2018

Investigating Emerging Security Threats in Clouds and Data Centers

Xing Gao
College of William and Mary - Arts & Sciences, xgao01@email.wm.edu

Investigating Emerging Security Threats in Clouds and Data Centers

Xing Gao

Xiangtan, Hunan, China

Bachelor of Science, Beijing Institute of Technology, 2011

A Dissertation presented to the Graduate Faculty
of The College of William & Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

College of William & Mary
August, 2018

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

Xing Gao

Xing Gao

Reviewed by the Committee, August 2018

Committee Chair
Professor Haining Wang, Electrical and Computer Engineering
University of Delaware

Professor Qun Li, Computer Science
College of William & Mary

Professor Evgenia Smirni, Computer Science
College of William & Mary

Associate Professor Gang Zhou, Computer Science
College of William & Mary

Dr. Zhongshu Gu,
IBM Research

ABSTRACT

Data centers have been growing rapidly in recent years to meet the surging demand of cloud services. However, the expanding scale of a data center also brings new security threats. This dissertation studies emerging security issues in clouds and data centers from different aspects, including low-level cooling infrastructures and different virtualization techniques such as container and virtual machine (VM).

We first unveil a new vulnerability called reduced cooling redundancy that might be exploited to launch thermal attacks, resulting in severely worsened thermal conditions in a data center. Such a vulnerability is caused by the wide adoption of aggressive cooling energy saving policies. We conduct thermal measurements and uncover effective thermal attack vectors at the server, rack, and data center levels. We also present damage assessments of thermal attacks. Our results demonstrate that thermal attacks can negatively impact the thermal conditions and reliability of victim servers, significantly raise the cooling cost, and even lead to cooling failures. Finally, we propose effective defenses to mitigate thermal attacks.

We then perform a systematic study to understand the security implications of the information leakage in multi-tenancy container cloud services. Due to the incomplete implementation of system resource isolation mechanisms in the Linux kernel, a spectrum of system-wide host information is exposed to the containers, including host-system state information and individual process execution information. By exploiting such leaked host information, malicious adversaries can easily launch advanced attacks that can seriously affect the reliability of cloud services. Additionally, we discuss the root causes of the containers' information leakage and propose a two-stage defense approach. The experimental results show that our defense is effective and incurs trivial performance overhead.

Finally, we investigate security issues in the existing VM live migration approaches, especially the post-copy approach. While the entire live migration process relies upon reliable TCP connectivity for the transfer of the VM state, we demonstrate that the loss of TCP reliability leads to VM live migration failure. By intentionally aborting the TCP connection, attackers can cause unrecoverable memory inconsistency for post-copy, significantly increase service downtime, and degrade the running VM's performance. From the offensive side, we present detailed techniques to reset the migration connection under heavy networking traffic. From the defensive side, we also propose effective protection to secure the live migration procedure.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

First I thank my doctoral advisor Dr. Haining Wang. His patient guidance and constructive advice helped me obtain all my research outcomes. His enthusiasm in pursuing exciting research and rigorous academic attitude greatly impact my attitude towards research and even life. Also, without his kind and unconditional support, it is impossible for me to finally go through my years in the graduate school. It has been my greatest honor to have him as my advisor.

Next I thank Drs. Gang Zhou, Evgenia Smirni, Qun Li, and Zhongshu Gu for serving on my dissertation committees.

I thank all my coauthors, Drs. Xiaorui Wang, Kun Sun, Angelos Stavrou, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, Zhang Xu, Dachuan Liu, Daiping Liu, Jidong Xiao, Mingwei Zhang, Bo Dong, Li Li, and Lei Lu, for their insightful suggestions, valuable feedbacks, and tremendous efforts.

I thank all professors who instructed me in their classes at William and Mary: Drs. Weizhen Mao, Phil Kearns, Peter Kemper, Xipeng Shen, Evgenia Smirni, and Andreas Stathopoulos. Their exceptional instruction has influenced my study, research, and life in many ways.

I thank our department chair Dr. Robert Michael Lewis, graduate director Dr. Denys Poshyvanyk, and the administration team, Vanessa Godwin, Jacqulyn Johnson, and Dale Hayes for creating an excellent research environment. I thank Dr. Pieter Peers for serving my initial academic advisor.

I thank all members from our research group for providing useful insights: Shuai Hao, Haitao Xu, Yubao Zhang, Dachuan Liu, Daiping Liu, Zhang Xu, Jidong Xiao, and Yue Li.

Finally, I thank my family for their endless support, encouragement, and love.

I would like to dedicate this dissertation to my parents, Xieping Gao and Qin Liu, who have provided endless support and love for me.

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

As cloud computing has become the mainstream of providing IT services, it is essential to achieve high availability and reliability for computing services hosted inside a cloud. Unfortunately, system failures and data center power outages are not rare. One reason is that data centers have to massively expand their scales to meet the ever-increasing service demands. Over-subscription is widely adopted to deploy more servers than the infrastructure can support. Such aggressive policies, though saving costs of upgrading facilities, exhaust the redundancies and tolerances for unexpected faults and abnormal conditions. Even worse, the existence of those issues opens the door for attackers and makes data centers vulnerable to various Denial-of-Service (DoS) attacks. To avoid service interruptions, it is imperative to uncover the potential vulnerabilities and build more robust defense mechanisms in advance.

This dissertation investigates three emerging security threats in public clouds and data centers. First, we unveil a new vulnerability called reduced cooling redundancy that could be exploited by adversaries to severely worsen the thermal conditions in a data center. We systematically study the induced damage and propose a dynamic thermal-aware load balancing to mitigate the threat. Then, we uncover the information leakage problem in multi-tenancy container cloud services. We demonstrate that the consequence of such leakages could lead to privacy breaches or even service outages, and implement a two-stage defense system to secure container clouds. Finally, we expose a new but serious vulnerability that can cause unrecoverable memory inconsistency in existing virtual machine live migration, and we also propose effective defenses.

1.1 Reduced Cooling Redundancy: A New Security Vulnerability in a Hot Data Center

With more powerful servers being deployed at data centers, the amount of heat emitted by those servers is also surging, which requires the cooling system to more efficiently dissipate the increased heat. Otherwise, the overheating would potentially lead to serious hardware failures and even server shutdown for self-protection. Unfortunately, in recent years, the online service interruptions due to cooling failures have not been rare in cloud vendors and enterprises, including Microsoft [15], Rackspace [19], Wikipedia [21], iiNet [12], and University of Pennsylvania [20].

To regulate the temperature in computer rooms, a significant portion of the power consumption of data centers is used for cooling. The cooling cost has reached 24% of a data center's budget [10]. The key factor affecting the cooling cost of CRAC (Computer Room Air Conditioning) systems is the supply air temperature. Data centers have deployed a variety of methods to control the CRAC supply air temperature. For example, some data centers set the supply air temperature automatically based on the workload. More importantly, a recent study shows that increasing the supply air temperature by merely 1°C can save approximately 2-5 percent of the cooling power [38]. Thus, there is a trend in data centers to raise the highest set temperature from 75°F to 85°F or even higher. It is reported that Google has raised the temperature of the cold aisle to 80°F [7]. Those aggressive cooling energy saving policies achieve a low PUE (Power Usage Effectiveness)[1] in a data center. However, the temperature increment also forces the servers running in a hotter environment than before.

Furthermore, advanced techniques like power oversubscripion [42,96] have been widely adopted to accommodate more servers in data centers without upgrading existing power and cooling infrastructures. While the infrastructures were initially well designed with sufficient cooling redundancies, those redundancies have been excessively consumed by power oversubscription. Under the reduced cooling redundancies, an accidental synchronization of running intensive workloads in a set of adjacent servers could result in a local hotspot and even a cascade effect further deteriorating

[1] PUE is the ratio of the total energy consumption of a data center to the energy consumed by computing equipments in the data center.

the thermal conditions.

In this dissertation, we unveil a new vulnerability of existing data centers with aggressive cooling energy saving policies. Such a vulnerability might be exploited to launch thermal attacks that could severely worsen the thermal conditions in a data center. Specifically, we conduct thermal measurements and uncover effective thermal attack vectors at the server, rack, and data center levels. We also present damage assessments of thermal attacks. Our results demonstrate that thermal attacks can (1) largely increase the temperature of victim servers degrading their performance and reliability, (2) negatively impact on thermal conditions of neighboring servers causing local hotspots, (3) raise the cooling cost, and (4) even lead to cooling failures that force some servers to shut down. Finally, we propose effective defenses to prevent thermal attacks from becoming a serious security threat to data centers.

1.2 ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds

Cloud computing has been widely adopted to consolidate computing resources. Multi-tenancy is the enabling feature of cloud computing that allows computation instances from different tenants running on a same physical server. Among different types of cloud services, the multi-tenancy container cloud has recently emerged as a lightweight alternative to conventional virtual machine (VM) based cloud infrastructures. Container is an operating system (OS) level virtualization technology with multiple building blocks in the Linux kernel, including resource isolation/control techniques (e.g., *namespace* and *cgroup*) and security mechanisms (e.g., *Capabilities*, *SELinux*, *AppArmor*, and *seccomp*). By avoiding the overhead of additional abstraction layers, containers are able to achieve near-native performance and outperform VM-based systems in almost all aspects [5, 43, 85]. In addition, the advent of container management and orchestration systems, such as *Docker* and *Kubernetes*, have profoundly changed the ecosystem of building, shipping, and deploying multi-tier distributed applications in the cloud.

Despite the success of container services, there always exist security and privacy concerns for running multiple containers, presumably belonging to different tenants, on the same OS kernel.

To support multi-tenancy on container clouds, we have observed on-going efforts in the Linux kernel to enforce cross-container isolation and de-privilege user-level containers. Existing container-enabling kernel features have greatly shrunk the attack surface exposed to container tenants and could restrain most existing malicious attacks. However, not all subsystems of the Linux kernel can distinguish execution contexts between a container and a host, and thus they might expose system-wide information to containerized applications. Some subsystems are considered to be of low priority for container adaptations. The rest are facing implementation difficulties for transforming their code base, and their maintainers are reluctant to accept drastic changes. In order to close these loopholes, current container runtime software and container cloud providers typically leverage access control policies to mask the user-kernel interfaces of these container-oblivious subsystems. However, such manual and temporary fixes could only cover a small fraction of the exposed attack surfaces.

In this dissertation, we present the information leakage channels we discovered that are accessible within the containers. Such channels expose a spectrum of system-wide host information to the containers without proper resource partitioning. By exploiting such leaked host information, it becomes much easier for malicious adversaries (acting as tenants in the container clouds) to launch advanced attacks that might impact the reliability of cloud services. Additionally, we discuss the root causes of the containers' information leakages and propose a two-stage defense approach. As demonstrated in the evaluation, our solution is effective and incurs trivial performance overhead.

1.3 Sudden Death of Live Migration and Its Security Implications

Virtual machine 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 [28], load balancing [103], fault tolerance [86], energy saving [37, 87], and performance enhancement [27], VM live migration has been widely used in mainstream cloud services [68]. 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 [34], 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 [113] depending on the copying rate and intensity of memory activities. To reduce service downtime, the *post-copy* approach [58] 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 environment. Moreover, the confidentiality of the process can be safeguarded using encrypted live migration [90, 108, 128]. This can further secure the migration process, preventing adversaries in the middle of a network to hijack or falsify memory pages.

In this dissertation, 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. 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.

1.4 Overview

The remainder of this dissertation is structured as follows. Chapter 2 presents our research efforts on launching and defending thermal attacks on data centers. Chapter 3 presents our investigation of the information leakage problem on multi-tenancy container cloud services. Chapter 4 presents our study on attacking and securing the live migration of virtual machines. Finally, Chapter 5 concludes this dissertation.

Chapter 2

Thermal Attack on Data Centers

In this chapter, we systematically investigate the security risk posed by those aggressive policies applied on data centers. We introduce the concept of thermal attack, which can be easily and remotely launched to seriously worsen the thermal conditions at a server level, a rack level, or even a data center level, without requiring any privileges of a hypervisor. Thermal attacks simply run thermal-intensive workloads in victim servers or VMs (Virtual Machines) to rapidly generate a large amount of heat, forcing the victim servers into a high temperature. The overheated servers suffer performance and reliability degradation. Even worse, the accumulated heat can further exacerbate the thermal condition of the peripheral atmosphere, raising the inlet temperature of other servers. The increase of the inlet temperature then increases other servers' outlet temperatures, leading to a vicious cycle. The consequence could be the great increase of hardware failures of many servers in a data center, the significant waste of the cooling costs, and even thermal accidents that force some servers to shut down.

To form the basis for mounting a thermal attack, we measure how thermal-related factors are exhibited in a real server using different HPC (High Performance Computing) benchmarks. We observe that CPU-intensive workloads can generate more heat and cause a higher temperature than other types of workloads, even if the system utilization is at the same level; thus it can be used as thermal-intensive workload to rapidly raise the temperature of a server (i.e., within a few minutes), despite the fact that the temperature increase requires the accumulation of heat. Then, based on our thermal measurements on the real server, we propose to mount thermal attacks in both non-virtualized and virtualized environments, as well as a pulsation thermal attack. As expected, those attacks can largely raise the temperature of the hosting server within a short period time. We further conduct a damage analysis in terms of electromigration, time dependent dielectric breakdown, thermal cycling intrinsic hard failure mechanisms, and disk failure.

To evaluate the impact of thermal attacks on the entire data center, we conduct thermal attacks at the data center level using a computational fluid dynamics based, trace-driven simulation, with a special consideration of the air recirculation condition in the data center. We observe that launching thermal attacks on less than 2 percent of servers in a data center can seriously affect the thermal conditions of the whole data center and raise the cooling costs significantly. Even worse, in some severe attack scenarios, thermal attacks can lead to cooling failures.

Finally, we discuss the attack costs and propose effective defenses against thermal attacks. Although some prior studies have proposed temperature-aware load balancing (e.g., [84]), their approaches are mainly designed based on a *static* profile of the data center thermodynamics, which is profiled using *normal* server workloads. As a result, such solutions cannot effectively handle hotspots that are generated rapidly by malicious workloads at runtime, because the thermal conditions can become significantly different and even completely opposite to the static pre-calculated profile.

The chapter is organized as follows. We provide the background information on system cooling and its inherent vulnerability to a thermal attack in data centers in Section 2.1. We present our thermal measurement study on a real server in Section 2.2. We detail the thermal attacks in non-virtualized/virtualized environments and the damage analysis in Section 2.3. We present the thermal attacks and their impacts at the rack and data center levels in Section 2.4. We describe an effective defense in Section 2.5. We survey the related work in Section 2.6, and finally we conclude the chapter in Section 2.7.

2.1 Background

2.1.1 Server Cooling

Computers have to be operated within a specific range of environmental temperatures. Otherwise, electronic devices may fail to have their normal characteristics and may even malfunction. Most hardware failures cause permanent damage and cannot be recovered. For a server, all its components, like CPU, memory, cache, and bus, produce heat. The mega-scale and complicated design of integrated circuits and chips in modern computer systems further exacerbates the heat generation problem. As a result, cooling techniques or thermal management methods are critical for both hardware and software design. For instance, the chip design must consider heat generation and emission. The positions of different chip components must be carefully chosen to avoid hotspots. A fan is also often used to accelerate the heat emission. In software, various dynamic thermal management mechanisms are proposed: If the temperature violates a carefully selected threshold, the component would have to degrade its performance by reducing the operating frequency, or it may even be forced to shut down for protection from physical damages [91]. Therefore, a redline temperature is often set for a server. If the inlet temperature exceeds the redline temperature, the server would be shut down because the fan itself is not sufficient to cool down the server.

2.1.2 Data Center Cooling

In a data center, tens to hundreds of thousands of high-density (e.g., blade) servers are placed in one closed space, running simultaneously with a high workload utilization. A large amount of heat is generated every second by those servers. Moreover, the fans push the generated heat into the room. Due to possible air recirculation, the hot air may influence the environmental temperature and further impact other servers. Therefore, data center cooling is even more critical due to the high server density.

