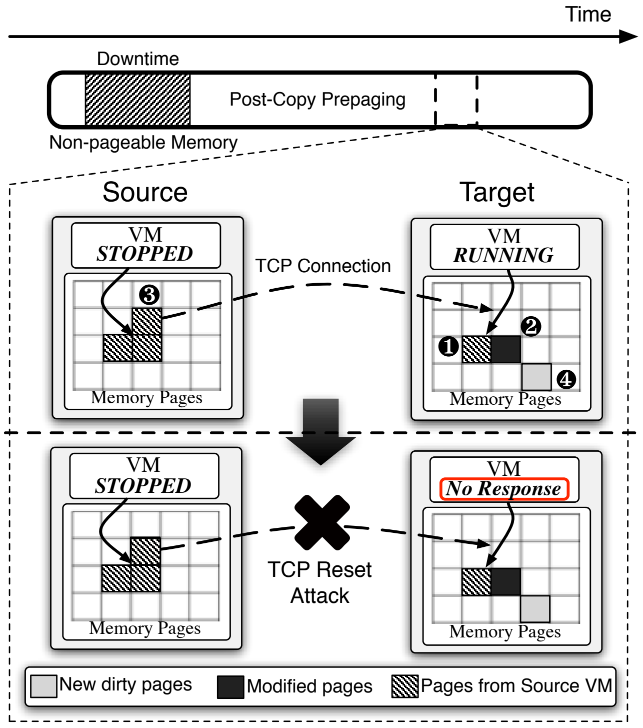


Fig. 1: Post-copy VM Live Migration.

In this section, we argue that the TCP-RST attack could lead to devastating and unrecoverable consequences for *post-copy*-based VM live migrations. The termination of a TCP connection would cause a VM to enter into an inconsistent memory state. Besides, while one remarkable advantage for *post-copy* is the short service downtime, we show that the TCP reset attack could significantly increase service downtime. Also, the performance of the running VM might be affected for both *pre-copy* and *post-copy* live migration approaches. We first present the vulnerability overview for *post-copy* and *pre-copy* VM live migration. We then detail the procedure to launch the TCP reset attack on the VM live migration.

3.1 Vulnerability Overview

3.1.1 Post-copy

Post-copy stops the source VM immediately after the TCP connection is established, and runs the VM on the target as soon as possible by transferring the minimum processor states. This strategy greatly reduces the service downtime. The rest of the memory pages are actively transferred from the source to the target, and the destination node will be halt if particular required pages are being transferred. Figure 1 shows a snapshot of memory page statuses at one time. In this procedure, the case ❶ represents memory pages that have been transferred from the source to the target. Since the VM is running on the target server at the same time that memory pages are copying (as the case of ❸), pages that have been transferred might be modified (as the case of ❷), and new dirty pages are generated (as the case of ❹).

If attackers can terminate the underlying connection at this point, in the current hypervisor implementation, the VM on the target server would become unresponsive immediately and cannot be recovered. Particularly, for a TCP connection, this can be achieved by successfully launching a TCP RST attack. In some cases, the VM would crash. The reason is that part of the memory is still on the source (like the case of ❸ in Figure 1). There is no way to resume the target VM. On the other side, the source VM still remains stopped. As we see, the service encounters an outage since both VMs are forced to shutdown. The bigger problem is inconsistent memory: all newly generated dirty pages in the target VM (as the cases of ❷ and ❹) are lost. This disastrous scenario is similar to a sudden power loss.

3.1.2 Pre-copy

In the scenario of *pre-copy*, the damage might be less than in *post-copy* migration. Similarly, once the TCP connection is reset, the destination VM crashes; however, the source VM is still alive and holds an entire up-to-date memory and processor states. In modern hypervisor implementation, the whole migration procedure must be restarted. Since migration is typically used as a tool to improve the performance of the VM or upgrade the server, such an attack would waste *pre-copy*'s total migration time, consume system resources, and degrade the performance. In particular, the service downtime of *pre-copy* is much larger than *post-copy*. If the TCP-RST attack happens during the downtime, the downtime would be significantly increased because both VMs are shutdown.

3.2 Attacking VM Live Migration

3.2.1 Threat Model

The attacker sets its target on destroying VM live migration in the edge or cloud environment by resetting the TCP connection. We assume that the attacker knows several IP addresses of the servers in the target cloud or the nodes in the target edge environment. By monitoring these IP addresses, the attacker attempts to block the migration from the source server to the destination server. We also assume that at least one server is vulnerable to the blind reset attack or side-channel reset attack mentioned in Section 2. The attacker can send crafted packets with spoofed IP address to the target servers. Additionally, the server has an open port (e.g., port 22) that allows the attacker to build a legitimate TCP connection (note that the attacker does not need to have a valid SSH connection with credential exchanged). In the cloud environment, bandwidth between servers is limited. To ensure the quality of services of other servers, administrators would not allow a migration to consume all network resources. Thus, we assume that the migration process has a constant but limited speed.

3.2.2 Possibilities of TCP RST Attacks

The ability to send crafted RST packets is the pre-requisite for launching TCP reset attacks. First, in an edge environment, attackers can use their own devices to send packets to the target edge node since edge computing is performed in an open environment where any nearby users can connect to edge nodes or devices.

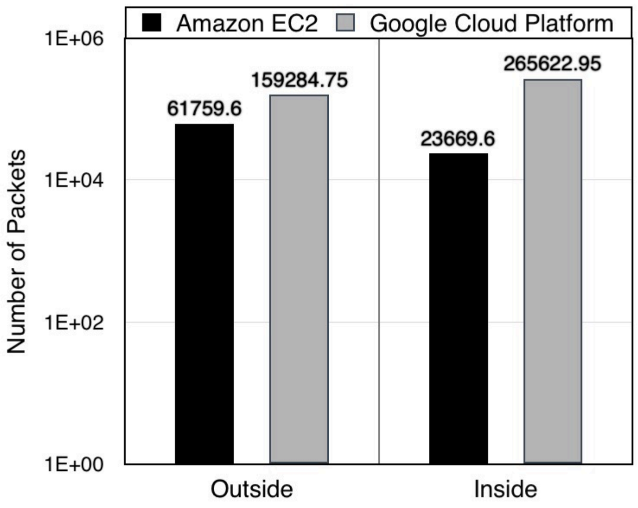


Fig. 2: Number of RST packets per second in clouds.

We further investigate whether the firewalls of cloud vendors would drop the RST packets for safety purposes or not. We choose two most popular clouds, Google Cloud Platform (GCP) and Amazon Web Services (AWS). We build a small tool to flood crafted reset packets into our subscribed instances, and measure the number of receiving packets using *tcpdump*. We choose the time interval of one second and repeat the experiment for 60 times. For AWS, we use a t2.micro instance in Amazon EC2. For GCP, we run an instance with 1 vCPU and default other configurations. We test a limited number of RST packets from inside and outside the clouds. From the outside, we send the crafted packets from our own servers. Inside the cloud, we send the packets from another rented instance.