To cool down servers, existing data centers are equipped with various cooling technologies, including the traditional CRAC air cooling, free air cooling, and liquid cooling. Restrained by geographic limitations and facility costs, CRAC-based air cooling remains the most widely used cooling

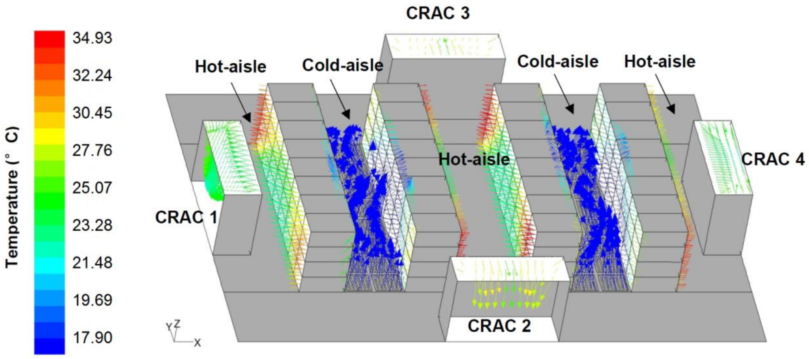


Figure 2.1: Layout of a Typical Data Center.

solution. To enhance cooling efficiency, computer rooms are divided into hot aisles and cold aisles, as shown in Figure 2.1. In a cold aisle, cold air is supplied from the raised floor and flows through the back side of the server racks to absorb the heat generated by the servers. The resulting hot air, with the help of server fans, then enters the hot aisle from the front side of the server racks, and is returned (through the ceiling vents) to the CRAC system and cooled down by the chillers again.

The inlet temperatures of servers must be strictly controlled in a data center in order to avoid overheating, which increases the possibility of causing permanent hardware damage. If the inlet temperature exceeds a threshold, parts of servers or even racks would be forced to shut down. Ideally, the data center should have the cold air and hot air perfectly isolated for high cooling efficiency. However, air recirculation can be common in a data center, in which hot air enters the cold aisles through small gaps between insulation material and the server racks. Thus, cold and hot air can get mixed to some extent as a result. The air density, air flow between servers, supply air temperature of the cooling facilities, power consumption, and outlet temperature of each server can get intertwined in a complicated way to impact the temperature of the whole server room. Therefore, the hot air generated by the servers could potentially cause the inlet temperature to violate the threshold.

To keep the inlet temperature below the safe threshold, the cooling facility's supply air temperature selection is critical. For safety, a low supply air temperature is desired to reduce the possibility of any thermal emergencies. However, a lower supply air temperature can result in a higher cooling cost. As a result, many data centers (e.g., Google) are trying to raise the supply air temperature for a lower cooling cost. Currently, data centers can remain safe even with a higher supply air temperature due to the existence of the cooling redundancies. However, with the deployment of more powerful servers and the current trend of power oversubscription [45, 133], which allows managers to deploy more servers in a room, data centers will soon have almost no redundancies in the near future. As a result, the possibilities of thermal emergencies can significantly increase with a higher supply air temperature, making data centers vulnerable to thermal-related attacks.

Currently in data centers, the common cause of a cooling failure is the breakdown of cooling facilities [12, 19, 21], which significantly reduces cooling capacity and slow down heat dissipation. Conversely, by intentionally running thermal-intensive workloads, the vast heat generation can also exceed the cooling capacity, resulting in a cooling failure in a data center.

2.1.3 Threat of Thermal Attacks

The security threat posed by thermal attacks to a data center is real and difficult to address, mainly due to the following five reasons.

I. The root cause of the threat lies in the wide adoption of aggressive cooling and power management policies, such as raising the supply air temperature and power oversubscription, which allow more servers being deployed in a data center with less cooling cost. Although data centers were initially designed with sufficient cooling redundancies, those aggressive policies significantly reduce the cooling redundancies, making data centers themselves vulnerable to abnormal thermal conditions.

II. Although modern servers are equipped with various chip-level sensors, such as temperature sensor for each core, those chip-level sensors cannot provide server-level information (e.g., server inlet and outlet temperature). Core temperature does not equal to inlet temperature and outlet temperature. A core's temperature varies much faster. An effective thermal attack does not need to stress the CPU all the times but just keep the outlet temperature at a high level. As a result, Core sensors cannot provide server-level or DC-level thermal monitoring.

III. While thermal sensors definitely help to monitor the thermal conditions in a computer room, most of today's production data centers have just a few thermal sensors or probes for the entire data center. Deploying sensors on the server- or rack-level would be very costly. Even with rack-level sensors (only 2-4 sensors per rack), it is impossible for them to cover hundreds of servers in a rack. Thus, the occurrence of a local hotspot is inevitable. Without the global knowledge of overall thermal conditions in a computer room, chip-level protections, like the Intel RAPL (Running Average Power Limit) providing and limiting the power consumption for each CPU package, cannot prevent the occurrence of local hotspots.

IV. To defend against thermal attacks on a data center, the thermodynamics of a data center is important to consider. The temperature-based feedback control mechanism would help to limit the temperature for local hotspots. However, such a feedback control mechanism does not exist in the current data centers. At the chip-level, there are some feedback control mechanisms used for overheating protection; however, at the data-center-level, without the global knowledge on thermal conditions, it is very challenging to deploy feedback control mechanisms for temperature management of the whole data center.

V. Since thermal-intensive workloads themselves are benign and do not exploit any system vulnerabilities, it is difficult to distinguish attackers from normal users. Moreover, process-level power/thermal profiling also cannot defend against data center level thermal attacks. Attackers are not limited to use just one process to generate much more heat. They can use many accounts to run different workloads simultaneously to generate a significant amount of heats.

2.2 Real Server Measurement

The thermal conditions of a server are affected by various factors. Unlike power that can be generated and terminated instantaneously, the change of temperature is a process of heat accumulation and dissipation. Different components of a server run simultaneously and generate heat. The accumulated heat then causes the temperature to raise. To understand the thermal characteristics of a physical server, we first perform a measurement study running in our small testbed. We carefully design a set of experiments to explore the thermal characteristics of a server in the following four

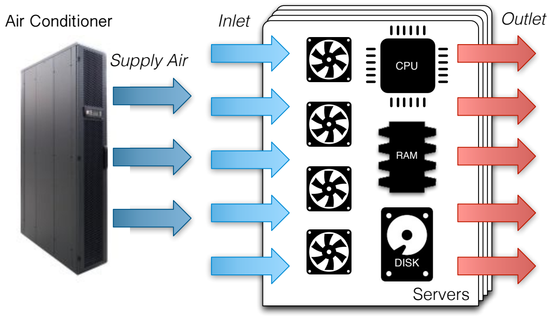


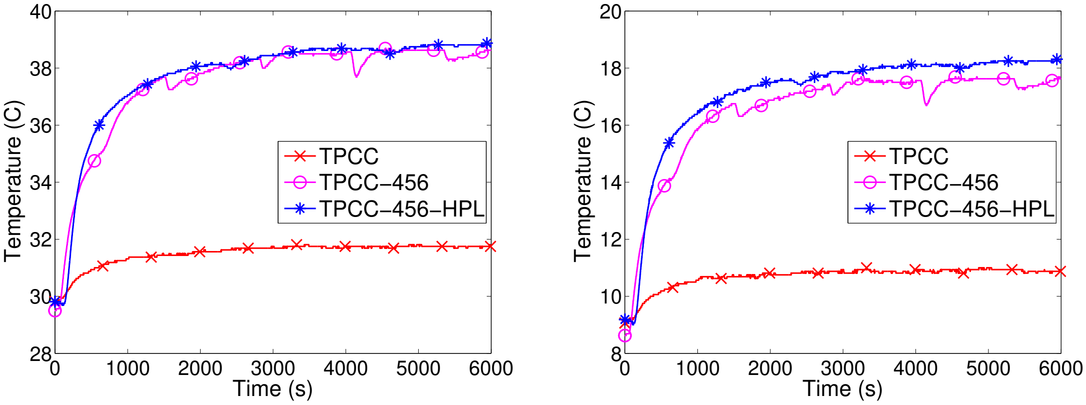
Figure 2.2: Inlet and Outlet Temperature of Servers.

aspects: (1) the impact of different workloads, (2) the thermal condition variations under the same system utilizations, (3) the relationship between the thermal condition and power consumption, and (4) the speed of heat accumulation and dissipation.

Our testbed is a mainstream Supermicro server, equipped with Intel Xeon 2.27GHz CPU with 16 cores, 32 GB of RAM and running Ubuntu Linux 12.04 with kernel version 3.13.0. The testbed is placed in a sealed environment without any window to the outside world. The ambient temperature is about 21°C cooled by the central air conditioner in our building. We measure the inlet and outlet temperature of the server using "Go!Temp" temperature probe, which has a resolution of 0.07°C [11], and monitor the power consumption using "Watts up? .Net" digital power meter. The inlet and outlet temperature of our server is illustrated in Figure 2.2.

2.2.1 Workload Impacts

We first explore how different workloads affect the thermal conditions of a server. The attack workload should be designed to fully utilize all components, generate a large amount of heat and raise the temperature in a fast and significant manner. The "man-made" hot status would reduce the reliability of a server. To evaluate the impact, we pay special attention to the outlet temperature of the server for the following two reasons. First, the output air is a direct sign of the temperature of the server. The inner temperature can only be hotter than the outlet temperature. Second, the output air is exported to the hot aisle of the computer room and further impacts the whole atmosphere due to air recirculation. Thus, a high outlet temperature can negatively affect adjacent servers in a computer room.



(a) Outlet Temperature.

(b) Margin Temperature.

Figure 2.3: Temperature with HPL, SPECCPU and TPCC-UVa.

We measure the outlet temperature of our server under three different scenarios. First, we use TPCC-UVa database benchmark [78] as the workload to generate a large amount of I/O operations. TPCC-UVa benchmark is an open source TPC-C benchmark, which is an online transaction processing benchmark, written in C language and using the PostgreSQL database engine. We set the warehouse number 50 to ensure a sufficiently large workload, but keep the CPU utilization less than 10% to limit the intensity of CPU activities. In the second scenario, we simultaneously run the TPCC-UVa and the 456.hmmer benchmark from SPECCPU2006, which is a widely used HPC benchmark. We run the 456.hmmer with 16 copies to fully utilize the CPU resources in our server. In the third scenario, we add another HPC benchmark, the High Performance Linpack (HPL). The Linpack benchmarks measure the capability of solving random matrix productions. There are multiple configurable parameters that could affect the workload of the benchmark. We set the number of processors to 16 since our server is a 16-core machine. We set the problem size N, which is the size of the input matrix, to 40,000. For the block size, we choose 100 in this experiment. This configuration promises a heavy computing workload on our server. Since temperature increase is a relatively long process of heat generation, we run the experiment for 100 minutes.

The experimental results are illustrated in Figure 2.3. Figure 2.3(a) shows the outlet temperature under the three different scenarios. Although the air conditioning is set with a fixed supply air temperature, the inlet temperature still varies in a range of 1°C due to various surrounding

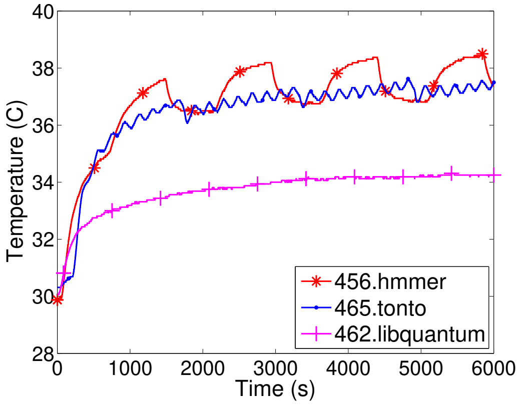


Figure 2.4: Outlet Temperature with Different SPECCPU Benchmarks.

factors. We define the margin temperature as the difference between the outlet and inlet temperature, which is shown in Figure 2.3(b). The margin temperature clearly indicates the temperature increase because of running specific workloads on our server.

While the outlet temperature of the server at idle is about 30°C, after heating by intensive workloads on CPU, memory, and disk, the outlet temperature can reach up to 39°C, as shown in Figure 2.3(a). Specifically, I/O intensive workloads create heat and raise the outlet temperature to some extent; more obviously, after running with CPU-intensive workloads like 456.hmmer, the outlet temperature increases significantly. Then, additional workloads like HPL can further generate more heat on this basis. Given that the inlet temperature is about 21°C set by the cold air, the thermal-intensive workloads achieve a more than 18-degree temperature difference. Also, a nearly 40°C outlet temperature indicates an even hotter temperature inside the server. Thus, our experimental results demonstrate the feasibility of mounting a thermal attack against a server using thermal-intensive workloads, especially CPU-intensive workloads.

2.2.2 System Utilization

To explore the outlet temperature variation under the same system utilization, we use a set of SPECCPU 2006 benchmarks to represent different types of workloads running in the server. We carefully choose the set of benchmarks in which the system resources are consumed at the same level. To ensure exactly the same CPU utilization, we repeatedly and simultaneously run each benchmark with 16 copies to fully utilize all cores. Also, proven by [123], the memory consumption of those benchmarks are quite similar. For benchmark 456.hmmer and 462.libquantum, the average memory utilization is about 24% and 25%, respectively. Moreover, as SPECCPU involves limited I/O activities, the system resources consumed by the chosen benchmarks are very close to one another. The experimental results are illustrated in Figure 2.4.

We observe that under the same system utilization, different types of workloads could lead to different thermal conditions. An almost 6°C temperature difference is generated by different workloads with the same CPU and memory utilizations. According to [123], 462.libquantum consumes a relatively high memory consumption but produces the minimum outlet temperature increment. By contrast, 456.hmmer can cause a much higher temperature increment than 462.libquantum, and 465.tonto raises the outlet temperature more than 7°C while consuming the least amount of memory. The main reason could be that the types and ratios of instructions composing the benchmarks are different. Although the system utilization is the same, the underlying pipeline flows are actually very different. The ratio of different types of instructions, the probability of branch prediction, and the data dependence could be very different. Those differences further cause CPU halt and leave functional units idle, resulting in generating different amounts of heat. We also observe the zigzag shape of the temperature dynamics of 456.hmmer. The reason is that multiple copies repeatedly run together to keep generating heat; however, they do not finish at the same time. In the gap between the first end of one copy and the start of next round, the system utilization is reduced, resulting in less heat generation.

2.2.3 Power Consumption

Heat is generated when currents flow through resistors, obeying the Joule-Lenz law. Consequently, power consumption, which represents the rate of energy transformation, is closely related to heat generation. The heat in joules can be given by:

$$H = I^2Rt = Pt, \tag{2.1}$$

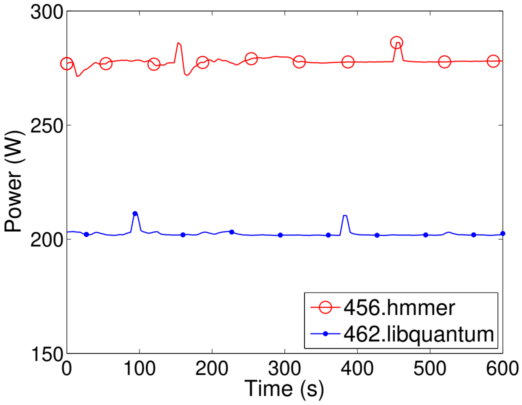


Figure 2.5: Power Consumption of Two Benchmarks.

where I is the current, R is the resistance, and t is the time. From the equation, we can see that the heat generation is linearly increased to the power consumption. Using the results obtained from the same set of SPECCPU 2006 experiments conducted above, we present the power consumptions of 456.hmmer and 462.libquantum in Figure 2.5. As the benchmark that raises to almost 7°C higher than 462.libquantum, on average 456.hmmer also consumes almost 70W more power than 462.libquantum. Both theory analysis and experimental results indicate that running a power-intensive workload ensures more heat, which could be exploited for mounting a thermal attack. This result (i.e., 70W difference in power consumption) also forms the basis for the configuration of one parameter in our large-scale experimental evaluation.

2.2.4 Heating Speed

We further measure the speed to heat a server. Unlike a power stimulus that surges instantaneously, the temperature dynamics is a process of heat accumulation. We choose HPL with a block size of 50 as the thermal-intensive workload. We run the workload for 30 minutes and then stop it. In the first 10 minutes, the temperature starts to increase quickly. Then, although the temperature is still raising, the speed drops. The dynamics of the server's outlet temperature is illustrated in Figure 2.6. The temperature starts to increase quickly in the first 10 minutes. The temperature can increase about 6.8°C higher than that of the idle state. Then, although the temperature is still raising, the speed drops. After we stop the workload, the cooling system can quickly

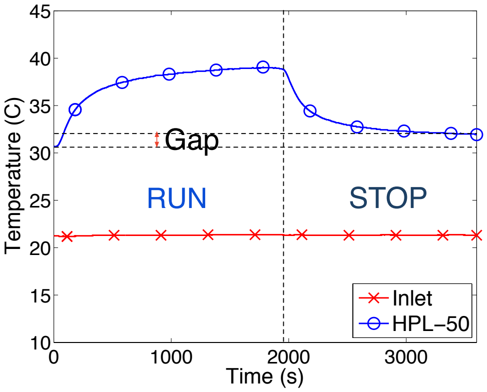


Figure 2.6: Temperatures with HPL Running and Stopping.

decrease the temperature in the first several minutes. The temperature drops almost 4.5°C within the first five minutes and 6°C within the first 10 minutes. However, after 10 minutes, the speed to dissipate the heat slows down considerably. Even after half an hour, the outlet temperature still cannot return to the original value at the idle state, which implies that the full dissipation of heat requires quite a long time. Overall, we have two major observations: (1) the temperature of a server varies quickly when a thermal-intensive workload starts/stops; and (2) the dynamics of the server temperature is non-linear.