Figure 2 illustrates our results. Both cloud services accept crafted RST packets. In GCP, the number of RST packets could be more than 100,000 per second from the inside and outside. For EC2, the number of RST packets from the outside could be more than 60,000 per second. While the number of RST packets from the inside is smaller than from the outside in EC2, we can still send about 23,000 packets per second. The cause of this difference might be that the outgoing traffic of our t2.micro is limited.

The live migration of VM could be applied either inside one data center or across different data center sites. For across site migration, once the cloud does not block RST packets, the migration procedure is vulnerable to TCP reset attacks. For migration inside a cloud, attackers need to rent instances inside the clouds. However, in these two clouds, the VMMs (Virtual Machine Managers) block packets with spoofed inner IP addresses at the guest level. Under such a circumstance, an attacker needs to either rent a dedicated server where it can fully control the hypervisor, or compromise the VMM of the subscribed attacking instance. Such an exploitation is still feasible to achieve [12], [34], since the security vulnerabilities of VMMs have been reported on a regular basis due to the complexity of VMMs.

Finally, packet header encryption techniques like IPsec [16] cannot prevent attackers from resetting the connection as attackers do not need to know the exact value

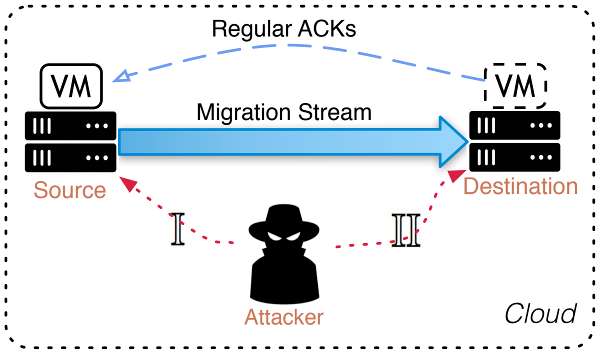
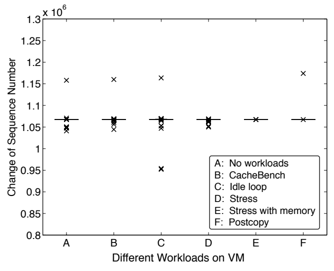
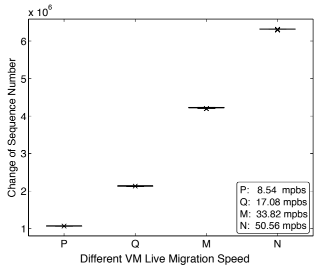


Fig. 3: Attack Scenarios.



(a) Migration Methods and Workloads.



(b) Migration Speed.

Fig. 4: Factors Affecting the Change of Sequence No. of VM Live Migration.

of the packet but send multiple crafted packets with one of them matches the correct information. Any high-level encryption solutions such as SSL/TLS do not help as they function in or above the transport layer.

3.2.3 Catch Timing and Four Tuples

Unlike a long-lived and idle TCP connection, VM live migration is a short procedure with heavy traffic. Thus, the first step for an efficient attack is to catch the timing of live migration. While several previous works [33], [15] have demonstrated multiple different methods (both internal and external) to uncover the timing of the migration process, we further introduce a new way to expose such timing information. As mentioned in Section 2, before the migration, the destination needs to enter into a ready-for-migration mode by listening to a specific port. This port is opened for accepting a migration request. One critical problem is that modern migration implementation ignores authentication, which means that anyone who knows the destination IP address and port number can initiate the migration.

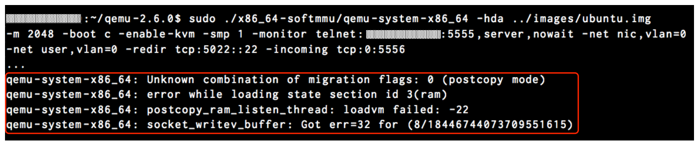
The attacker can launch fake migration requests (on the attacking machine) to all ports of the target server. In order to catch the timing, the attacker can conduct the scan on a regular basis. In most periods, the scanner should fail to establish any migration connections because the port is closed. If the attacker succeeds in building the migration connection with the target server, the source VM would be unable to initiate the migration, since all preparatory operations on the target would have been occupied by the fake migration. Once the attacker disconnects with the target server, the target VM simply crashes, which is similar to the reset attack on *pre-copy* VM live migration. Otherwise, if the attacker discovers a newly opened port but is unable to build the migration connection, he has high confidence that this port is opened for VM live migration. In this way, the attacker can obtain the timing as well as the target port number. Besides, similar approach or previously described method [7] could be utilized to explore the rest of the tuples.

3.2.4 Resetting the Connection

After confirming the four tuples of an ongoing migration, the attacker can start to reset the TCP connection by abusing the vulnerabilities in either the source server or the destination server. Previous research has shown multiple methods to obtain the correct sequence number by off-path attackers [8], [21], [22], [46], [45]. Here we mainly discuss the exploitation of TCP blind attack and the challenge ACK vulnerability. The identification of the vulnerability of the server is straightforward. Attackers can build a legitimate TCP connection with the target and attempt to terminate the connection using in-window RST packets. If the in-window RST packet can cut the connection, the attacker can launch a blind reset attack. Otherwise, if the target server only replies with 100 challenge ACKs per second, it means the server has a challenge ACK vulnerability.

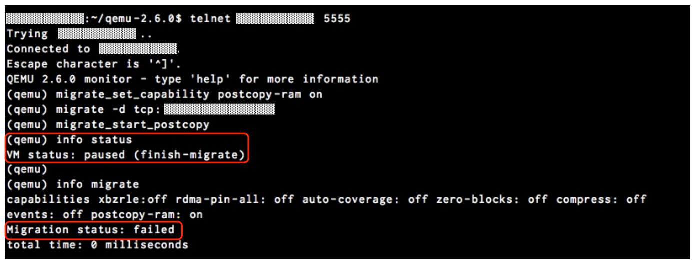
Blind reset attack. If the server does not deploy the implementation of TCP following the recommended standard RFC 5961, resetting the TCP connection is easy. An in-window RST packet can reset the connection. Hence, with the knowledge of four tuples of the connection, the attacker simply floods the RST packets with sequence numbers increased by the window size. Once the attacker can brute-force all the sequence number spaces, the connection is terminated. For example, suppose the window size is around 10,000, and the attacker needs to flood $2^{32}/10000 \approx 420,000$ RST packets. With 20,000 RST packets sent per second, a reset attack could be achieved in merely 21 seconds.