2.3 Thermal Attack

In this section, we first describe our threat model. We then mount thermal attacks on both virtualized and non-virtualized environments, as well as a pulsation attack. Based on the attack results, we further conduct damage assessment.

2.3.1 Threat Model

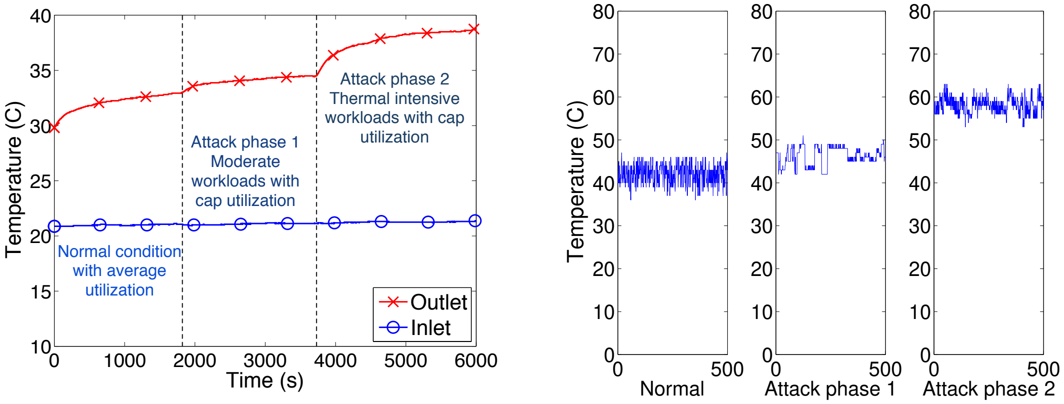
A thermal attack can be launched at the server level, rack level, or data center level. We assume that the target data center has the following features. (1) It is cooled by traditional CRAC cooling systems with optimal cooling policies deployed to maximize cooling efficiency, i.e., the supply air temperature is set as high as possible to save energy while keeping the inlet temperature below a redline threshold. (2) It provides utility-based computing services that are accessible over the

Internet. (3) Thermal sensors are equipped in the data center, and temperature monitoring is conducted at the rack level. Note that most current data centers have just a few thermal probes for the entire data center, and only some experimental data centers (like HP Labs) have about 2-4 thermal sensors per rack. (4) Like most current data centers, the target data center also deploys power oversubscription.

The attacker could be an individual, a competitor of a cloud service provider, or a cybercrime organization. The attacker does not require more privileges than a regular user to access the target cloud service, and no compromised hypervisor is required. This is mainly due to the fact that in a cloud environment, especially IaaS (Infrastructure as a Service) and PaaS (Platform as a Service), a tenant has the privilege to run any workloads/applications at the guest level, including the benchmarks used in our measurement study. While those workloads are developed to assess the performance of specific scenarios, different combinations and configurations of them compose thermal-intensive workloads.

However, the attacker might utilize the publicly available information or network probing to figure out the network topology inside a cloud [98], and exploit more advanced probing techniques to achieve tenant co-residence in the same physical server or rack [122]. Moreover, the attacker could run advanced thermal-intensive programs (e.g., power virus [46, 47]), instead of regular thermal-intensive workloads, to further exaggerate the heat generation.

Note that a more tangible goal of a thermal attack is not to shut down an entire data center, but to cause a cooling failure in a data center, in which some victim servers are forced to shut down. Under a thermal attack, much more heat will be generated than normal. Once the heat released into the hot aisle surpasses the recyclability of a cooling machine, the inlet temperature increases. The raising of the inlet temperature will further increase the outlet temperature and generate more heat. Such a vicious cycle will finally lead to a cooling failure. Moreover, the overheat caused by thermal attacks will reduce the performance and reliability of victim servers, increase the possibility of hardware failures, and force the data center to reset its cooling configuration, resulting in a much higher cooling cost.



(a) Outlet Temperature

(b) Temperature of the 1st Core

Figure 2.7: Thermal Attack on a Non-Virtualized Environment.

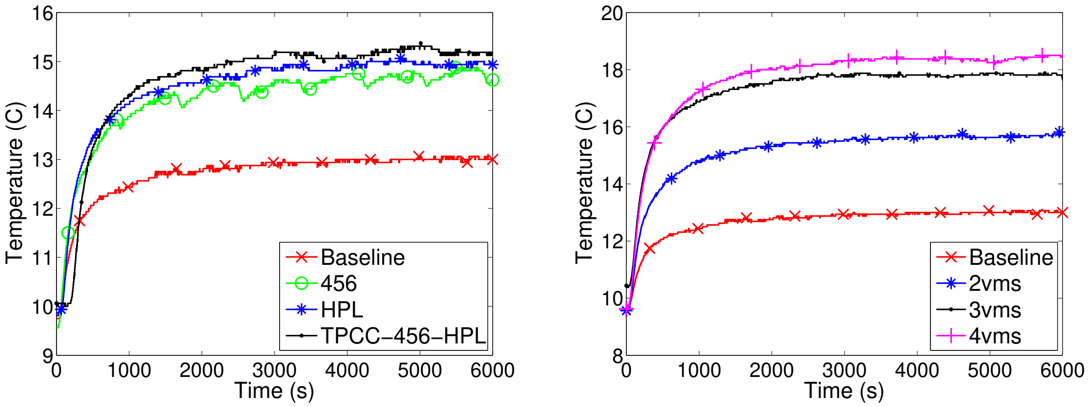
2.3.2 Non-virtualized Environments

In this scenario, we assume that an attacker owns a dedicated host (e.g., a dedicated instance in Amazon EC2) in a data center. As the attacker can choose any workload to run at will, the attack vector is straightforward, running thermal-intensive workloads for a relatively long time.

As reported in [25], the average system utilization of most servers in a data center is between 20 - 30%. We also assume that the victim server is running moderate workloads with 25% system utilization. We choose benchmark 462.libquantum to represent a moderate workload. Although 462.libquantum belongs to the CPU-intensive benchmark suite, it is highly vectorizable. The high-dimensional matrix computation requires more I/O operations and makes 462.libquantum consume less power consumption than other SPECCPU benchmarks like 456.hmmer, as shown in Figure 2.5.

The thermal attack starts after the victim server running the moderate workload for 30 minutes. The attacker first pushes the system utilization to 100%. The first attack phase lasts 30 minutes. After the utilization reaches its cap, the attacker replaces the moderate workload with the thermal-intensive workload to further heat the server. In this attack, we use a combination of SPECCPU 456.hmmer and HPL with a block size set of 50 to represent the thermal-intensive workload. Again, the attack lasts about 30 minutes.

The dynamics of the server's outlet temperature is shown in Figure 2.7(a). At the beginning,



(a) One VM.

(b) Multiple VMs.

Figure 2.8: Margin Temperature of Thermal Attacks on a Virtualized Environment.

the moderate workload runs at an average of 25% utilization in the server. The outlet temperature of this phase is 32°C. On the next phase, the moderate workload pushes the server into the cap utilization. With fully utilized system sources, the outlet temperature reaches 34°C. Since the difference is less than 2°C, the cooling control system can handle it without any trouble. In the final phase, under the thermal-intensive workloads, the outlet temperature is rapidly raised to more than 38°C, which is 6°C higher than the temperature under the moderate workload. Even under the same utilization, it is evident that malicious thermal-intensive workloads can generate a significant temperature rise and emit a large amount of heat to the computer room. We also show the temperature of the first core in our testbed under the thermal attack on a non-virtualized environment in Figure 2.7(b), which clearly demonstrates the significant temperature increase of the core under the thermal attack.

2.3.3 Virtualized Environments

Most computing services provided by clouds run in virtualized environments, except for dedicated hosting services. A simple approach to mounting a thermal attack is to rent a cloud instance and run thermal-intensive workloads in a VM, regardless of cloud service models (e.g., SaaS (Software as a Service), PaaS, and IaaS). However, only in IaaS can attackers achieve tenant co-residence and fully control the running workloads. Thus, the effective attack vector in a virtualized environment is to subscribe one or multiple VM(s) in one physical host and then run thermal-intensive workloads at the same time.

We emulate the IaaS services by running four VMs on our testbed, each VM allocated with 4GB memory and 4 vCPUs, under Xen hypervisor, which is the same hypervisor running in Amazon EC2. We run all VMs with 25% utilization as our baseline for representing a normal case. Then we select one or more VM(s) with designed workloads running to exhaust the host resources. In our controlled VM(s), we run the following thermal-intensive workloads: (1) SPECCPU 456.hmmer, (2) HPL with the block size set to 200 and N set to 10,000, and (3) a combination of 456.hmmer, HPL and TPCC-UVa. Figure 2.8(a) shows the attack results.

While the outlet temperature is about 33°C under the baseline condition, some attacks can raise the temperature to almost 37°C. The attacks running HPL, 456.hmmer and TPCC-UVa, which ensure both CPU and I/O intensive workloads, can lead to about a 16°C margin temperature. With just one malicious VM controlled, an attacker can raise the temperature by almost 4°C higher than that of the baseline within 10 minutes.

Recent research shows that it is still relatively easy to achieve tenant co-residence in public clouds. When an attacker can run multiple VMs on the same physical machine, it can mount a more powerful thermal attack. We measure the temperature of the host when different numbers of VMs run thermal-intensive workloads. The results in Figure 2.8(b) clearly demonstrate that with more VMs controlled, a thermal attack can heat the server to a higher temperature. Also, a thermal attack on three VMs can lead to about a 5°C higher temperature than the baseline case. However, once all CPU resources are already fully utilized in a physical machine (under four VMs in our experiment), running even more VMs in the same machine cannot achieve a significant difference in heating generation.

In our experiment, we assume that VM live migration is not adopted in the target data center, and the attackers can run thermal-intensive workloads on the target server/VMs to reach full system utilization. This assumption is valid because VM live migration has not yet been widely adopted in real clouds, and live migration suffers non-negligible downtime on the migrated VMs running with intensive workloads. Note that if VM live migration is enabled, the process of live migration can

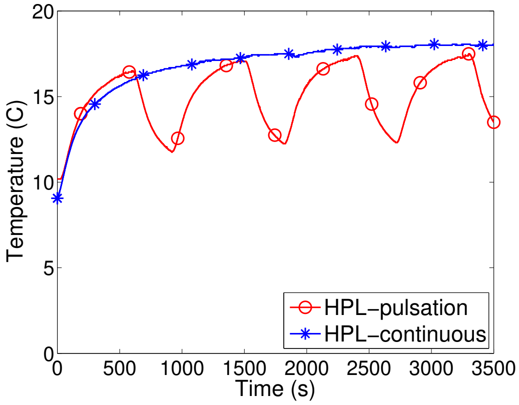


Figure 2.9: Margin Temperature of Pulsation and Continuous Attacks.

cause a sharp rise of power consumption on both source and destination servers [123]. Such a power spike may trip a circuit breaker and result in a power outage.

2.3.4 Pulsation Thermal Attack

Based on our measurement results, the heat cannot be emitted immediately, and the temperature dynamics is non-linear. Thus, attackers can heat the targeted server within a certain amount of time and then pause for a while, and repeat this on/off heat pattern for many times. We call such an on/off heat strategy a pulsation thermal attack. Using the same HPL benchmark with the block size of 50 and the problem size of 40,000, we mount both a pulsation thermal attack and a continuous thermal attack. The results are shown in Figure 2.9.

In comparison with the continuous attack, on one hand, the pulsation attack can reach almost the same maximum temperature, with about 0.56°C lost; but over the entire attack duration, the difference of average temperature between these two attacks is about 1.8°C, which means that less heat is generated by the pulsation attack.

On the other hand, in the pulsation attack, the temperature variation can be as large as 6°C in 15 minutes, which is much higher than that of the continuous attack. A recent work [38] reports that a high variability in temperature has a stronger effect on hardware reliability. Based on their observations, the probability of hardware failure is almost double when the CoV (coefficient of variation) in temperature is bigger than 0.0074. We list the CoV in temperature of these two attacks

Table 2.1: CoV.

Attack	CoV
Continuous	0.0071
Pulsation	0.0440

in Table 2.1. It clearly shows that the CoV of the pulsation attack is much larger than both 0.0074 and that of the continuous attack. Thus, a pulsation attack will have a less chance to cause local hotspots and cooling failures but degrade its end-host's hardware reliability more seriously than its continuous counterpart.

Moreover, the attack cost of a pulsation attack is lower than that of a continuous attack, especially for those attacks on SaaS platforms such as web servers. Besides the periodic style, a pulsation attack can also be launched with random intervals to avoid specific traffic patterns, making the pulsation attack harder to detect.

While our current attack vector is to simply run thermal-intensive workloads on IaaS platforms, there are many other potential attack vectors on different platforms (e.g, PaaS and SaaS). For instance, as shown in the previous work [117], random webpage requests can cause significant cache misses and then consume much more energy. Such a feature could also be exploited in thermal attacks. We will explore more effective attack vectors in our future work.

2.3.5 Damage Analysis

We use several most common temperature induced intrinsic hard failure mechanisms, including electromigration, time dependent dielectric breakdown, thermal cycling and disk failure [36, 101, 107], to analyze the reliability degradation caused by a thermal attack. We use λ to represent the failure rate of each failure mechanism.

Electromigration (EM). EM is induced by the gradual movement of the ions in a conductor as a result of the momentum transfer between electrons and the diffusing metal atoms [9]. Hardware failures including the opening of metal lines/contacts, shorts between adjacent metal lines, or metal levels or junctions could be caused by EM. Equation 2.2 gives the EM failure rate (λ_{EM}), which is commonly based on Black's model.

$$\lambda_{EM} = A_0(J - J_{crit})^{-n}e^{(-E_a/kT)} \tag{2.2}$$

where A_0 is a constant that is empirically determined, J represents the current density, J_{crit} is the threshold current density, E_a is a material dependent constant representing the activation energy, and k is Boltzmann's constant.

Time dependent dielectric breakdown (TDDB). TDDB occurs when the gate oxide breaks down due to low electric fields. It causes the formation of conductive paths through dielectrics. The model is defined in Equation 2.3. The parameters are similar to those of EM.

$$\lambda_{TDDB} = A_0e^{\gamma E_{ox}}e^{(-E_a/kT)} \tag{2.3}$$

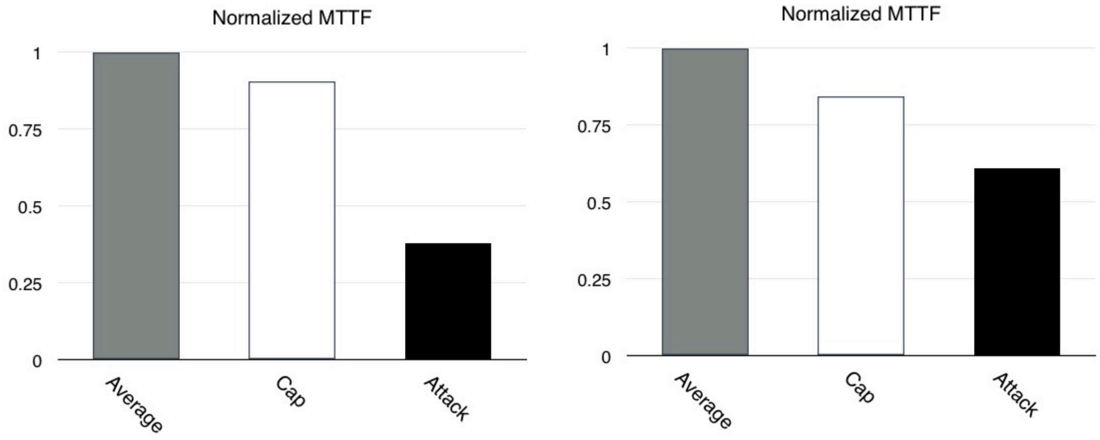
Thermal cycling (TC). The large difference in thermal expansion coefficients of silicon substrate and other materials, which causes thermal cycling (TC), eventually leads to permanent failures, such as the creation of cracks, fractures, short circuits, die interface, and more. The effects of low frequency cycling can be modeled via the Coffin-Mason equation [62].

$$\lambda_{TC} = C_0(\Delta T - \Delta T_o)^{-q}, \tag{2.4}$$

where ΔT is the temperature difference, q is a material dependent Coffin Manson exponent with a common value ranging from 1 to 9. ΔT_o is the portion of the temperature cycle range in the elastic region and could be dropped since it is usually much less than the temperature cycling range.

Disk failure. For damage analysis on a disk due to elevated temperatures, we use the model build in [101], which is derived from the Arrhenius equation. Since our testbed does not contain an embedded sensor for monitoring disk temperature, we analyze the thermal impact on the lifetime of a disk based on the outlet temperature, which should be lower than the actual disk temperature.

$$\lambda_{disk} = Ae^{(-E_a/kT)} \tag{2.5}$$



(a) CPU Damage.

(b) Disk Damage.

Figure 2.10: Attack on a Non-Virtualized Environment.

We use the reliability model in RAMP (Reliability-Aware MicroProcessors) [107], which assumes that all individual failures are independent. We add up all the individual failure rates and compute the MTTF (Mean-Time-To-Failure) as $1/\lambda$.

Our model is calibrated so that the default MTTF is set to 10 years under normal conditions as previous hardware reliability research such as [64]. While different hardware with different technology could possess different lifetimes, the trend remains similar. Therefore, we normalize all results to minimize the calibration error.

We use the temperature of all CPU cores to estimate the CPU reliability. The normalized results of impacts upon reliability are illustrated in Figure 2.10, where Figure 2.10(a) indicates the core damages while Figure 2.10(b) shows the disk failure. The average bar represents the first phase where the testbed is running moderate workloads with average utilization. The second phase is represented as “cap”, which means the utilization has reached the cap utilization. The “attack” bar represents the last phase where thermal-intensive workloads are running. We can see that when the system utilization reaches a certain cap, the increased temperature affects the hardware reliabilities. However, under thermal attacks, the hardware reliabilities are seriously degraded.

Figure 2.11 shows the impact of thermal attacks on a virtualized environment. Even with just one VM controlled, the reliability of the host suffers much degradation, with almost a half reduction of the baseline case. Figure 2.12 shows the dynamics of MTTF under thermal attacks with different

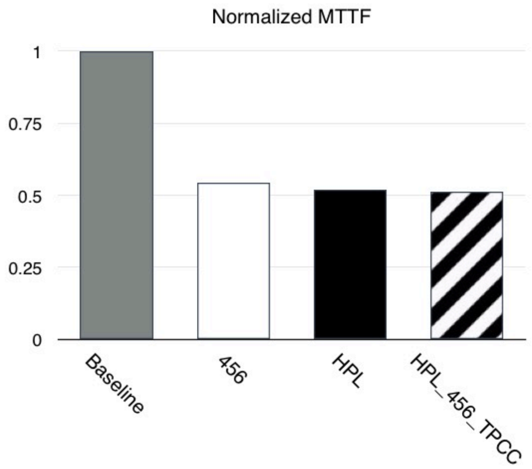


Figure 2.11: Attack on a Virtualized Environment.

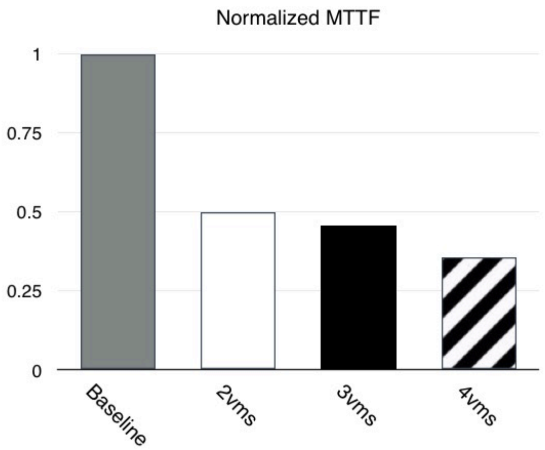


Figure 2.12: Attack with Various Numbers of VMs.

numbers of VMs, indicating that the reliability would suffer more degradation with more VMs under malicious control.

Soft errors. High temperature could also negatively affect the soft errors, which will happen if the collected charge at a junction is equal to the critical charge. The critical charge represents the minimum charge to flip the bit in the cell. It was observed that the increase of temperature decreases the value of the critical charge [32, 66], making the occurrence of soft errors more probable.

2.4 Attacks on Data Centers

The damage of thermal attacks is not limited to victim servers. Thermal attacks also change the surrounding environmental temperature of these victim servers, and thus impacting the thermal conditions of the entire computer room, as well as the thermal performance of other servers and the cooling costs of the data center. To evaluate the impact of thermal attack at the data center level, we conduct a trace-driven simulation based on computational fluid dynamics (CFD), a powerful mechanical fluid dynamic analysis approach that can be used to simulate the air recirculation conditions in the data center. We assume that the simulated data center is equipped with air-cooling CRACs. Also, the data center has adopted a smart workload scheduling policy (e.g., [75]), so that the supply air temperature set point can be set higher for minimizing cooling costs, as suggested in [38].

For the data center layout, we use a standard layout with alternating hot and cold aisles, as illustrated in Figure 2.1. The cold air goes under the raised floor and enters the cold aisle through perforated tiles to cool down the servers. Note that the standard data center layout has been designed to minimize the air recirculation effect. Other data center structures could result in worse damages under a thermal attack. We assume that the targeted data center has one computer room, which contains four rows of servers, with eight racks in each row. Each rack contains 40 servers, totaling 1,280 servers. The volumetric flow rate of the intake air is set to 0.0068m^3/s for each server. The rate for a CRAC unit to push chilled air into a raised floor plenum is 9,000ft^3/min. We use a widely adopted CFD package, Fluent [4], to simulate the thermal environment under different workload distributions, and then get the percentage of heat flow recirculated among the servers.

Based on the CFD analysis [110], the air recirculation effect among different servers can be modeled as follows:

$$K_i T_{out}^i = \sum_{j=1}^{N} h_{ji} K_j T_{out}^j + (K_i - \sum_{j=1}^{N} h_{ji} K_j) T_{sup} + P_i, \tag{2.6}$$

$$T_{in}^i = \sum_{j=1}^{N} h_{ji} * (T_{out}^j - T_{sup}) + T_{sup}, \tag{2.7}$$

where T_{out}^i represents the outlet temperature of server i. K_i is a multiplicative factor representing the air density and the air flow rate. The air recirculation is described with h. T_{in}^i and T_{sup} represent the inlet temperature of server i and the supply air temperature of CRAC cooling units, respectively. P_i is the power consumption of server i. The outlet temperature of a server is impacted by the air recirculation from server j to server i, the cooling effect of the supplied cooling air, and the power consumption of server i. Equation 2.7 indicates that the supply air temperature and the recirculation heat, together, determine the inlet temperature. The power consumption also contributes to the thermal condition in the data center.

We use a server trace file from one of the largest cloud service vendors in our simulation, which contains the average CPU utilization of 5,415 servers in every 15 minutes for one week. We

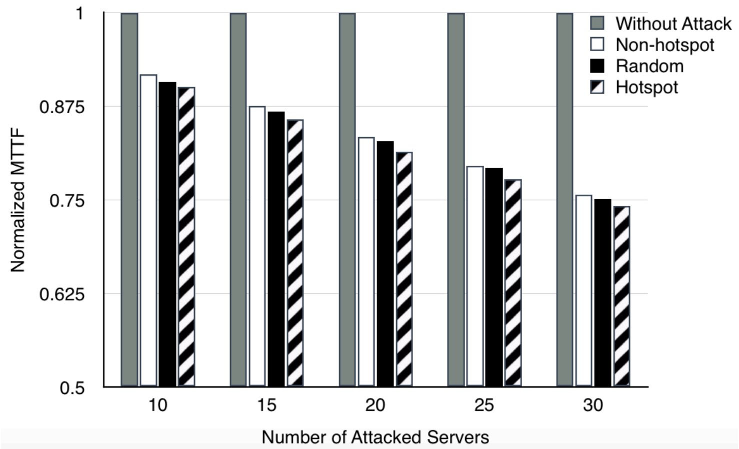


Figure 2.13: Thermal Attacks on a Rack Significantly Reduces the Server MTTF. A Hotspot Attack is Shown to be More Effective Because of Its Consideration of Air Recirculation.

conduct the thermal attack based on the results from our server-level experiments. We assume the server consumes 100W of idle power and 300W when fully utilized with moderate workloads. As illustrated in Figure 2.5, by running some thermal-extensive workloads, the power consumption can become higher than that of running moderate workloads, even when the utilization is the same. The difference could be as large as 70W. In our attacks, we conservatively assume that thermal-intensive workload consumes 60W more than the normal cases.

2.4.1 Rack-level Attack

We have shown that a thermal attack can significantly raise the temperature of the targeted server. Here we first evaluate the impact of a thermal attack on a server rack. In the rack environment, air recirculation causes servers to affect the temperatures of one another. As a result of heat flows, different rack locations lead to different temperature impacts. For example, servers at the bottom of the rack might have smaller impacts because cold air is supplied from the raised floor and flows upward to the ceiling vents. As result, the bottom servers normally have lower temperatures. To consider the thermal effects of air recirculation, we compare three attack strategies for a rack: (1) hotspot attack; (2) non-hotspot attack; and (3) random attack. In a hotspot attack, we mainly attack servers that have larger impacts on the rack (e.g., at the top of the rack) and so it is easier to create hotspots. In a non-hotspot attack, we focus on servers located in the position with small thermal effects (e.g., at the rack bottom). In a random attack, we randomly choose servers to attack. We attack different numbers of servers in one rack and leave other servers in the idle state. As introduced in Section 2.3.5, the temperature rise affects the hardware failure rate. We thus conduct a hardware failure analysis using EM, TDDB, and disk failure to assess the thermal attack on a rack. We use the same parameters and reliability model as Section 2.3.5.

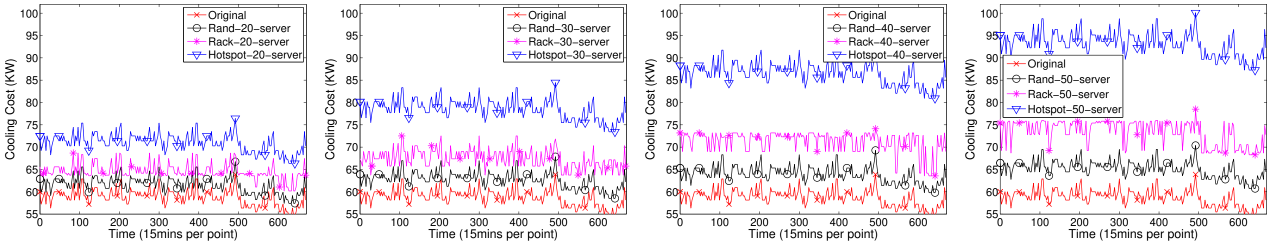
Figure 2.13 illustrates the results (normalized to those of conducting no attacks). We can see that the average server MTTF is reduced when more servers are under attack. For example, with about 10 servers under attack, thermal attacks can enlarge the failure rate by $\sim$10% for all servers in a rack, resulting in a normalized MTTF of $\sim$90%. When 30 servers are attacked, the normalized MTTF drops to 75%. Among the three types of attack, the hotspot attack causes the most damage as those positions can maximize the thermal effects. It indicates that air recirculation indeed affects the server thermal conditions even in a single rack.

2.4.2 Datacenter-level Attack

2.4.2.1 Attack scenarios

Since the knowledge of air recirculation can make a difference, we now consider three types of attackers. In the first case, the attacker does not have any knowledge about the target data center, such as the position of the servers or the layout of the data center. Such attackers just randomly choose a certain number of servers to launch thermal attacks. To evaluate this kind of attacks, we randomly attack different numbers of victim servers in the target data center, and call them *random* thermal attacks.

Unlike the first type, the second type of attacker owns some advanced networking intrusion knowledge, but still does not know the data center layout. Those advanced attackers could launch more powerful thermal attacks by exploiting side-channel techniques. For example, by repeatedly creating instances and checking IP addresses or using networking probe techniques, an advanced attacker can achieve co-residence on one rack. We assume that the selection of the racks for mounting a rack-level thermal attack is random. To evaluate those advanced attack scenarios, we conduct rack-level thermal attacks on one or a few randomly selected racks with massive thermal-



(a) 20 Servers under Attack. (b) 30 Servers under Attack. (c) 40 Servers under Attack. (d) 50 Servers under Attack.

Figure 2.14: Cooling Costs in the Data Center under Different Types of Thermal Attacks.

intensive workloads. We call such attacks *rack-level* thermal attacks.

In the third case, attackers may gain much knowledge of the target data center via various approaches. For example, they could know the overall physical layout of the target data center through either public documents or network probing. Also, via an elaborately long-term observation of servers in the target data center, the attackers may roughly infer where the hotspots could be and whether a server is located in a hotspot, as well as the load balancing policy possibly adopted by the data center. Though somewhat costly to acquire, such knowledge can largely expand the damage of a sophisticated thermal attack and be devastating to the data center. We evaluate these well-planned attack scenarios to understand the consequences. In such attacks, we assume that an attacker can roughly pinpoint the positions of the target servers, and call them *hotspot* thermal attacks.

2.4.2.2 Impacts on cooling costs

Thermal attacks not only damage the hardware, but can also generate local hotspots in the computer room. Those servers located in a hotspot suffer higher environmental temperatures than others. A local hotspot can further affect the entire data center. To eliminate any local hotspots, the data center has to decrease the supply air temperature set point. Otherwise, the hotspot might cause cooling system failures. We first investigate the impacts of a thermal attack on the cooling costs. As mentioned above, existing data centers normally deploy aggressive cooling energy saving strategies to reduce cooling costs. The temperature raised by a thermal attack can force the targeted data center to change the cooling conditions, thus increasing the cooling electricity bills.

We assume that the targeted data center is equipped with intelligent cooling adjustment mechanisms, where the supply air temperature set point is automatically adjusted as high as possible, in order to keep the inlet temperatures of all servers below the redline threshold. Note that such an intelligent cooling system is not yet adopted in most of today's production data centers due to their incapability of conducting thermal monitoring for individual servers. We assume such a system to investigate the impacts of thermal attack on even a data center with advanced cooling systems.

We estimate the cooling cost based on the power consumption of traditional CRACs, which relies on the cooling coefficient of performance (COP) and the power consumption of all servers. It can be calculated as follows,

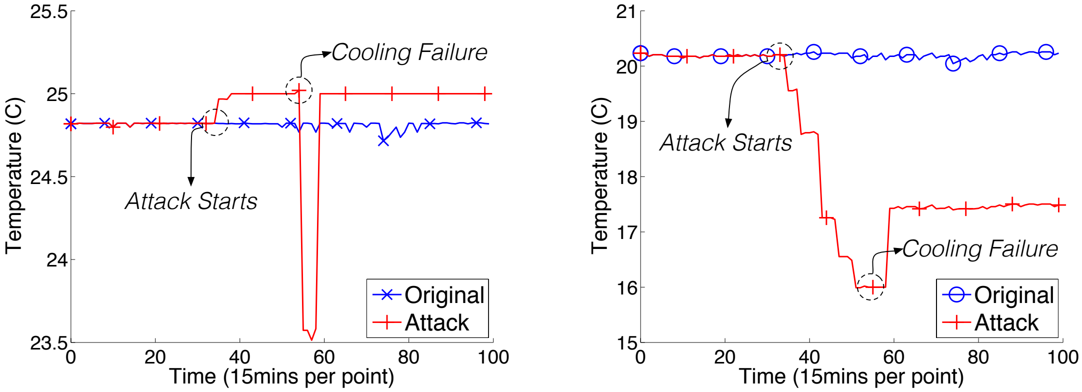
$$P_{CRAC} = \frac{P_{server}}{COP}, \tag{2.8}$$

where P_{server} is the power consumption of all the servers. COP characterizes the cooling efficiency of a CRAC system. According to [23], the COP could be further acquired based on the supply air temperature, T_{sup},

$$COP{=}0.0068 \times T_{sup}^2 + 0.0008 \times T_{sup} + 0.458. \tag{2.9}$$

We conduct three types of thermal attacks (i.e, random, rack-level, and hotspot thermal attacks) on the targeted data center with different numbers of servers under attack. In thermal attacks, the position of the attacked servers plays an important role because of the air recirculation in the computer room. However, only in a hotspot thermal attack, an attacker can have control on the selection of racks and servers for mounting the attack. Due to the random selection of servers or racks in the random or rack-level thermal attacks, we conduct both attacks for 10 times and average the results. For the hotspot thermal attacks, we choose those servers located in hotspots, which have the largest thermal effects on the entire room.