Side-channel reset attacks on the source. In the case that the source server is vulnerable to CVE-2016-5696, the attacker can conduct a reset attack on the source server following the previous work [7], as in the case of $\mathbb{I}$ illustrated in Figure 3. While VM live migration generates huge traffic from the source to the target, the backward traffic is relatively idle. The target merely replies with regular ACKs (which would not consume the limit of challenge ACKs) corresponding to the data packets. The sequence number of the backward traffic is invariable. We briefly list the attacking procedures here. The basic idea is to send multiple spoofed packets and 100 in-window non-spoofed packets (in the legitimate connection) per second. If the number of replied ACKs is less than 100, it means the spoofed packets trigger challenge ACKs. The detailed steps are: (1) Synchronize with the clock on the server to ensure that all packets sent from the attacker arrive within the same 1-second interval. (2) Identify the approximate sequence number range, which is the range of window size multiplied by the number of packets sent. (3) Narrow down the sequence number space to a single block (estimated window) through binary searching. Finally, (4) flood RST



```
[redacted]:~/qemu-2.6.0$ sudo ./x86_64-softmmu/qemu-system-x86_64 -hda ../images/ubuntu.img
-m 2048 -boot c -enable-kvm -smp 1 -monitor telnet:[redacted]:5555,server,nowait -net nic,vlan=0
-net user,vlan=0 -redir tcp:5022::22 -incoming tcp:0:5556
...
qemu-system-x86_64: Unknown combination of migration flags: 0 (postcopy mode)
qemu-system-x86_64: error while loading state section id 3(ram)
qemu-system-x86_64: postcopy_ram_listen_thread: loadvm failed: -22
qemu-system-x86_64: socket_writev_buffer: Got err=32 for (8/18446744073709551615)
```

(a) Migration Results (Destination).



```
[redacted]:~/qemu-2.6.0$ telnet [redacted] 5555
Trying [redacted]..
Connected to [redacted].
Escape character is '^]'.
QEMU 2.6.0 monitor - type 'help' for more information
(qemu) migrate_set_capability postcopy-ram on
(qemu) migrate -d tcp:[redacted]
(qemu) migrate_start_postcopy
(qemu) info status
VM status: paused (finish-migrate)
(qemu)
(qemu) info migrate
capabilities xbzrle:off rdma-pin-all: off auto-coverage: off zero-blocks: off compress: off
events: off postcopy-ram: on
Migration status: failed
total time: 0 milliseconds
```

(b) Migration Status (Source).

Fig. 5: Attack on Post-copy Live Migration.

packets in that block.

Side-channel reset attacks on the destination. Attacking the target server (as in the case of Ⅲ in Figure 3) is much more difficult than the source server case. The heavy traffic makes the sequence number vary rapidly, leaving the previous method [7] no chance to shrink the sequence number into a window after catching the approximate sequence number.

In order to the achieve the attack, we conduct two experiments to figure out the pattern of the change of sequence number in VM live migration. We run different workloads on the source VM. We then fix the throughput of migration to 8.54 Mbps, and start the migration with different migration approaches. Five workloads are chosen with the *pre-copy* option: (1) *Idle* without workloads; (2) *CacheBench* benchmark; (3) *Idle loop*; (4) *Stress* benchmark; and (5) *Stress* with memory configuration. One workload has the *post-copy* option: (6) *Idle*. We measure the change of sequence number in one second for a total of 50 instances. Figure 4(a) shows the boxplot result. As we see, except for few outliers, the change of sequence number is almost the same in all cases: around $10.6 * 10^6$ per second. In our second experiment, we set the speed limit of migration to 17.08 Mbps, 33.82 Mbps, and 50.56 Mbps. The results in Figure 4(b) demonstrate that the change of sequence number is linear to the speed limit. Overall, we have two major observations: (1) the change of sequence number is quite constant and only related to the migration speed; (2) The change of sequence number is irrelevant to the workloads running in the VM or migration approaches.

After understanding these two features, the attacker can launch the reset attack on migration. The basic idea is to obtain the rough speed of live migration and track the step that the sequence number changes per second at the same time. Specifically, the attacker can reset the migration with three procedures, as depicted in Algorithm 1. This algorithm is designed to reset a connection with heavy traffic.

The first two steps are similar to the attack on the source server: get synchronized with the target server, and identify the approximate sequence number range. After that,

Algorithm 1 Identify and stalk the correct sequence number range

```
 1: procedure SEQUENCE_NUMBER_STALKING($s'_l$, $s'_u$)
 2:     $step_o \leftarrow 5 \times (s'_u - s'_l)$;
 3:     $s_{u,l} \leftarrow s'_{u,l} + step_o$;
 4:     $range \leftarrow (s_u - s_l)$;
 5: Step profiling:
 6:     while $range\_hit()$ do
 7:         $count \leftarrow count + 1$;
 8:         wait(1);
 9:     $step \leftarrow step_o/count$;
10:     $s_{u,l} \leftarrow s_{u,l} + step$;
11: Micro tuning:
12:     while $range\_unmeet()$ do
13:         if $range\_hit()$ then
14:             binary_cut_range();
15:         else if $range\_miss()$ then
16:             left_shift_range();
17:             if $range\_miss()$ for 3 times then
18:                 adjust_step();
19:                 if $adjust\_step()$ for 3 times then
20:                     $s_{u,l} \leftarrow s'_{u,l} + step_o$;
21:                     $step \leftarrow step_o/(count + 1)$;
22:                     goto Micro tuning.
23:         $s_{u,l} \leftarrow s_{u,l} + step$;
24:         wait(1);
25: Connection reset:
26:     for i in 20 do
27:         flood($s_l, s_u$)
28:         wait(1);
29:     if failed then
30:         goto Micro tunning.
```

assuming the attacker sends 4,000 packets per second with a block size 10,000, the range is 40,000,000. Then the attacker can increase the sequence numbers of those probe packets with a relatively large size (e.g., 5 times the range), and simply waits for the next hit, as described in Lines 5∼10. This procedure, *step profiling*, could obtain an approximate value of the speed by dividing the increased size by time consumption.

Lines 11∼24 present the next procedure, *micro tuning*. This step is the most critical one, and has two goals: (1) this step is responsible for adjusting the speed as accurate as possible; (2) after this step, the attacker should narrow down the range to an acceptable small value, so that he is able to flood RST packets to cover all the sequence numbers in that range. To achieve those two goals, the attacker can first assume that the estimated speed is smaller than the actual speed. Then, every time the sequence number is hit in the range, the attacker cuts the range to the second half part (Lines 13∼14), which could promise a future hit if the assumption is correct. If the sequence number is not hit in the next second, the attacker slightly shifts the range to the left (Lines 15∼16). If the sequence number still cannot hit within the range in a few times (e.g., 3), the attacker can then slightly reduce the estimated speed (e.g., 1%), as shown in Lines 17∼18. If the hit does not come even after

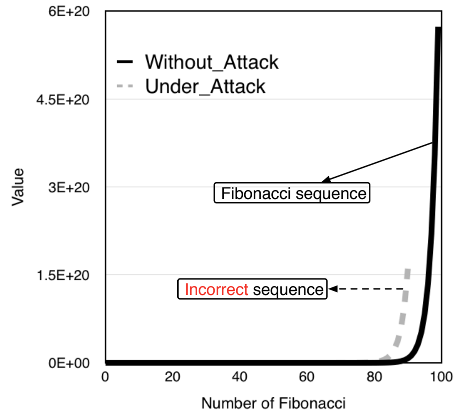


Fig. 6: Memory Inconsistence: Incorrect Fibonacci Number.

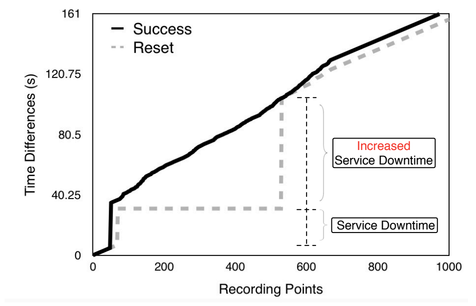


Fig. 7: Downtime Increases.

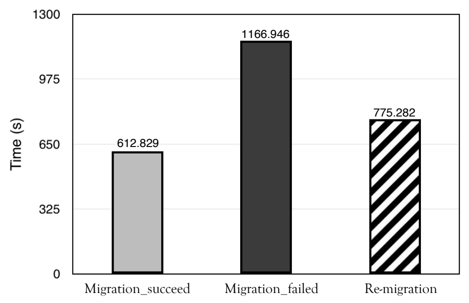


Fig. 8: Performance Degradation.

several speed adjustments, it is highly probable that the assumption is incorrect. The attacker should recalculate the estimated speed, and repeat the *micro tuning* (Lines 19∼22). The procedure is repeatedly conducted until the range is cut to a 2-4 block size.

The last procedure is *connection reset*, in which the attacker can flood RST packets with brute-force all possible sequence numbers. The attacker may need several tries to reset the connection, since the RST packet needs to hit the exactly correct sequence number. Another issue for this step is that errors exist between the estimated speed and actual speed. After a few trials, the stalk might be lost. Under such scenarios, the attacker needs to restart the next round of *micro tuning* to resume the tracking of the correct sequence number.

4 EVALUATIONS

In this section, we present the evaluation from three aspects: (1) the attacking consequences on VM live migration in existing hypervisors; (2) the attacking effects on the running VM; and (3) the effectiveness of our proposed reset algorithm.

4.1 Attacking Consequences

We first demonstrate that the TCP reset attack can cause the devastating consequences we expect. Our experiments are conducted on QEMU version 2.6. We also test QEMU 2.8 and 2.9, which give similar results. We use two Ubuntu servers with Linux kernel 4.4 with independent, public IP addresses. Our destination server is equipped with an Intel Xeon CPU E5-2690 with 2.60GHz frequency, and the source server is equipped with an Intel Xeon W3550 CPU.

We first start our modified malicious QEMU (scanner) to periodically monitor the targeted server (destination) with one-minute-intervals. Then we initiate the setup preparation on both source and destination servers and start the live migration with the *post-copy* approach. If our scanner successfully establishes the migration connection with the target, the source server would be failed on the migration connection. The VM on the target would be crashed if our scanner breaks the connection. Otherwise, we launch the TCP-RST attack on the destination server using our proposed algorithm. The migration runs into errors immediately once the attack succeeds. The destination VM is crashed immediately, which is demonstrated in Figure 5(a). In some cases, the destination VM would not be crashed, but totally unresponsive. Figure 5(b) shows the status of the source VM. We can clearly see that the migration status is failed. Also, the source VM is on a paused state waiting for the migration to finish. The result of pre-copy is quite similar. The difference is that the source VM is still running after the attack, but the target VM is crashed.

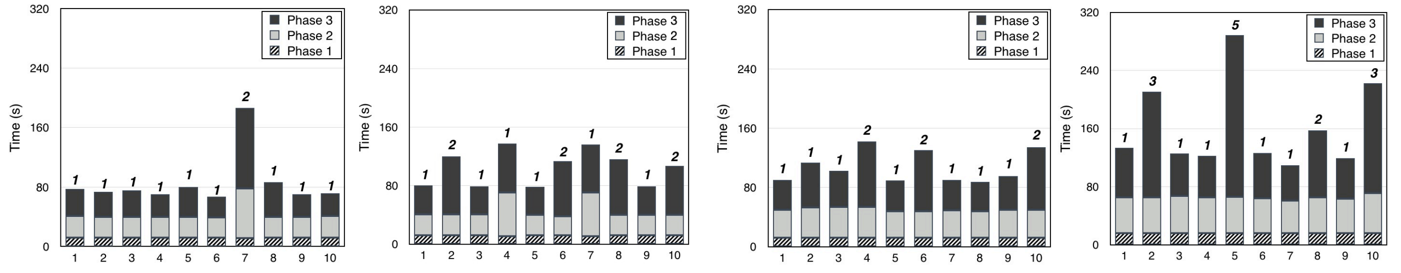
4.2 Attacking Impact

We conduct the TCP reset attack in three different scenarios to demonstrate that resetting the VM live migration could lead to (1) memory inconsistence on the victim VM; (2) significant increases to the service downtime; and (3) performance downgrade on the application on the victim VM. To ensure that the connection is reset at the same time for a fair comparison, we use *tcpdump* to get the correct backward sequence number (from the destination to the source) and reset the connection in all three cases.

4.2.1 Memory inconsistence

To expose the memory inconsistence problem, we build a simple application to simulate the online transaction system. The application calculates the Fibonacci number using $F_n = F_{n-1} + F_{n-2}$. The source VM only holds the initial value 0. We create a toy server that listens for incoming requests. For every two seconds, the victim VM sends a request to the toy server asking for the next value F_{n-1}. Based on a counter (used as the index) maintained in the toy server, the server sends the value to our victim VM.

The request from the victim VM guarantees that the application would have no action when the victim is down in the downtime period. Also, the toy server replies with exactly 100 requests. The victim VM keeps the generated sequence in memory until the toy server no longer replies to the next corresponding value. We present the results on *post-copy*-based live migration, which suffers from the memory inconsistence problem.



(a) 8.54 Mpbs. (b) 17.08 Mpbs. (c) 33.82 Mpbs. (d) 50.56 Mpbs.

Fig. 9: Attacks on VM Live Migration with different throughput.