Figure 2.14 shows the cooling costs under the three types of thermal attacks with different numbers of attacked servers, indicating that thermal attacks can significantly increase the cooling costs. Even with only 20 servers under a thermal attack, which is less than 2% of all servers, the cooling costs increase by about 5∼20% on average. The attack becomes more powerful with more



(a) Inlet Temperature.

(b) Supply Air Temperature.

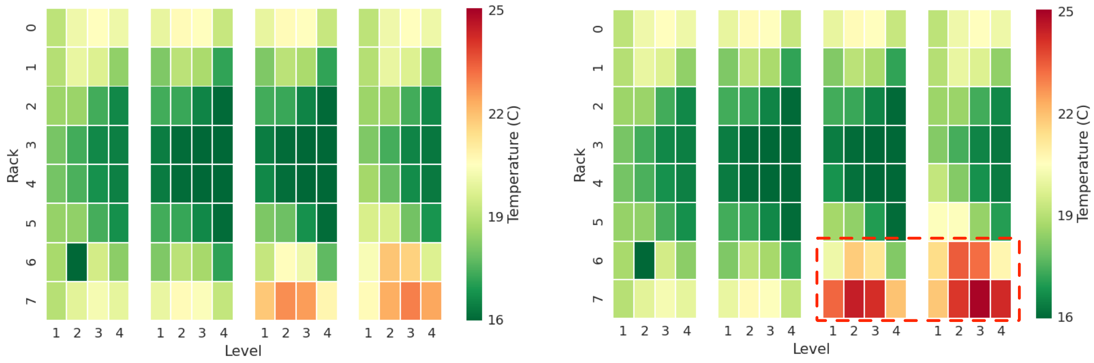
Figure 2.15: A Snapshot of a Thermal Attack on One Block of Servers.

servers under attack. For example, when 50 servers are under attack, the data center has to pay 58% more cooling costs compared to the normal case.

Among those three types of attacks, the hotspot attack again consumes the highest cooling costs compared to the other two attacks. It indicates that the locations of the attacked servers indeed impact the effectiveness of thermal attacks. Due to air recirculation, servers in specific locations can be used to magnify the heat generated by the attack. We can also see that the rack-level attack consumes more cooling costs than the random attack. The reason lies in the fact that, by attacking adjacent servers, the heat can be easily accumulated to form a local hotspot, forcing the data center to lower the temperature set point of the cooling system, resulting in higher cooling costs.

2.4.2.3 Cooling failures

A snapshot view. We simulate thermal attacks using the server trace obtained from one of the largest cloud service vendors. The targeted data center is supposed to have a smart scheduler that performs load balancing and adjusts the supply air temperature of CRACs in the room, with the minimum supply air temperature of 16°C [100]. The redline threshold for the inlet temperature is set to 25°C, which is higher than the set points used in most existing data centers. We simulate thermal attacks with thermal-intensive workloads. As shown in Figure 2.6 in Section 2.2, the



(a) Normal Condition.

(b) Thermal Attacks.

Figure 2.16: The Global Views of Thermal Conditions in a Computer Room.

temperature can be raised within 10 minutes. In our attack simulation, we gradually increase the number of attacked servers with a period of 15 minutes. The simulation results are presented in Figure 2.15, where Figure 2.15(a) shows the dynamics of the inlet temperature of our server block, which contains 10 servers. Figure 2.15(b) shows the dynamics of the supply air temperature of the CRACs. Before the attack starts, the supply air temperature set point is configured to 20°C. Once the thermal attack starts, the heat generated by the thermal attack forces the CRACs to gradually lower the temperature set point. As more and more servers go under attack, the temperature set point is decreased to 16°C. Finally, the cooling system fails to cool down the hotspot generated by the thermal attack. To avoid hardware damage, 10 servers are forced to shut down. A few minutes later, since no more heat is being generated, the scheduler adjusts the supply air temperature set point back to 17.3°C.

Global views. Figure 2.16 demonstrates the global views of thermal conditions in the targeted data center. Each block contains 10 servers and four blocks stack up as a rack. The level indicates the position of a block in a rack. For example, the level 4 means the block locates at the top of the rack. The supply air temperature is fixed as 16°C. The attacker first increases the utilization of controlled servers to the capping limit by running moderate workloads. Under this scenario as shown in Figure 2.16(a), the targeted data center still stays in a health thermal condition: the block with the highest temperature is about 22°C. After that, the attacker switches all moderate workloads to thermal-intensive workloads. As Figure 2.16(b) shows, although all servers' utilizations

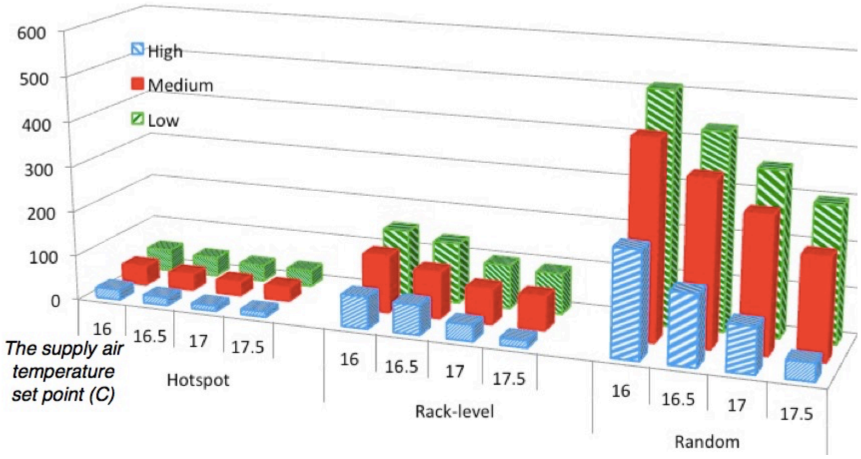


Figure 2.17: Number of Servers to Cause a Cooling Failure under Different Supply Air Temperature.

remain unchanged, the thermal condition seriously deteriorates. The inlet temperature of multiple blocks at the right corner is approaching the redline threshold, and one of them already surpasses the redline temperature (25°C). Such results indicate that existing utilization-based load balance cannot defend against thermal attacks.

Attack efficiency. We further explore the efficiency of different attack scenarios. We check the number of servers needed to successfully cause a local cooling system failure. We consider three scenarios: high, medium, and low, representing background workload utilizations around 60%, 40% and 20%, respectively, in the data center. We also consider the impacts of different supply air temperature. While the result for the hotspot attack is deterministic, for the random and rack-level attacks, due to the randomness in the or rack selection, we conduct each type of attack ten times and take the average. All the results are shown in Figure 2.17. It is clear that, with the same supply air temperature (e.g., 16°C), the hotspot attack requires the smallest number of attacked servers (e.g., 30-50) to cause a local cooling failure. Also, we are not surprised to see that the rack-level attack achieves much higher attack efficiency than the random attack, even though the rack selection is random. For a random attack, a large number of servers are required to cause a local cooling failure. That is reasonable as the cooling facility in a data center is designed to support all servers even if some of them may experience high power consumption at the same time. However, under extreme conditions, it is still possible to cause local hotspots even when the

Table 2.2: Attack Efficiency on a Large Data Center.

	Low	**Medium**	**High**
Rack-level	608	556	292

CRAC cooling system is working properly.

The results further suggest that a thermal attack is more effective when the data center workload is in the high scenario, where the data center is already in a hot environment. Thus, by running thermal intensive workloads in the high scenario, it is easier to exhaust the remainder cooling redundancy and cause a cooling failure. As the usage of data centers normally follows specific patterns and might be leaked by side-channels, it is not difficult for a well-motivated attacker to detect the occurrence of the high scenario.

Figure 2.17 illustrates the number of servers needed to cause a cooling failure under different supply air temperature. It also clearly indicates that the higher the supply air temperature set by data centers, the smaller efforts are needed for attackers to cause cooling failures. More specifically, with the supply air temperature raising by 1°C (e.g., from 16°C to 17°C), the number of required servers is reduced to half. These results imply that although a higher temperature set point of the cooling system is able to significantly lower the cooling costs, as suggested in [38], this management strategy also makes a data center more vulnerable to thermal attacks. While cloud vendors might like a data center hot to reduce cooling cost, malicious attackers will also like the data center hot to much more easily launch a thermal attack. Thus, if today's data centers should continue raising their temperature, thermal attacks would become a serious threat to these future more cooling-efficient data centers.

Extension to a larger scale. We extend the data center model to include 10,240 servers to evaluate the impacts of thermal attacks in a larger scale. For hotspot attacks, we achieve quite similar results as before: with the knowledge of the locations of controlled servers, just tens of servers are enough to cause a local hotspot. We list the results of rack-level attacks in Table 2.2. As we can see, if the attacker can achieve rack-level co-residence, controlling 5% of servers is sufficient to cause a cooling failure. Note that the rack selection is still random in this rack-level attack model. Thus, it is feasible for an advanced attacker to cause a local hotspot and even a cooling failure in a large data center. Moreover, such a potential threat would be more serious in the future given the fact that cloud vendors have been continuing to deploy more servers in their data centers to meet the rapidly increasing cloud service demands.

Financial losses caused by cooling failures. Cooling failures are devastating to cloud vendors and various services hosted in their data centers. Since one physical server could be shared by dozens of VMs in a cloud environment, the shutdown of even a single server would lead to the disruption of many services and significant financial losses. It is reported that the four-hour outage of Amazon Web Services S3 in February 2017 caused an incredible loss of $150 million for S&P 500 enterprises, plus with a loss of $160 million for U.S. financial services [2]. In 2016, Delta Airline also suffered a loss of $150 million in a five-hour disruption of their data center [6].

2.4.3 Attack Cost Analysis

The cost of mounting a thermal attack against a data center mainly lies in subscribing the computing services offered in the data center, in which the dedicated server hosting is the most expensive service and its usage sets the upper-limit of the attack cost. A dedicated server service enables a client to own an entire physical server without any resource sharing with others, allowing a thermal attack to be mounted by fully exploiting its computing sources for heat generation.

The price of an Amazon EC2 dedicated server with 20 Cores is $1.84 per hour [1]. Assume that each thermal attack uses 50/100/400 dedicated servers and lasts 60 minutes as our simulation results show the cooling failure occurs within one hour of the thermal attack being launched. Therefore, to own 50/100/400 dedicated servers for one hour, the attacker just needs to pay $92/$184/$736, for mounting a hotspot attack, a rack-level attack, or a random attack, respectively. However, based on our simulation results, such thermal attacks are powerful enough to cause a cooling failure in a computer room with 1,280 servers.

Compared with service subscription costs, the other costs for mounting a thermal attack is minor. Although it is technically challenging, there is no extra financial cost for conducting advanced network probing to explore the layout of a data center. Even if VPC (Virtual Private Cloud) has been deployed to lower the risk of tenant co-residence, a recent work proposes cost-effective attack strategies to achieve co-residence without much cost. For example, it just costs 14 cents to have a more than 90 percent chance of achieving co-residence on Google Computing Engine [112].

2.5 Defense Approaches

The root cause of thermal attacks is the adoption of aggressive cooling energy saving policies in the data centers, which results in heavily reduced cooling redundancies. Although this thermal vulnerability cannot be completely fixed without the increase of cooling redundancies, a well-designed load balancing strategy could mitigate the thermal attacks. Obviously, the traditional utilization-based load balancing strategy cannot effectively defend against a thermal attack, as shown in Section 2.2. The previous temperature-based load balancing approaches (e.g., [84]) relying on a *static* thermal profile running with normal workloads cannot ward off the threat either. This is because the thermal conditions could be dramatically changed due to thermal attacks, even completely opposite to the static pre-calculated profile. The absence of fine-grained sensors further limits the effectiveness of those approaches.

Note that a high power consumption process or high temperature core not only affects itself, but also impacts its neighboring servers. Thus, without a global knowledge of the thermodynamics in a computer room, any single server based capping mechanism (e.g., power capping) alone cannot fully address the security threat posed by thermal attacks.

To mitigate the thermal attacks, we propose a dynamic thermal-aware load balancing approach at the rack and data center levels. Our thermal-aware load balancing allocates workloads with the consideration of the thermal conditions and physical locations of servers. Based on the thermal condition in a rack or the entire data center, thermal-intensive workloads would be dynamically distributed to servers to limit their thermal impacts. Such a mechanism can effectively mitigate the negative impacts of a local hotspot, allowing further actions like dynamic voltage and frequency scaling to be taken inside the local hotspot for temperature reduction.

We implement a simple prototype of the proposed thermal-aware load balancing mechanism and conduct one experiment to demonstrate its effectiveness. In our prototype, we maintain a list of

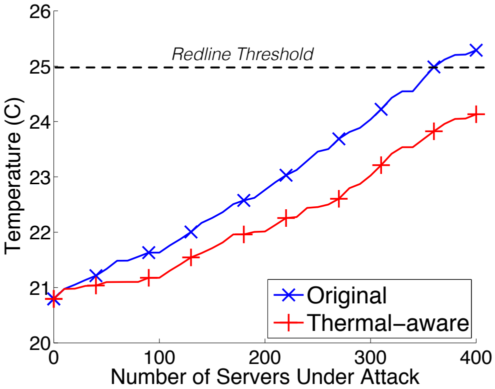


Figure 2.18: Inlet Temperature of a Local Hotspot.

spots that have large air recirculation effects, especially under thermal attacks, due to their physical locations. The VMs of these spots are less likely to be allocated to users. The configuration of the targeted data center is the same as Section 2.4. The initial system utilization is 25%.

In our experiment, we conduct a thermal attack using thermal-intensive workloads that consume 20% more power consumption than regular workloads. The attacker gradually increases the number of servers under attack. We fix the supply air temperature as 16°C. We measure the inlet temperature of the local hotspot that has the highest temperature under different number of attack servers. The result is illustrated in Figure 2.18, in which the "+" line indicates the results using thermal-aware load balancing and the "x" line represents the results under the original utilization-based load balancing policy. Whereas the cooling system cannot keep the hotspot below the redline threshold using the original policy, our simple thermal-aware load balancing is able to lower the temperature by more than 1°C and keep the hotspot safe under the thermal attack.

Overall, the proposed thermal-aware load balancing is simple and straightforward to implement, and it can significantly improve the robustness of a data center against thermal attacks. However, in an oversubscribed data center, such a reactive thermal management will still fail to handle much more severe attacks with much more servers running malicious thermal-intensive workloads simultaneously. In our future work, we plan to pursue more advanced proactive defense strategies to detect the occurrence of thermal anomalies in real-time and prevent potential hotspots from overheating themselves and surrounding servers. The more effective defense systems are expected to have the following capabilities: (1) robust anomaly detection with an emphasis on thermal-intensive workloads at chip and server levels; (2) cost-effective sensing solutions to profile the thermal dynamics by deploying sensors with the consideration of costs, networking traffics, locations and sense ranges of those additional sensors; and (3) proactive thermal management on rack and data center levels by exploiting energy storage such as phase change materials [104].

2.6 Related Work

Thermal/power threats. There are some previous works focus on thermal attacks on individual components of machines. Kong et al. [69] studied thermal attacks on instruction caches. They run malicious code to heat other places such as the back end of the cache, instead of those traditional hotspots, and thus avoid the temperature regulation by the dynamic thermal management (DTM). A heating attack on flash memory devices is studied in [105]. A memory cell inside a memory array is locally heated up by an inexpensive laser-diode module. As a result, the contents of the memory can be altered and compromised. Paul et al. [91] launched a thermal attack on disk storages. They used intensive hot seeks that maximize the power consumption of the disk arm to rapidly and repeatedly increase the temperature. The disk is throttled due to DTM. Hasan et al. [57] conducted a heat based attack on Simultaneous Multithreading (SMT) by repeatedly accessing a shared source to create a hotspot in one malicious thread. Thermal covert channel attacks based on CPU cores' temperature variations have been presented in [26, 82]. While these works provide a guidance on thermal attacks on individual components, they do not study the impacts of thermal attacks on servers and data centers.