For the attack case, after the victim VM starts calculating the Fibonacci sequence, we launch the TCP reset attack. At this time, the victim VM on the source server is paused. The VM immediately runs in the destination server and keeps sending requests to our toy server. Once the reset attack succeeds, the VM on the destination is crashed. We modify the hypervisor so that it initiates another migration request immediately once observing the failure of the migration.

As demonstrated in Figure 6, the generated sequence when the VM is under a TCP reset attack is represented by the grey dotted line. When the victim VM is paused, it only obtained the value 5, which is the fifth Fibonacci number. Then the crashed VM in the first migration keeps sending requests to the toy server, and obtains the 15th number. After this VM is crashed, the re-migration soon opens a new VM running in the destination server. However, the memory of this new VM stays at the fifth Fibonacci number and begins to calculate the sixth one. From the perspective of the toy server, it treats the VM as calculating the 16th number because the requests sent by the crashed VM consumes 10 counts. As a result, it sends the value 610 to the server. The calculated value in the running VM is 615, which is wrong. Even worse, since the next number is computed by this incorrect "Fibonacci" value, all of the following results are also incorrect due to the memory inconsistence. As we see, compared with the black solid line representing the correct Fibonacci number, the number of the generated sequence under attack is obviously less than 100. Also, all calculated numbers after the fifth one are incorrect. While this is just a simple application, the problem could be much more serious in realistic applications, such as online trading.

4.2.2 Downtime Increases

In this experiment, we write a program to repeatedly obtain the difference between the current time and the time that the program is started. The program runs on the victim VM and constantly writes the time difference into a file. We use *post-copy* as the migration method. For *pre-copy*, if the attack succeeds when the source VM is down, the result is similar. The re-migration happens immediately once the failure of migration is detected. Figure 7 presents the results. Without an attack, the program resumes working at 35s after the migration. Under a TCP-RST attack, although the migration happens almost at the same time (about 5s), the program is resumed at 104s. The downtime under attack is more than three times compared with a successful migration. Note that the re-migration starts immediately in this experiment. Otherwise, the increase of downtime could be much more serious.

4.2.3 Performance Impact

We use the time of compiling the Linux kernel to evaluate the performance. We run some background workloads in the source server to simulate the resource contention in the cloud environment. The workload on the source is heavy, and the VM is needed to migrate to another server. We choose the *pre-copy* approach since the VM is still running, even while under attack. We measure the time for compiling the Linux kernel 4.4 on three scenarios: (1) The VM successfully migrates to the destination server, and (2) The migration is failed due to the TCP reset attack. As a result, the victim VM is kept running in the source server; (3) the migration fails because of the attack, and a re-migration follows immediately.

We present the time consumption for all three scenarios in Figure 8. If the migration is failed, the time for compiling the kernel is about 1,169 seconds, since the VM is always running in the source server suffering heavy resource contentions. If the migration succeeds, running it in the destination could save massive amounts of time (about 500 seconds). The re-migration incurs a worse performance compared to the successful migration since the VM is forced to run longer in the source server.

4.3 Effectiveness of the Reset Attack

As we mentioned before, if an attacker can send 20,000 RST packets per second, several seconds would be enough to reset a TCP connection in a server whose TCP implementation does not follow RFC 5961. We have also demonstrated that such a requirement is easily fulfilled in Section 3. Here, we conduct experiments to investigate the effectiveness of launching TCP reset attack on the destination server.

The first two procedures (synchronization and finding the approximate sequence number range) are the same as previous work [7], which can be achieved in less than one minute. We do not repeat similar experiments. Instead, we measure the time consumption for the proposed algorithm (Algorithm 1). We start recording the time once the approximate sequence number range is identified. We initiate the migration with the *pre-copy* option and conduct experiments under four different throughput scenarios: 8.54 Mpbs, 17.08 Mbps, 33.82 Mbps, and 50.56 Mbps. For each speed, we launch the reset attack 10 consecutive times, and each attack does not stop until it successfully resets the connections. To infer the approximate sequence number range, we send 4,000 crafted RST packets per second.

We illustrate the results of our experiments in Figure 9. We break down the time consumption for each phase: step profiling (phase 1), micro tuning (phase 2), and connection reset (phase 3). As we mentioned in Section 3, one round of phase 3 might be failed to reset the connection due to the heavy traffic. We list the number of rounds in phase 3 above each bar. The total time is less than 80 seconds when the throughput is 8.54 mpbs. Also, the difficulty to reset the connection is raised as the throughput increases: the average time to reset a migration with a 50.56 mpbs throughput exceeds two minutes. Since the total migration time lasts tens of minutes (migrating the disk involves more time), attackers would be able to reset the migration connection. In particular, phase 1 takes about 10 seconds to roughly profile the step size of the variation of sequence numbers. Phase 2 consumes much more time to refine the speed and narrow down the range of sequence numbers. In some cases (as the seventh bar in Figure 9(a)), the attacker needs to recalculate the estimated speed in phase 1, and restarts phase 2, which almost doubles the time. In the last phase, resetting the connection by flooding RST packets requires a little luck. While sometimes it merely takes seconds to succeed, it takes at most five rounds to reset the migration when the throughput is 50.56 mpbs. Though our experiments are limited by the bandwidth of our servers, the results demonstrate that enhancing the speed of migration could make the migration safer to some extent.

5 Potential Mitigation Approaches

The unrecoverable consequences in the post-copy based live migration are caused by those newly updated memory pages in the destination node (those pages remain unchanged in the source). Thus, the termination of the underlying connection (e.g., a TCP connection) will cause the memory inconsistency problem. A complete solution involves an enhancement (e.g., simultaneously updating the memory pages at the source by adding multiple checkpoints) or even a redesign on the post-copy based live migration. We leave the detailed design and implementation as our future work. In the following, we discuss several potential mitigation approaches from different perspectives.

A complete fix involves an enhancement (e.g., simultaneously updating the memory pages at the source by adding multiple checkpoints) or even a redesign on the post-copy based live migration. We leave the detailed design and implementation as our future work. In the following, we discuss several potential mitigation approaches from different perspectives.