Wu et al. [117] conducted energy attacks on servers. They manipulated malicious requests that can cause a large number of cache misses and achieved up to 42.3% power consumption increment. Xu et al. [123] demonstrated the vulnerability of cloud services to power attack due to power oversubscription at data centers. An attacker can force victim servers to reach their power peaks at the same time and then trip the circuit breaker. Islam et al. [61] further proposed using a hot air recirculation based thermal side channel to infer the power usage of benign users and then launch power attacks when the aggregated power consumption of benign users is high. Even battery-backed data centers are vulnerable to such attacks [74]. Power attack [48] differs from thermal attack in that power attack attempts to generate a power spike within a very short period and cause a power outage. In comparison with power attack, thermal attack is more stealthy but its damage could be as serious as that of power attack. The overheat induced by thermal attack leads to hardware failure and even shutdown for self-protection.

Cooling in data center. Different cooling strategies in data centers have been proposed in the past [52,60,77,80,84]. Currently a popular trend is leveraging CFD to simulate the air recirculation in computing rooms [75]. The introduction of CFD helps a data center to place CRAC and servers in an efficient way and thus saves cooling cost. However, those cooling strategies only focus on the optimization of the cooling power in data centers, without considering the risk of a thermal attack.

Impacts of temperature on servers. The impact of temperature on servers has been investigated for years. El-Sayed et al. [38] collected a large amount of field data from different data centers to study the impact of temperature on hardware reliability. They found that temperature variation exhibits strong correlation to hardware failures and server outages. Sankar et al. [101] also used a large collection of data to study the correlation between temperature and hard disk failures. They reported that temperature is strongly correlated with disk failures. Coskun et al. [36] studied the reliability of different job scheduling and power management methods. They proposed a fine grained technique to simulate the thermal behaviors and input the thermal trace to the reliability model. PGCapping [79] integrates various techniques to optimize the multiprocessor performance within a power cap.

2.7 Chapter Summary

In this chapter, we have presented a new security threat called thermal attack on data centers. We first conduct a real server measurement study to systematically investigate the impacts of different factors on thermal conditions of a physical server. Based on the measurement results, we propose effective attack vectors to mount thermal attacks on both virtualized and non-virtualized environments. To evaluate the impacts of thermal attack on a data center, we further simulate rack-level and datacenter-level thermal attacks under three different attack scenarios using a real-world data center trace. We present the damage analysis at both the server and data center levels. Our evaluation and analysis results demonstrate that thermal attack can degrade the performance and reliability of victim servers, cause local hotspots, increase the cooling cost, and even worse lead to cooling failures, in which some servers are forced to shut down for overheat protection. Finally, we present a simple thermal-aware load balancing mechanism to help data centers defend against thermal attacks, which paves the way for more sophisticated defenses.

Chapter 3

ContainerLeaks: Synergistic Power Attack via Information Leakage in Multi-Tenancy Container Cloud Services

In this chapter, we systematically explore and identify the in-container leakage channels that may accidentally expose information of host OSes and co-resident containers. Such information leakages include host-system state information (e.g., power consumption, performance data, global kernel data, and asynchronous kernel events) and individual process execution information (e.g., process scheduling, cgroups, and process running status). The distinguishing characteristic information exposed at specific timings could help uniquely identify a physical machine. Furthermore, a malicious tenant may optimize attack strategies and maximize attack effects by acquiring the system-wide knowledge in advance. We discover these leakage channels in our local testbed on *Docker* and *LinuX Container (LXC)* and verify their (partial) existence on five public commercial multi-tenancy container cloud services.

In order to reveal the security risks of these leakage channels, we design an advanced attack, denoted as *synergistic power attack*, to exploit the seemingly innocuous information leaked through these channels. We demonstrate that such information exposure could greatly amplify the attack effects, reduce the attack costs, and simplify the attack orchestration. Power attacks have proved to be real threats to existing data centers [74, 123]. With no information of the running status of underlying cloud infrastructures, existing power attacks can only launch power-intensive workloads blindly to generate power spikes, with the hope that high spikes could trip branch circuit breakers to cause power outages. Such attacks could be costly and ineffective. However, by learning the system-wide status information, attackers can (1) pick the best timing to launch an attack, i.e., superimpose the power-intensive workload on the existing power spikes triggered by benign workloads, and (2) synchronize multiple power attacks on the same physical machine/rack by detecting proximity-residence of controlled containers. We conduct proof-of-concept experiments on one real-world container cloud service and quantitatively demonstrate that our attack is able to yield higher power spikes at a lower cost.

We further analyze in depth the root causes of these leakage channels and find that such exposures are due to the incomplete coverage of container implementation in the Linux kernel. We propose a two-stage defense mechanism to address this problem in container clouds. In particular, to defend against the *synergistic power attacks*, we design and implement a power-based namespace in the Linux kernel to partition power consumption at a finer-grained (container) level. We evaluate our power-based namespace from the perspectives of accuracy, security, and performance overhead. Our experimental results show that our system can neutralize container-based power attacks with trivial performance overhead.

The chapter is organized as follows. Section 3.1 introduces the background of container technology and describes existing power attack threats on data centers. Section 3.2 presents the cross-container leakage channels discovered by us and their leaked information. Section 3.3 details the synergistic power attack that leverages the leaked information through these channels. Section 3.4 presents the design and implementation of our power-based namespace on the Linux kernel. Section 3.5 shows the evaluation of our defense framework from different aspects. Section 3.6 discusses the limitations and the future work. Section 3.7 surveys related work, and finally Section 3.8 concludes.

3.1 Background

In this section, we briefly describe the background knowledge of three topics: internals of Linux containers, multi-tenancy container cloud services, and existing power attacks in data centers.

3.1.1 Linux Kernel Support for Container Technology

Containers depend on multiple independent Linux kernel components to enforce isolation among user-space instances. Compared to VM-based virtualization approaches, multiple containers share the same OS kernel, thus eliminating additional performance overheads for starting and maintaining VMs. Containers have received much attention from the industry and have grown rapidly in recent years for boosting application performance, enhancing developer efficiency, and facilitating service deployment. Here we introduce two key techniques, *namespace* and *cgroup*, that enable containerization on Linux.

3.1.1.1 Namespace

The first namespace was introduced in the Linux kernel 2.4.19. The key idea of namespace is to isolate and virtualize system resources for a group of processes, which form a container. Each process can be associated with multiple namespaces of different types. The kernel presents a customized (based on namespace types) view of system resources to each process. The modifications to any namespaced system resources are confined within the associated namespaces, thus incurring no system-wide changes.

The current kernel has seven types of namespaces: *mount (MNT)* namespace, *UNIX timesharing system (UTS)* namespace, *PID* namespace, *network (NET)* namespace, *inter-process communications (IPC)* namespace, *USER* namespace, and *CGROUP* namespace. The *MNT* namespace isolates a set of file system mount points. In different *MNT* namespaces, processes have different views of the file system hierarchy. The *UTS* namespace allows each container to have its own host name and domain name, and thus a container could be treated as an independent node. The *PID* namespace virtualizes the process identifiers (pids). Each process has two pids: one pid within its *PID* namespace and one (globally unique) pid on the host. Processes in one container could only view processes within the same *PID* namespace. A *NET* namespace contains separate virtual network devices, IP addresses, ports, and IP routing tables. The *IPC* namespace isolates inter-process communication resources, including signals, pipes, and shared memory. The *USER* namespace was recently introduced to isolate the user and group ID number spaces. It creates a mapping between a root user inside a container to an unprivileged user on the host. Thus, a process may have full privileges inside a user namespace, but it is de-privileged on the host. The *CGROUP* namespace virtualizes the cgroup resources, and each process can only have a containerized cgroup view via *cgroupfs* mount and the */proc/self/cgroup* file.

3.1.1.2 Cgroup

In the Linux kernel, cgroup (i.e., control group) provides a mechanism for partitioning groups of processes (and all their children) into hierarchical groups with controlled behaviors. Containers leverage the cgroup functionality to apply per-cgroup resource limits to each container instance, thus preventing a single container from draining host resources. Such controlled resources include CPU, memory, block IO, network, etc. For the billing model in cloud computing, cgroup can also be used for assigning corresponding resources to each container and accounting for their usage. Each cgroup subsystem provides a unified *sysfs* interface to simplify the cgroup operations from the user space.

3.1.2 Container Cloud

With these kernel features available for resource isolation and management, the Linux kernel can provide the lightweight virtualization functionality at the OS level. More namespace and cgroup subsystems are expected to be merged into the upstream Linux kernel in the future to enhance the container security. Containerization has become a popular choice for virtual hosting in recent years with the maturity of container runtime software. *LXC* is the first complete implementation of the Linux container manager built in 2008. *Docker*, which was built upon *LXC* (now with *libcontainer*), has become the most popular container management tool in recent years. *Docker* can wrap applications and their dependencies (e.g., code, runtime, system tools, and system libraries) into an image, thus guaranteeing that application behaviors are consistent across different platforms.

A large number of cloud service providers, including Amazon ECS, IBM Bluemix, Microsoft Azure, and Google Compute Engine, have already provided container cloud services. For multi-tenancy container cloud services, containers can either run on a *bare metal* physical machine or a *virtual machine*. In both situations, containers from different tenants share the same Linux kernel with the host OS.

3.1.3 Power Attacks on Data Centers

Power attacks have been demonstrated to be realistic threats to existing cloud infrastructures [74, 123]. Considering the cost of upgrading power facilities, current data centers widely adopt power oversubscription to host the maximum number of servers within the existing power supply capabilities. The safety guarantees are based on the assumption that multiple adjacent servers have a low chance of reaching peaks of power consumption simultaneously. While power oversubscription allows deploying more servers without increasing power capacity, the reduction of power redundancy increases the possibility of power outages, which might lead to forced shutdowns for servers on the same rack or on the same power distribution unit (PDU). Even normal workloads may generate power spikes that cause power outages. Facebook recently reported that it prevented 18 potential power outages within six months in 2016 [116]. The situation would have been worse if malicious adversaries intentionally drop power viruses to launch power attacks [46, 47]. The consequence of a power outage could be devastating, e.g., Delta Airlines encountered a shutdown of a power source in its data center in August 2016, which caused large-scale delays and cancellations of flights [18]. Recent research efforts [74, 123] have demonstrated that it is feasible to mount power attacks on both traditional and battery-backed data centers.

Launching a successful power attack requires three key factors: (1) gaining access to servers in the target data center by legitimately subscribing services, (2) steadily running moderate workloads to increase the power consumption of servers to their capping limits, (3) abruptly switching to power-intensive workloads to trigger power spikes. By causing a power spike in a short time window, a circuit breaker could be tripped to protect servers from physical damages caused by overcurrent or overload.

The tripping condition of a circuit breaker depends on the strength and duration of a power spike. In order to maximize the attack effects, adversaries need to run malicious workloads on a group of servers belonging to the same rack or PDU. In addition, the timing of launching attacks is also critical. If a specific set of servers (e.g., on the same rack) in a data center have already run at their peak power state, the chance of launching a successful power attack will be higher [123].

The techniques of power capping [73] have been designed to defend against power attacks. At the rack and PDU level, by monitoring the power consumption, a data center can restrict the power consumption of servers through a power-based feedback loop. At the host level, *Running Average Power Limit (RAPL)* is a technique for monitoring and limiting the power consumption for a single server. *RAPL* has been introduced by Intel since Sandy Bridge microarchitecture. It provides fine-grained CPU-level energy accounting at the microsecond level and can be used to limit the power consumption for one package.

Power capping mechanisms significantly narrow down the power attack surface, but it cannot address the problem of power oversubscription, which is the root cause of power outages in data centers. Although host-level power capping for a single server could respond immediately to power surges, the power capping mechanisms at the rack or PDU level still suffer from minute-level delays. Assuming attackers could deploy power viruses into physically adjacent servers, even if each server consumes power lower than its power capping limit, the aggregated power consumption of controlled servers altogether can still exceed the power supply capacity and trip the circuit breaker. We demonstrate in the following sections that malicious container tenants can launch *synergistic power attacks* by controlling the deployment of their power-intensive workloads and leveraging benign workloads in the background to amplify their power attacks.

3.2 Information Leakages in Container Clouds

As we mentioned in Section 3.1, the Linux kernel provides a multitude of supports to enforce resource isolation and control for the container abstraction. Such kernel mechanisms are the enabling techniques for running containers on the multi-tenancy cloud. Due to priority and difficulty

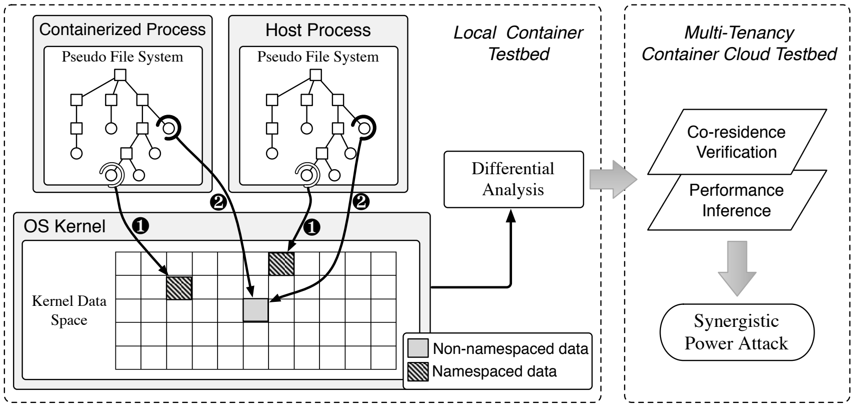


Figure 3.1: The Framework for Information Leakage Detection and Cloud Inspection.

levels, some components of the Linux kernel have not yet transformed to support containerization. We intend to systematically explore which parts of the kernel are left uncovered, what the root causes are, and how potential adversaries can exploit them.

3.2.1 Container Information Leakages

We first conduct experiments on our local Linux machines with *Docker* and *LXC* containers installed. Linux provides two types of controlled interfaces from userspace processes to the kernel, system calls, and memory-based pseudo file systems. System calls are mainly designed for user processes to request kernel services. The system calls have strict definitions for their public interfaces and are typically backward compatible. However, memory-based pseudo file systems are more flexible for extending kernel functionalities (e.g., *ioctl*), accessing kernel data (e.g., *procfs*), and adjusting kernel parameters (e.g., *sysctl*). In addition, such pseudo file systems enable manipulating kernel data via normal file I/O operations. Linux has a number of memory-based pseudo file systems (e.g., *procfs*, *sysfs*, *devfs*, *securityfs*, *debugfs*, etc.) that serve the different purposes of kernel operations. We are more interested in *procfs* and *sysfs*, which are by default mounted by container runtime software.

As illustrated in the left part of Figure 3.1, we design a cross-validation tool to automatically discover these memory-based pseudo files that expose host information to containers. The key idea is to recursively explore all pseudo files under *procfs* and *sysfs* in two execution contexts, one

Table 3.1: Leakage Channels in Commercial Container Cloud Services.

Leakage Channels	Leakage Information	Potential Vulnerability			Container Cloud Services[1]				
		Co-re	DoS	Info leak	CC_1	CC_2	CC_3	CC_4	CC_5
`/proc/locks`	Files locked by the kernel	●	○	●	●	●	●	●	◐
`/proc/zoneinfo`	Physical RAM information	●	○	●	●	●	●	●	◐
`/proc/modules`	Loaded kernel modules information	○	○	●	●	●	●	●	●
`/proc/timer_list`	Configured clocks and timers	●	○	●	●	●	●	○	●
`/proc/sched_debug`	Task scheduler behavior	●	○	●	○	○	●	○	●
`/proc/softirqs`	Number of invoked softirq handler	●	●	●	●	●	●	●	●
`/proc/uptime`	Up and idle time	●	○	●	●	●	●	●	◐
`/proc/version`	Kernel, gcc, distribution version	○	○	●	●	●	●	●	●
`/proc/stat`	Kernel activities	●	●	●	●	●	●	●	◐
`/proc/meminfo`	Memory information	●	●	●	●	●	●	●	○
`/proc/loadavg`	CPU and IO utilization over time	●	○	●	●	●	●	●	◐
`/proc/interrupts`	Number of interrupts per IRQ	●	○	●	●	●	●	●	●
`/proc/cpuinfo`	CPU information	●	○	●	●	●	●	●	○
`/proc/schedstat`	Schedule statistics	●	○	●	●	●	●	●	◐
`/proc/sys/fs/*`	File system information	●	○	●	●	●	○	●	●
`/proc/sys/kernel/random/*`	Random number generation info	●	○	●	●	●	●	●	●
`/proc/sys/kernel/sched_domain/*`	Schedule domain info	●	○	●	●	●	●	●	●
`/proc/fs/ext4/*`	Ext4 file system info	●	○	●	●	●	●	●	●
`/sys/fs/cgroup/net_prio/*`	Priorities assigned to traffic	○	○	●	●	●	○	○	○
`/sys/devices/*`	System device information	●	●	●	●	●	●	○	○
`/sys/class/*`	System device information	○	●	●	●	●	●	○	○

running within an unprivileged container and the other running on the host. We align and reorder the files based on their file paths and then conduct pair-wise differential analysis on the contents of the same file between these two contexts. If the system resources accessed from a specific pseudo file has not been namespaced in the Linux kernel, the host and container reach the same piece of kernel data (as the case of ❷ in Figure 3.1). Otherwise, if properly namespaced, the container can retrieve its own private and customized kernel data (as the case of ❶ in Figure 3.1). Using this cross-validation tool, we can quickly identify the pseudo files (and their internal kernel data structures) that may expose system-wide host information to the container.