The first method of increasing the difficulty to mount such attacks is to hide the timing information of live migration. For example, blocking attackers from probing specific ports would largely mitigate such threats in realistic scenarios. The enforcement of authentication on live migration can also greatly reduce the attack surface. However, in practice, since live migration will inevitably incur system overheads and thus degrade the server's performance, a few techniques have been proposed to detect the migration process [33], [15]

An intuitive approach to avoiding the unrecoverable negatives caused by unexpected interruptions is to re-establish the TCP connection. The migration could be resumed immediately instead of rebooting the instance or re-setting the whole migration step. This approach, which can effectively prevent the memory inconsistency problem, involves non-trivial engineering efforts. First, after receiving the RST packet from an attacker, the server hosting the source VM simply closes the TCP connection. To resume the migration process, a new TCP connection must be established. This is totally different from resuming a hang connection due to poor networking conditions. Even worse, the target server is unable to determine if the connection is closed or suffers from packet losses caused by a bad networking condition. From the perspective of the target server, it cannot know the termination of the connection, and hangs in a blocking state until the *TCP keepalives* timeout is reached. Also, the target server is unable to differentiate cases in which the connection is closed or suffering from a bad networking environment where packets are simply lost. Those cases make it difficult to design a valid auto re-connection mechanism. Second, the migration process must be careful to record the copying step of memory pages on both source and target sides. From the perspective of the source server, the number of accepted *packets on-the-fly* is unknown. A naive re-transmitting strategy might lead to overlaps or losses of memory, which would crash the target VM. Finally, re-establishing the TCP connection does not fundamentally solve the problem. Malicious attackers can keep attacking the migration and terminate the re-connected TCP connection. The results could significantly increase VM downtime or degrade performance on both sides.

Another strategy is ignoring the TCP RST packets when the migration is in progress. By discarding the RST packet, attackers are unable to cut the TCP connection by crafting reset packets. To reduce the abjective effects, the prohibition of TCP RST packets should be restricted within the specific migration connection. This could be achieved by recording the four tuples, `<source IP address, source port number, destination IP address, destination port address>`. Once the data transfer starts, the hypervisor can block the RST packets by setting a firewall rule (e.g., setting an `iptables` rule), which does not require any change to the kernel code. One issue is that `iptables` suffers a significant performance penalty [18]. Particularly, if multiple migrations are conducted simultaneously, each networking packet, including those non-RST packets, must be checked by all `iptables` rules, one by one.

Another option is to reply a challenge ACK when the RST packet hits the exactly correct sequence number. Otherwise, the TCP works as usual. Such a method can maintain the integrity of the original TCP. Also, it only affects the performance of processing RST packets. However, such a method involves the modification of the kernel.

We implement the defense mechanism by preventing the reset of the TCP connection for the migration stream. We modify the Linux kernel with version 4.4 and QEMU 2.6. We create a pseudo file in *procfs* to record the four tuples of the migration connection. The pseudo file is mounted with root permission so that only the system administrator can view and modify it. The four tuples are recorded when the migration connection is established, and is maintained in a map form. During the migration, instead of resetting the connection, the incoming RST packet with the correct sequence number would trigger a challenge ACK. All other functionalities of TCP are left unchanged. Obviously, the effects of the defense are just like the case without a TCP reset attack. In our evaluation experiments, our implementation works as expected to provide effective protection.

6 Related Work

In this section, we list some research efforts that inspire our work and compare our work with previous research. We mainly discuss research works in the following three areas.

6.1 VM Live Migration

VM live migration with *pre-copy* was first proposed in 2005 [10]. Then, several new approaches based on *pre-copy* were proposed to improve performance, including the Fast Transparent Migration [40], Delta Compression Techniques [53], Sandpiper [59], and MiG [48]. In addition, Liu et al. [36] designed a novel approach by adopting checkpointing/recovery and trace/replay technology. Instead of transferring the dirty memory pages, their method transfers the execution log of the source VM, which can save up to 31.5% on the total migration time. Hou et al. [26] designed an application-assisted live migration framework that migrates VMs running Java applications by skipping the transfer of garbage in Java memory. Their approach can save up to over 90% downtime. Voorsluys et al. [57] studied the impact of VM live migration on the performance of applications running inside the Xen hypervisor, and demonstrated an acceptable overhead incurred by the migration process. Then, Hines [23], [24] presented the design, implementation, and evaluation of post-copy VM live migration in 2009. Ali [37] designed XvMotion, allowing VM migration over long distances. Instead of improving the performance, our work aims to avoid performance downgrades by fixing potential vulnerabilities.

While major research efforts have been put into enhancing the performance of VM live migration, few works focus on the security of VM migration. Perez et al. [44] presented a brief introduction on the potential threats suffered by VM live migration. The prerequisite for attacking VM live migration is the ability to detect the migration process. König et al. [33] showed that the round trip time of ICMP packets could be utilized by outside attackers to detect the migration process. Fiebig [15] further demonstrated that the migration process could be detected by a combination of delay measurements by ICMP pings and time-lag detection with the NTP. In particular, they presented several internal approaches for detecting the process since VM live migration affects a server's performance. While we present a novel approach for exploiting flaws in existing hypervisors' implementations, the above methods are complementary for us to confirm the detection result.

Another serious issue is the lack of encryption in VM live migration. Several works have been proposed on both sides of the issue [65], [43]. Oberheide et al. [41] developed a tool to perform man-in-the-middle attacks on the VM live migration by manipulating the memory on the network. As a result, encryption has been introduced in VM live migration, and KVM has supported live migration on TLS. However, despite those protections, TCP-RST attacks occur in the transportation layer, and thus can still cause the unrecoverable effects on VM live migration.

6.2 TCP Attacks

The security of TCP, especially in off-path attacks, always receives much attention. Gilad et al. [21] performed several off-path attacks on based on a global IP-ID counter that allows an attacker to learn the sequence numbers of the client and server in a TCP connection. Several works [46], [45], [8] abuse OS packet counters to determine whether the guessed sequence number is correct. Gilad et al. [22] also proposed techniques to infer the TCP connection and four-tuples between two hosts. While those techniques could be utilized to attack VM live migration, we leverage the blind reset attack and TCP challenge ACK vulnerability in this paper. In addition to cracking the sequence number, there are also several other security issues for TCP. Alexander et al. [3] presented a novel technique for estimating the RTT latency between two off-path hosts. Ensafi et al. [14] showed that it is even possible for an attacker to port scan a network from outside the firewall. Abramov et al. [2] presented ACK-Storm DoS attacks that could reach a level of 400,000 amplifications against popular websites. Different from those works focusing on vulnerabilities in TCP protocol, we study the devastating consequences on VM caused by TCP reset attacks.

6.3 Denial-of-Service Attacks on Clouds

DoS attacks on clouds are also closely related to our work. For instance, multiple attacks [64], [35], [17], [19], [20], [30], [31], [29] can cause exhaustion of the shared infrastructure resources in data centers, and lead to forced shutdowns for servers on the same rack or on the same power distribution unit. Varadarajan et al. proposed resource-freeing attacks [54], in which attackers can gain extra utilization by freeing up resources used by victims's instances. Zhang et al. [67] showed that memory DoS attacks can cause 38$\times$ delay for an E-commerce website. Huang et al. [27] proposed cascading performance attacks to exhaust a hypervisor's I/O processing capability. For those DoS attacks, attackers usually exploit specific techniques to co-locate multiple controlled instances on the same physical server to amplify the attack's effects. Techniques like cache [62], memory bus [60], networking [50], and system process statistics [55] have been proved to achieve co-residence in clouds. Furthermore, attackers can achieve a malicious instance co-resident with the target VM and launch side-channel attacks. Zhang et al. [69], [70] showed several approaches for extracting private keys and collecting potentially sensitive application data on co-resident instances. Although multiple defense mechanisms have also been proposed [71], [61], [68], [9], [66], [72], two previous works [63], [56] show that it is still practical (and cheap) to achieve co-residence in existing mainstream cloud services. While our proposed attack does not require co-resident instance with the target VM, we believe co-residence techniques could be helpful since it can provide insider information. We plan to explore in this direction in the future. Also, our work demonstrates that resetting the TCP connection of post-copy migration could generate similar effects of a power outage.