3.2.2 Leakage Channel Analysis

[1] The most recent check on the leakage channels was made on November 28, 2016.

We list all pseudo files that may leak host information in Table 3.1. Those leakage channels contain different aspects of host information. Container users can retrieve kernel data structures (e.g., */proc/modules* shows the list of loaded modules), kernel events (e.g., */proc/interrupts* shows the number of interrupts per IRQ), and hardware information (e.g., */proc/cpuinfo* and */proc/meminfo* show the specification of CPU and memory, respectively). In addition, container users are able to retrieve performance statistics data through some channels. For example, containers can obtain hardware sensor data (if these sensors are available in the physical machine), such as power consumption for each package, cores, and DRAM through the *RAPL sysfs* interface, and the temperature for each core through the *Digital Temperature Sensor (DTS) sysfs* interface. Moreover, the usage of processors, memory, and disk I/O is also exposed to containers. While leaking such information seems harmless at first glance, it could be exploited by malicious adversaries to launch attacks. More detailed discussion is given in Section 3.3.

We further investigate the root causes of these information leakages by inspecting the kernel code (in the Linux kernel version 4.7). Generally, the leakage problems are caused by the incomplete implementation of namespaces in the kernel. To be more specific, we summarize the two main causes as follows: (1) Context checks are missing for existing namespaces, and (2) some Linux subsystems are not (fully) namespaced. We give two case studies on *net_prio.ifpriomap* and *RAPL in containers* to reveal the origins of leakages.

3.2.2.1 Case study I — net_prio.ifpriomap

The pseudo file *net_prio.ifpriomap* (under */sys/fs/cgroup/net_prio*) contains a map of the priorities assigned to traffic starting from processes in a cgroup and leaving the system on various interfaces. The data format is in the form of *[ifname priority]*. We find that the kernel handler function hooked at *net_prio.ifpriomap* is not aware of the *NET* namespace, and thus it discloses all network interfaces on the physical machine to the containerized applications. To be more specific, the read operation of *net_prio.ifpriomap* is handled by the function `read_priomap`. Tracing from this function, we find that it invokes `for_each_netdev_rcu` and sets the first parameter as the address of `init_net`. It iterates all network devices of the host, regardless of the *NET* namespace. Thus, from the view of a container, it can read the names of all network devices of the host.

3.2.2.2 Case study II — *RAPL* in containers

RAPL was recently introduced by Intel for setting power limits for processor packages and DRAM of a single server, which can respond at the millisecond level [55]. In the container cloud, the *sysfs* interface of RAPL, which locates under */sys/class/powercap/intel-rapl*, is accessible to containers. Therefore, it is possible for container tenants to obtain the system-wide power status of the host, including the core, DRAM, and package, through this *sysfs* interface. For example, container users can read the current energy counter in micro joules from the pseudo file *energy_uj*. The function handler of *energy_uj* in the Intel *RAPL* Linux driver is `get_energy_counter`. This function retrieves the raw energy data from the *RAPL MSR*. As namespace has not been implemented for the power data, the `energy_raw` pointer refers to the host's energy consumption data.

We further investigate the information leakage problems on container cloud services that adopt the *Docker*/*LXC* container engine. We choose five commercial public multi-tenancy container cloud services for leakage checking and present the results in Table 3.1. We anonymize the names (**CC$_\mathbf{i}$** stands for **i**th **C**ontainer **C**loud) of these container cloud services before the cloud providers patch the channels. The ● indicates that the channel exists in the cloud, while the ○ indicates the opposite. We find that most of the leakage channels on local machines are also available in the container cloud services. Some of them are unavailable due to the lack of support for specific hardware (e.g., Intel processor before Sandy Bridge or AMD processors that do not support RAPL). For cloud **CC$_\mathbf{5}$**, we find that the information of some channels is different from our local testbed, which means that the cloud vendor has customized some additional restrictions. For example, only the information about the cores and memory belonging to a tenant is available. However, those channels partially leak the host information and could still be exploited by advanced attackers. We mark them as ◐.

3.2.3 Inference of Co-resident Container

We further look in depth into specific cases to see whether they could be exploited to detect co-resident containers.

3.2.3.1 Co-residence problems in cloud settings

Co-residence is a well-known research problem in cloud security. In order to extract a victim's information, adversaries tend to move malicious instances to the same physical host with the victim. Zhang et al. have shown that it is possible for an attacker to hijack user accounts [131] and extract private keys [130] with co-resident instances. In addition, the cost of achieving co-residence is rather low [112]. Co-residence still remains a problem in existing clouds, due to the intention of consolidating server resources and reducing cost. Traditional methods to verify co-residence are based on cache [127] or memory-based leakage channels [118]. The accuracy of those methods may downgrade due to the high noise in cloud settings.

3.2.3.2 Approaches and results of checking co-resident containers

Since containers can read the host information through the leakage channels we discovered, we tend to measure whether some channels can be used for checking container co-residence. We define three metrics, namely *uniqueness* ($\mathbb{U}$), *variation* ($\mathbb{V}$), and *manipulation* ($\mathbb{M}$) to quantitatively assess each channel's capability of inferring co-residence.

The metric $\mathbb{U}$ indicates whether this channel bestows characteristic data that can uniquely identify a host machine. It is the most important and accurate factor for determining whether two containers locate on the same host. We have found 17 leakage channels (ranked top 17 in Table 3.2) that satisfy this metric. Generally we can classify these channels into three groups:

1) Channels containing unique static identifiers. For example, *boot_id* under */proc/sys/kernel/random* is a random string generated at boot time and is unique for each running kernel. If two containers can read the same *boot_id*, this is a clear sign that they are running on the same host kernel. The data for channels in this group are both static and unique.

2) Channels into which container tenants can dynamically implant unique signatures. For example, from */proc/sched_debug*, container users can retrieve all active process information of the host through this interface. A tenant can launch a process with a uniquely crafted task name inside the container. From the other containers, they can verify co-residence by searching this task name in their own *sched_debug*. Similar situations apply to *timer_list* and *locks*.

3) Channels containing unique dynamic identifiers. For example, */proc/uptime* has two data fields: system up time and system idle time in seconds since booting. They are accumulated values and are unique for every host machine. Similarly, *energy_uj* in the *RAPL sysfs* interface is the accumulated energy counter in micro joules. The data read from channels in this group change at real time, but are still unique to represent a host. We rank the channels in this group based on their growth rates. A faster growth rate indicates a lower chance of duplication.

The metric $\mathbb{V}$ demonstrates whether the data change with time. With this feature available, two containers can make snapshots of this pseudo file periodically at the same time. Then they can determine co-residence by checking whether two data snapshot traces match with each other. For example, starting from the same time, we can record *MemFree* in */proc/meminfo* from two containers every second for one minute. If these two 60-point data traces match with each other, we are confident that these two containers run on the same host. Each channel contains a different capacity of information for inferring co-residence, which can be naturally measured via the *joint Shannon entropy*. We define the entropy $\mathbb{H}$ in Formula (3.1). Each channel C contains multiple independent data fields X_i, and n represents the number of independent data fields. Each X_i has possible values $\{x_{i1}, \cdots, x_{im}\}$. We rank the capability of revealing co-residence for the nine channels (for which $\mathbb{U}$=*False* and $\mathbb{V}$=*True*) based on their entropy results in Table 3.2.

$$\mathbb{H}[C(X_1, \cdots, X_n)] = \sum_{i=1}^{n}[-\sum_{j=1}^{m} p(x_{ij}) \log p(x_{ij})]. \tag{3.1}$$

The metric $\mathbb{M}$ indicates whether the container tenants can *manipulate* the data. We mark a channel ● if tenants can directly implant specially-crafted data into it. For example, we can create a timer in a program with a special task name inside a container. This task name and its associated timer will appear in */proc/timer_list*. Another container can search for this special task name in the *timer_list* to verify co-residence. We mark a channel ◐ if tenants can only indirectly influence the

Table 3.2: Leakage Channels for Co-Residence Verification.

Leakage Channels	$\mathbb{U}$	$\mathbb{V}$	$\mathbb{M}$	**Rank**
/proc/sys/kernel/random/boot_id	●	○	○	█████
/sys/fs/cgroup/net_prio/net_prio.ifpriomap	●	○	○	█████
/proc/sched_debug	●	●	●	█████
/proc/timer_list	●	●	●	█████
/proc/locks	●	●	●	█████
/proc/uptime	●	●	◐	▓▓▓▓
/proc/stat	●	●	◐	▓▓▓▓
/proc/schedstat	●	●	◐	▓▓▓▓
/proc/softirqs	●	●	◐	▓▓▓▓
/proc/interrupts	●	●	◐	▓▓▓▓
/sys/devices/system/node/node#/numastat	●	●	◐	▓▓▓▓
/sys/class/powercap/.../energy_uj[2]	●	●	◐	▓▓▓▓
/sys/devices/system/.../usage[3]	●	●	◐	▓▓▓▓
/sys/devices/system/.../time[4]	●	●	◐	▓▓▓▓
/proc/sys/fs/dentry-state	●	●	◐	▒▒▒
/proc/sys/fs/inode-nr	●	●	◐	▒▒▒
/proc/sys/fs/file-nr	●	●	◐	▒▒▒
/proc/zoneinfo	○	●	◐	░░
/proc/meminfo	○	●	◐	░░
/proc/fs/ext4/sda#/mb_groups	○	●	◐	░░
/sys/devices/system/node/node#/vmstat	○	●	◐	░░
/sys/devices/system/node/node#/meminfo	○	●	◐	░░
/sys/devices/platform/.../temp#_input[5]	○	●	◐	□
/proc/loadavg	○	●	◐	□
/proc/sys/kernel/random/entropy_avail	○	●	◐	□
/proc/sys/kernel/.../max_newidle_lb_cost[6]	○	●	○	□
/proc/modules	○	○	○	▯
/proc/cpuinfo	○	○	○	▯
/proc/version	○	○	○	▯
				Low High

data in this channel. For example, an attacker can use *taskset* command to bond a computing-intensive workload to a specific core, and check the CPU utilization, power consumption, or temperature from another container. Those entries could be exploited by advanced attackers as covert channels to transmit signals.

For those channels that do not have these $\mathbb{U}$ $\mathbb{V}$ $\mathbb{M}$ properties, we consider them hard to be ex-

[2] /sys/class/powercap/intel-rapl:#/intel-rapl:#/energy_uj

[3] /sys/devices/system/cpu/cpu#/cpuidle/state#/usage

[4] /sys/devices/system/cpu/cpu#/cpuidle/state#/time

[5] /sys/devices/platform/coretemp.#/hwmon/hwmon#/temp#_input

[6] /proc/sys/kernel/sched_domain/cpu#/domain#/max_newidle_lb_cost

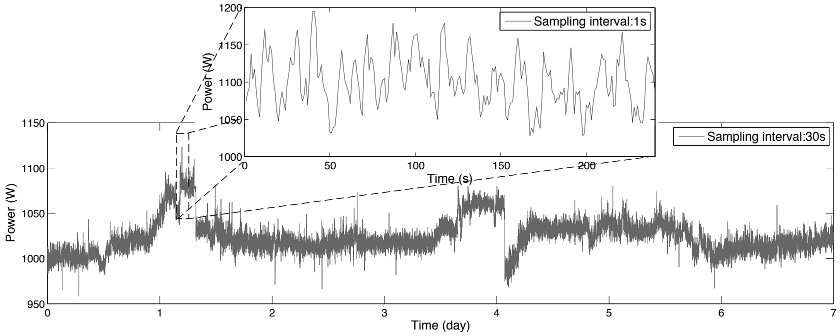


Figure 3.2: The Power Consumption for 8 Servers in One Week.

ploited. For example, most servers in a cloud data center probably install the same OS distribution with the same module list. Although */proc/modules* leaks the information of loaded modules on the host, it is difficult to use this channel to infer co-resident containers.

3.3 Synergistic Power Attack

At first glance, the leaked information discovered in Section 3.2 seems difficult to exploit. Because both *procfs* and *sysfs* are mounted read-only inside the containers, malicious tenants can only read such information, but modification is not allowed. We argue that attackers can make better decisions by learning the runtime status of the host machine.

In this section, we present a potential synergistic power attack in the scope of power outage threats that may impact the reliability of data centers. We demonstrate that adversaries can exploit these information leakages discovered by us to amplify the attack effects, reduce the attack costs, and facilitate attack orchestration. All experiments are conducted in a real-world container cloud.

3.3.1 Attack Amplification

The key to launching a successful power attack is to generate a short-time high power spike that can surpass the power facility's supply capacity. As we mentioned in 3.1.3, the root cause of power attacks is the wide adoption of power oversubscription, which makes it possible for power spikes to surpass the safe threshold. In addition, a rack-level power capping mechanism can only react in minute-level time granularity, leaving space for the occurrence of a short-time high power spike.

In the most critical situation, the overcharging of power may trip the branch circuit breaker, cause a power outage, and finally bring down the servers. The heights of power spikes are predominantly determined by the resources that are controlled by attackers. Existing power attacks maximize the power consumption by customizing power-intensive workloads, denoted as power viruses. For example, Ganesan et al. [46, 47] leveraged genetic algorithms to automatically generate power viruses that consume more power than normal *stress* benchmarks. However, launching a power attack from scratch or being agnostic about the surrounding environment wastes unnecessary attacking resources.

In a real-world data center, the average utilization is around 20% to 30%, as reported by Barroso et al. [25]. With such low utilization, the chance of tripping the circuit breaker by indiscriminately launching power attacks is extremely low. However, although the average utilization is low, data centers still encounter power outage threats under peak demands [116]. This indicates that the power consumption of physical servers fluctuates enormously with the changing workloads. To confirm this assumption, we conduct an experiment to monitor the whole-system power consumption (via the RAPL leakage channel in case study II of Section 3.2) of eight physical servers in a container cloud for one week. We present the result in Figure 3.2. We first average the power data with a 30-second interval and observe drastic power changes on both Day 2 and Day 5. Furthermore, we pick a high power consumption region in Day 2 and average the data at the interval of one second (which is a typical time window for generating a power spike). The peak power consumption could reach 1,199 Watts (W). In total, there was a 34.72% (899W $\sim$ 1,199W) power difference in this one-week range. We anticipate that the power consumption difference would be even larger if we could monitor it for a longer time period, such as on a holiday like Black Friday, when online shopping websites hosted on a cloud may incur a huge power surge.

For a synergistic power attack in a container cloud, instead of indiscriminately starting a power-intensive workload, the adversaries can monitor the whole-system power consumption through the RAPL channel [126] and learn the crests and troughs of the power consumption pattern at real time. Therefore, they can leverage the background power consumption (generated by benign workloads from other tenants on the same host) and superimpose their power attacks when the servers are

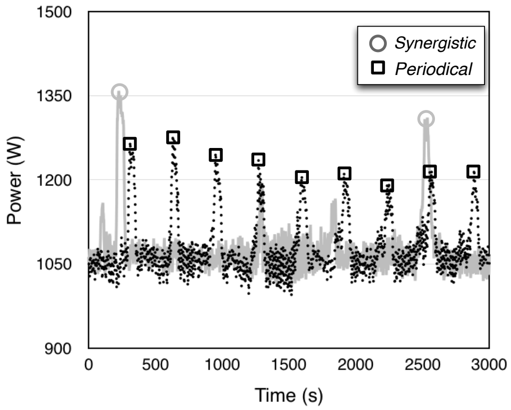


Figure 3.3: The Power Consumption of 8 Servers under Attack.

at their peak running time. This is similar to the phenomenon of *insider trading* in the financial market—the one with more insider information can always trade at the right time. The adversaries can boost their power spikes, by adding on already-high power consumption, to a higher level with the "insider" power consumption information leaked through the RAPL channel.