7 Conclusion

Virtual machine live migration has become popular because of its diverse use-cases for cloud vendors and power-users alike. It offers a scalable mechanism to maintain low-level systems, manage faults and balance workloads among different physical servers. In this paper, we demonstrate that by disrupting the robustness of the underlying network substrate using a successful TCP reset attack, an adversary can cause unrecoverable memory inconsistency problems. This is especially true for *post-copy*-based migration approaches. In addition, terminating the TCP connection could also cause significant service downtime and affect the running VM's performance. We further detail a procedure for resetting the migration connection utilizing heavy traffic. This procedure includes a novel technique to measure and expose the timing of the migration process, and a new method to off-path attack the TCP connection. Our evaluation demonstrates that the attack is effective and able to cause the expected devastating consequences.

References

[1] Big Increase in DDoS Attacks. https://mybroadband.co.za/news/security/157383-big-increase-in-ddos-attacks.html.

[2] R. Abramov and A. Herzberg. TCP Ack storm DoS attacks. *Computers & Security*, 2013.

[3] G. Alexander and J. R. Crandall. Off-Path Round Trip Time Measurement via TCP/IP Side Channels. In *IEEE INFOCOM*, 2015.

[4] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana. Towards virtual machine migration in fog computing. In *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 2015.

[5] N. Bobroff, A. Kochut, and K. Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *IFIP/IEEE IM*, 2007.

[6] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. In *ACM VEE*, 2007.

[7] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel. Off-Path TCP Exploits: Global Rate Limit Considered Dangerous. In *USENIX Security*, 2016.

[8] Q. A. Chen, Z. Qian, Y. J. Jia, Y. Shao, and Z. M. Mao. Static Detection of Packet Injection Vulnerabilities: A Case for Identifying Attacker-controlled Implicit Information Leaks. In *ACM CCS*, 2015.

[9] S. Chen, F. Liu, Z. Mi, Y. Zhang, R. B. Lee, H. Chen, and X. Wang. Leveraging Hardware Transactional Memory for Cache Side-Channel Defenses. In *ACM AsiaCCS*, 2018.

[10] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *USENIX NSDI*, 2005.

[11] T. Das, P. Padala, V. N. Padmanabhan, R. Ramjee, and K. G. Shin. LiteGreen: Saving Energy in Networked Desktops Using Virtualization. In *USENIX ATC*, 2010.

[12] N. Elhage. Virtunoid: Breaking Out of KVM. *Black Hat USA*, 2011.

[13] R. Ensafi, J. Knockel, G. Alexander, and J. R. Crandall. Detecting Intentional Packet Drops on the Internet via TCP/IP Side Channels. In *Springer PAM*, 2014.

[14] R. Ensafi, J. C. Park, D. Kapur, and J. R. Crandall. Idle Port Scanning and Non-interference Analysis of Network Protocol Stacks Using Model Checking. In *USENIX Security*, 2010.

[15] S. Fiebig, M. Siebenhaar, C. Gottron, and R. Steinmetz. Detecting VM Live Migration using a Hybrid External Approach. In *CLOSER*, 2013.

[16] S. Frankel and S. Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. In *RFC 6071*.

[17] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang. ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds. In *IEEE DSN*, 2017.

[18] X. Gao, Z. Gu, Z. Li, H. Jamjoom, and C. Wang. Houdini's Escape: Breaking the Resource Rein of Linux Control Groups. In *ACM CCS*, 2019.

[19] X. Gao, B. Steenkamer, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang. A study on the security implications of information leakages in container clouds. *IEEE Transactions on Dependable and Secure Computing*, 2018.

[20] X. Gao, Z. Xu, H. Wang, L. Li, and X. Wang. Reduced Cooling Redundancy: A New Security Vulnerability in a Hot Data Center. In *NDSS*, 2018.

[21] Y. Gilad and A. Herzberg. Off-Path Attacking the Web. In *USENIX WOOT*, 2012.

[22] Y. Gilad and A. Herzberg. Spying in the Dark: TCP and Tor Traffic Analysis. In *PETS*, 2012.

[23] M. R. Hines, U. Deshpande, and K. Gopalan. Post-Copy Live Migration of Virtual Machines. *ACM SIGOPS Operating Systems Review*, 2009.

[24] M. R. Hines and K. Gopalan. Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning. In *ACM VEE*, 2009.

[25] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi. Reactive Consolidation of Virtual Machines Enabled by Postcopy Live Migration. In *ACM VTDC*, 2011.

[26] K.-Y. Hou, K. G. Shin, and J.-L. Sung. Application-Assisted Live Migration of Virtual Machines with Java Applications. In *ACM EuroSys*, 2015.

[27] Q. Huang and P. P. Lee. An Experimental Study of Cascading Performance Interference in a Virtualized Environment. *ACM SIGMETRICS*, 2013.

[28] K. Z. Ibrahim, S. Hofmeyr, C. Iancu, and E. Roman. Optimized Pre-Copy Live Migration for Memory Intensive Applications. In *SC*, 2011.

[29] M. A. Islam and S. Ren. Ohm's Law in Data Centers: A Voltage Side Channel for Timing Power Attacks. In *ACM CCS*, 2018.

[30] M. A. Islam, S. Ren, and A. Wierman. Exploiting a Thermal Side Channel for Power Attacks in Multi-Tenant Data Centers. In *ACM CCS*, 2017.

[31] M. A. Islam, L. Yang, K. Ranganath, and S. Ren. Why Some Like It Loud: Timing Power Attacks in Multi-Tenant Data Centers Using an Acoustic Side Channel. *ACM SIGMETRICS*, 2018.

[32] P. Kokkinos, D. Kalogeras, A. Levin, and E. Varvarigos. Survey: Live migration and disaster recovery over long-distance networks. *ACM Computing Surveys*, 2016.

[33] A. König and R. Steinmetz. Detecting Migration of Virtual Machines. In *EuroView 2011*, 2011.

[34] K. Kortchinsky. Cloudburst: A VMware Guest to Host Escape Story. *Black Hat USA*, 2009.

[35] C. Li, Z. Wang, X. Hou, H. Chen, X. Liang, and M. Guo. Power Attack Defense: Securing Battery-Backed Data Centers. In *IEEE ISCA*, 2016.

[36] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu. Live Migration of Virtual Machine Based on Full System Trace and Replay. In *ACM HPDC*, 2009.

[37] A. J. Mashtizadeh, M. Cai, G. Tarasuk-Levin, R. Koller, T. Garfinkel, and S. Setty. XvMotion: Unified Virtual Machine Migration over Long Distance. In *USENIX ATC 14*, 2014.

[38] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive Fault Tolerance for HPC with Xen Virtualization. In *ACM ICS*, 2007.

[39] R. Nathuji and K. Schwan. Virtualpower: Coordinated Power Management in Virtualized Enterprise Systems. In *ACM SOSP*, 2007.

[40] M. Nelson, B.-H. Lim, G. Hutchins, et al. Fast Transparent Migration for Virtual Machines. In *USENIX ATC*, 2005.

[41] J. Oberheide, E. Cooke, and F. Jahanian. Empirical Exploitation of Live Virtual Machine Migration. In *Proc. of BlackHat DC convention*, 2008.

[42] O. Osanaiye, S. Chen, Z. Yan, R. Lu, K.-K. R. Choo, and M. Dlodlo. From cloud to fog computing: A review and a conceptual live vm migration framework. *IEEE Access*, 2017.

[43] V. P. Patil and G. Patil. Migrating Process and Virtual Machine in the Cloud: Load Balancing and Security Perspectives. *International Journal of Advanced Computer Science and Information Technology*, 2012.

[44] D. Perez-Botero. A Brief Tutorial on Live Virtual Machine Migration From a Security Perspective. *Princeton University, USA*, 2011.

[45] Z. Qian and Z. M. Mao. Off-Path TCP Sequence Number Inference Attack-How Firewall Middleboxes Reduce Security. In *IEEE S&P*, 2012.

[46] Z. Qian, Z. M. Mao, and Y. Xie. Collaborative TCP Sequence Number Inference Attack: How to Crack Sequence Number Under a Second. In *ACM CCS*, 2012.

[47] A. Quach, Z. Wang, and Z. Qian. Investigation of the 2016 Linux TCP Stack Vulnerability at Scale. In *ACM SIGMETRICS*, 2017.

[48] A. Rai, R. Ramjee, A. Anand, V. N. Padmanabhan, and G. Varghese. MiG: Efficient Migration of Desktop VMs Using Semantic Compression. In *USENIX ATC*, 2013.

[49] A. Ramaiah, R. Stewart, and M. Dalal. Improving TCP's Robustness to Blind In-Window Attacks. Technical report, 2010.

[50] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *ACM CCS*, 2009.

[51] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato. Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control. *IEEE Transactions on Computers*, 2017.

[52] A. Singh, M. Korupolu, and D. Mohapatra. Server-Storage Virtualization: Integration and Load Balancing in Data Centers. In *ACM/IEEE SC*, 2008.

[53] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines. In *ACM VEE*, 2011.

[54] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift. Resource-Freeing Attacks: Improve Your Cloud Performance (At Your Neighbor's Expense). In *ACM CCS*, 2012.

[55] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. Swift. A Placement Vulnerability Study in Multi-Tenant Public Clouds. In *USENIX Security*, 2015.

[56] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. Swift. A Placement Vulnerability Study in Multi-Tenant Public Clouds. In *USENIX Security*, 2015.

[57] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *IEEE Cloud*, 2009.

[58] P. Watson. Slipping in the Window: TCP Reset attacks. 2004.

[59] T. Wood, P. J. Shenoy, A. Venkataramani, M. S. Yousif, et al. Black-box and Gray-box Strategies for Virtual Machine Migration. In *USENIX NSDI*, 2007.

[60] Z. Wu, Z. Xu, and H. Wang. Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In *USENIX Security*, 2012.

[61] Q. Xiao, M. K. Reiter, and Y. Zhang. Mitigating Storage Side Channels Using Statistical Privacy Mechanisms. In *ACM CCS*, 2015.

[62] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An Exploration of L2 Cache Covert Channels in Virtualized Environments. In *ACM CCSW*, 2011.

[63] Z. Xu, H. Wang, and Z. Wu. A Measurement Study on Co-Residence Threat Inside the Cloud. In *USENIX Security*, 2015.

[64] Z. Xu, H. Wang, Z. Xu, and X. Wang. Power Attack: An Increasing Threat to Data Centers. In *NDSS*, 2014.

[65] F. Zhang and H. Chen. Security-Preserving Live Migration of Virtual Machines in the Cloud. *Journal of Network and Systems Management*, 2013.

[66] T. Zhang, Y. Zhang, and R. B. Lee. Cloudradar: A Real-Time Side-Channel Attack Detection System in Clouds. In *Springer RAID*, 2016.

[67] T. Zhang, Y. Zhang, and R. B. Lee. DoS Attacks on Your Memory in Cloud. In *ACM AsiaCCS*, 2017.

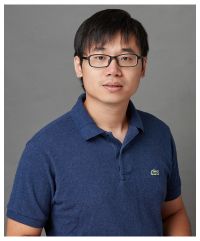
[68] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis. In *IEEE S&P*, 2011.

[69] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-VM Side Channels and Their Use to Extract Private Keys. In *ACM CCS*, 2012.

[70] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-tenant side-channel attacks in PaaS clouds. In *ACM CCS*, 2014.

[71] Y. Zhang and M. K. Reiter. Düppel: Retrofitting Commodity Operating Systems to Mitigate Cache Side Channels in the Cloud. In *ACM CCS*, 2013.

[72] Z. Zhou, M. K. Reiter, and Y. Zhang. A Software Approach to Defeating Side Channels in Last-Level Caches. In *ACM CCS*, 2016.



Xing Gao received the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA, USA, in 2018. He is an Assistant Professor in the Department of Computer Science at the University of Memphis, Memphis, TN, USA. His research interests include security, cloud computing, and mobile computing.

Jidong Xiao received his PhD degree in Computer Science from the College of William and Mary, Williamsburg, in 2016. He is an assistant professor at Boise State University. His research focuses on cybersecurity, with a particular emphasis on operating system security and cloud security.

Haining Wang received his Ph.D. in Computer Science and Engineering from the University of Michigan at Ann Arbor in 2003. He is a Professor of Electrical and Computer Engineering at the University of Delaware, Newark, DE. His research interests lie in the areas of security, networking system, and cloud computing.

Angelos Stavrou received the PhD degree in computer science from the Columbia University in 2007. He is an associate professor at the Department of Computer Science of George Mason University. His research interests include large systems security & survivability, intrusion detection systems, privacy & anonymity, security for MANETs and mobile devices.