3.3.2 Reduction of Attack Costs

From the attackers' perspective, they always intend to maximize attack outcomes with the lowest costs. Running power-intensive workloads continuously could definitely catch all the crests of benign power consumption. However, it may not be practical for real-world attacks for several reasons. First, it is not stealthy. To launch a power attack, the attacker needs to run power-intensive workloads. Such behavior has obvious patterns and could be easily detected by cloud providers. Second, utilization-based billing models are now becoming more popular. More cloud services provide finer-grained prices based on CPU/memory utilization and the volume of network traffic. For instance, *Elastic Container* provides containers with CPU metering-based billing for customers [8]. *IBM Cloud* provides billing metrics for computing resources in the cloud [13]. Amazon EC2 [3] offers *Burstable Performance Instances* that could occasionally burst but do not fully run most of the time. The VMware OnDemand Pricing Calculator [14] even gives an estimate for different utilization levels. For example, it charges $2.87 per month for an instance with 16 VCPUs with an average of 1% utilization, and $167.25 for the same server with full utilization. Under these cloud billing models, *continuous* power attacks may finally lead to an expensive bill.

For synergistic power attacks, monitoring power consumption through RAPL has almost zero CPU utilization. To achieve the same effects (the height of power spikes), synergistic power attacks can significantly reduce the attack costs compared to continuous and periodic attacks. In Figure 3.3, we compare the attack effects of a synergistic power attack with a periodic attack (launching power attacks every 300 seconds). Synergistic power attacks can achieve a 1,359W power spike with only two trials in 3,000 seconds, whereas periodic attacks were launched nine times and could only reach 1,280W at most.

3.3.3 Attack Orchestration

Different from traditional power attacks, another unique characteristic of synergistic power attack is its attack orchestration. Assume an attacker is already controlling a number of container instances. If these containers scatter in different locations within a data center, their power additions on multiple physical servers put no pressure on power facilities. Existing power-capping mechanisms can tolerate multiple small power surges from different locations with no difficulty. The only way to launch a practical power attack is to aggregate all "ammunition" into adjacent locations and attack a single power supply simultaneously. Here we discuss in depth on the orchestration of attacking container instances.

As we mentioned in Section 3.2, by exploiting multiple leakage channels[7], attackers can aggregate multiple container instances into one physical server. Specifically in our experiment on $\mathbf{CC_1}$, we choose to use *timer_list* to verify the co-residence of multiple containers. The detailed verification method is explained in Section 3.2.3. We repeatedly create container instances and terminate instances that are not on the same physical server. By doing this, we succeed in deploying three containers on the same server with trivial effort. We run four copies of *Prime* [17] benchmark within each container to fully utilize the four allocated cores. The results are illustrated in Figure 3.4. As we can see, each container can contribute approximately 40W power. With three containers, an attacker can easily raise the power consumption to almost 230W, which is about 100W more than the average power consumption for a single server.

[7] Typically, if a channel is a strong co-residence indicator, leveraging this one channel only should be enough.

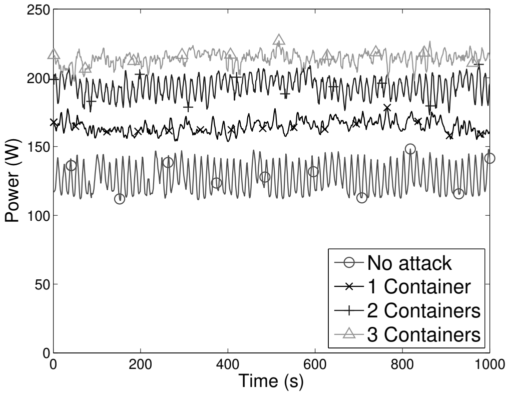


Figure 3.4: The Power Consumption of a Server under Attack.

We also find */proc/uptime* to be another interesting leakage channel. The *uptime* includes two data entries, the booting time of the physical server and the idle time of all cores. In our experiment, we find that some servers have similar booting times but different idle times. Typically servers in data centers do not reboot once being installed and turned on. A different idle time indicates that they are not the same physical server, while a similar booting time demonstrates that they have a high probability of being installed and turned on at the same time period. This is strong evidence that they might also be in close proximity and share the same circuit breaker. Attackers can exploit this channel to aggregate their attack container instances into adjacent physical servers. This greatly increases their chances of tripping circuit breakers to cause power outages.

3.4 Defense Approach

3.4.1 A Two-Stage Defense Mechanism

Intuitively, the solution should eliminate all the leakages so that no leaked information could be retrieved through those channels. We divide the defense mechanism into two stages to close the loopholes: (1) masking the channels and (2) enhancing the container's resource isolation model.

In the first stage, the system administrators can explicitly deny the read access to the channels within the container, e.g., through security policies in *AppArmor* or mounting the pseudo file "unreadable". This does not require any change to the kernel code (merging into the upstream Linux

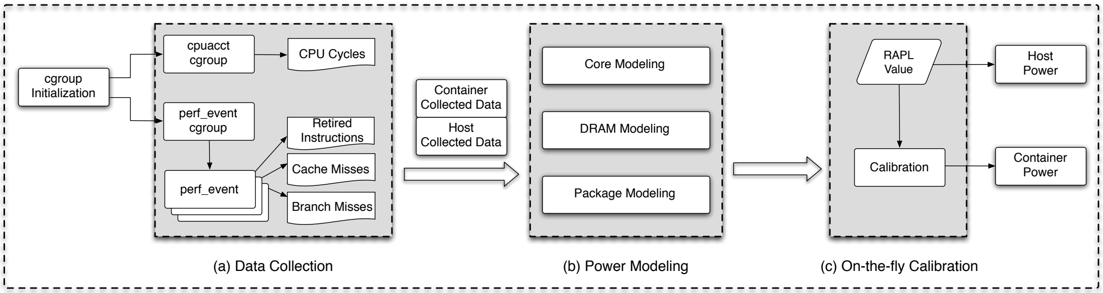


Figure 3.5: The Workflow of Power-Based Namespace.

kernel might take some time) and can immediately eliminate information leakages. This solution depends on whether legitimate applications running inside the container use these channels. If such information is orthogonal to the containerized applications, masking it will not have a negative impact on the container tenants. We have reported our results to *Docker* and all the cloud vendors listed in Table 3.1, and we have received active responses. We are working together with container cloud vendors to fix this information leakage problem and minimize the impact upon applications hosted in containers. This masking approach is a quick fix, but it may add restrictions for the functionality of containerized applications, which contradicts the container's concept of providing a generalized computation platform.

In the second stage, the defense approach involves fixing missing namespace context checks and virtualizing more system resources (i.e., the implementation of new namespaces) to enhance the container's isolation model. We first reported information disclosure bugs related to existing namespaces to Linux kernel maintainers, and they quickly released a new patch for one of the problems ([CVE-2017-5967]). For the other channels with no namespace isolation protection, we need to change the kernel code to enforce a finer-grained partition of system resources. Such an approach could involve non-trivial efforts since each channel needs to be fixed separately. Virtualizing a specific kernel component might affect multiple kernel subsystems. In addition, some system resources are not easy to be precisely partitioned to each container. However, we consider this to be a fundamental solution to the problem. In particular, to defend against synergistic power attacks, we design and implement a proof-of-concept power-based namespace in the Linux kernel to present the partitioned power usage to each container.

3.4.2 Power-based Namespace

We propose a power-based namespace to present per-container power usage through the unchanged RAPL interface to each container. Without leaking the system-wide power consumption information, attackers cannot infer the power state of the host, thus eliminating their chance of superimposing power-intensive workloads on benign power peaks. Moreover, with per-container power usage statistics at hand, we can dynamically throttle the computing power (or increase the usage fee) of containers that exceed their predefined power thresholds. It is possible for container cloud administrators to design a finer-grained billing model based on this power-based namespace.

There are three goals for our design. (1) Accuracy: as there is no hardware support for per-container power partitioning, our software-based power modeling needs to reflect the accurate power usage for each container. (2) Transparency: applications inside a container should be unaware of the power variations outside this namespace, and the interface of power subsystem should remain unchanged. (3) Efficiency: power partitioning should not incur non-trivial performance overhead in or out of containers.

We illustrate the workflow of our system in Figure 3.5. Our power-based namespace consists of three major components: *data collection*, *power modeling*, and *on-the-fly calibration*. We maintain the same Intel RAPL interface within containers, but change the implementation of handling read operations on energy usages. Once a read operation of energy usage is detected, the modified RAPL driver retrieves the per-container performance data (*data collection*), uses the retrieved data to model the energy usage (*power modeling*), and finally calibrates the modeled energy usage (*on-the-fly calibration*). We discuss each component in detail below.

3.4.2.1 Data collection

In order to model per-container power consumption, we need to obtain the fine-grained performance data for each container. Each container is associated with a *cpuacct* cgroup. A *cpuacct* cgroup accounts for the CPU cycles on a processor core for a container. The CPU cycles are accumulated. We only use CPU cycles to compute the rate of the cache miss rate and branch miss rate later. The Linux kernel also has a *perf_event* subsystem, which supports accounting for

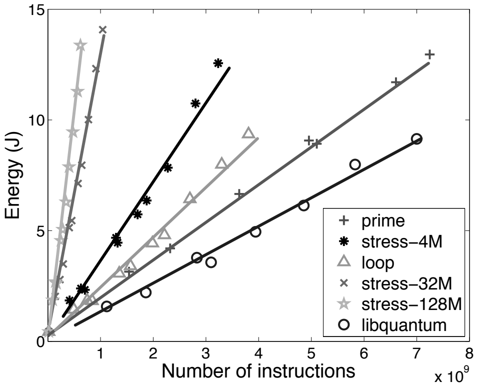


Figure 3.6: Power Modeling: the Relation Between Core Energy and the Number of Retired Instructions.

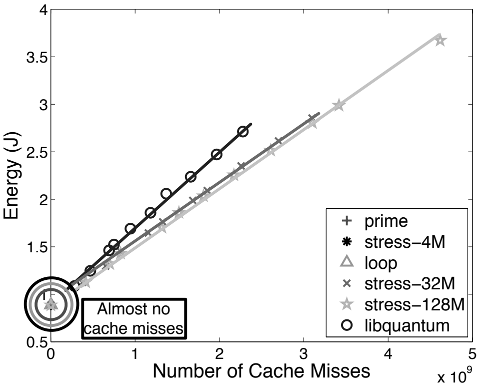


Figure 3.7: Power Modeling: the Relation Between DRAM Energy and the Number of Cache Misses.

different types of performance events. The granularity of performance accounting could be a single process or a group of processes (considered as a *perf_event* cgroup). By now, we only retrieve the data for retired instructions, cache misses, and branch misses (which are needed in the following *power modeling* component) for each *perf_event* cgroup. Our current implementation is extensible to collect more performance event types corresponding to the changes of *power modeling* in the future.

We monitor the performance events from the initialization of a power-based namespace and create multiple *perf_events*, each associated with a specific performance event type and a specific CPU core. Then we connect the *perf_cgroup* of this container with these *perf_events* to start accounting. In addition, we need to set the owner of all created *perf_events* as *TASK_TOMBSTONE*, indicating that such performance accounting is decoupled from any user process.

3.4.2.2 Power modeling

To implement a power-based namespace, we need to attribute the power consumption to each container. Instead of providing transient power consumption, RAPL offers accumulated energy usages for *package*, *core*, and *DRAM*, respectively. The power consumption can be calculated by measuring the energy consumption over a time unit window. Our power-based namespace also provides accumulative per-container energy data, in the same format as in the original RAPL interface.

We first attribute the power consumption for the *core*. Traditional power modeling leverages CPU utilization [83] to attribute the power consumption for cores. However, Xu et al. [123] demonstrated that the power consumption could vary significantly with the same CPU utilization. The underlying pipeline and data dependence could lead to CPU stalls and idling of function units. The actual numbers of retired instructions [72, 102] under the same CPU utilization are different. Figure 3.6 reveals the relation between retired instructions and energy. We test on four different benchmarks: the idle loop written in *C*, *prime*, *462.libquantum* in SPECCPU2006, and *stress* with different memory configurations. We run the benchmarks on a host and use *Perf* [16] to collect performance statistics data. We can see that for each benchmark, the energy consumption is almost strictly linear to the number of retired instructions. However, the gradients of fitted lines change correspondingly with application types. To make our model more accurate, we further include the cache miss rate [72] and branch miss rate to build a multi-degree polynomial model to fit the slope.

For the *DRAM*, we use the number of cache misses to profile the energy. Figure 3.7 presents the energy consumption for the same benchmarks with the same configurations in the *core* experiment. It clearly indicates that the number of cache misses is approximately linear to the *DRAM* energy. Based on this, we use the linear regression of cache misses to model the *DRAM* energy.

For the power consumption of *package*, we sum the values of *core*, *DRAM*, and an extra constant. The specific models are illustrated in Formula (3.2), where M represents the modeled energy; CM, BM, C indicate the number of cache misses, branch misses, and CPU cycles, respectively; and F is the function derived through multiple linear regressions to fit the slope. I is the number of retired instructions. α, β, γ, and λ are the constants derived from the experiment data in Figures 3.6 and 3.7.

$$\begin{aligned}
\mathrm{M_{core}} &= \mathrm{F}(\frac{\mathrm{CM}}{\mathrm{C}}, \frac{\mathrm{BM}}{\mathrm{C}}) \cdot \mathrm{I} + \alpha, \\
\mathrm{M_{dram}} &= \beta \cdot \mathrm{CM} + \gamma, \\
\mathrm{M_{package}} &= \mathrm{M_{core}} + \mathrm{M_{dram}} + \lambda.
\end{aligned} \tag{3.2}$$

Here we discuss the influence of floating point instructions for power modeling. While an individual floating point instruction might consume more energy than an integer operation, workloads with high ratios of floating point instructions might actually result in lower power consumption overall, since the functional units might be forced to be idle in different stages of the pipeline. It is necessary to take the micro-architecture into consideration to build a more refined model. We plan to pursue this direction in our future work. Furthermore, the choices of parameters α, β, γ are also affected by the architecture. Such a problem could be mitigated in the following calibration step.

3.4.2.3 On-the-fly calibration

Our system also models the energy data for the host and cross-validates it with the actual energy data acquired through RAPL. To minimize the error of modeling data, we use the following Formula (3.3) to calibrate the modeled energy data for each container. The $\mathrm{E}_{\mathrm{container}}$ represents the energy value returned to each container. This on-the-fly calibration is conducted for each read operation to the RAPL interface and can effectively reduce the number of errors in the previous step.

$$\mathrm{E}_{\mathrm{container}} = \frac{\mathrm{M}_{\mathrm{container}}}{\mathrm{M}_{\mathrm{host}}} \cdot \mathrm{E}_{\mathrm{RAPL}}. \tag{3.3}$$

3.5 Defense Evaluation

In this section, we evaluate our power-based namespace on a local machine in three aspects: accuracy, security, and performance. Our testbed is equipped with Intel i7-6700 3.40GHz CPU with 8 cores, 16GB of RAM, and running Ubuntu Linux 16.04 with kernel version 4.7.0.

3.5.1 Accuracy

We use the SPECCPU2006 benchmark to measure the accuracy of the power modeling. We compare the modeled power usage with the ground truth obtained through RAPL. The power consumption is equal to the energy consumption per second. Due to the restriction of the security policy of the *Docker* container, we select a subset of SPECCPU2006 benchmarks that are feasi-

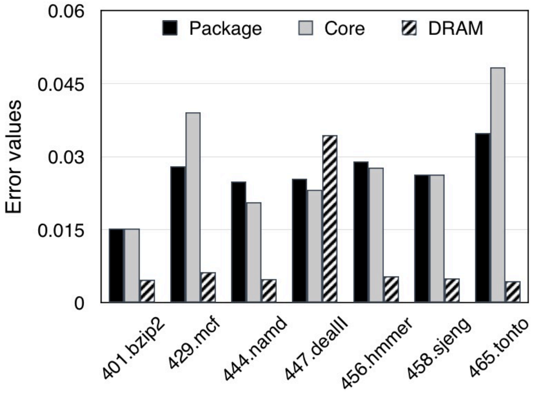


Figure 3.8: The Accuracy of Our Energy Modeling Approach to Estimate the Active Power for the Container from Aggregate Event Usage and RAPL.

ble to run inside the container and have no overlap with the benchmarks used for power modeling. The error ξ is defined as follows:

$$\xi = \frac{|(\mathrm{E}_{\mathrm{RAPL}} - \square_{\mathrm{diff}}) - \mathrm{M}_{\mathrm{container}}|}{\mathrm{E}_{\mathrm{RAPL}} - \square_{\mathrm{diff}}}, \tag{3.4}$$

where $\mathrm{E}_{\mathrm{RAPL}}$ is the power usage read from RAPL on the host, and $\mathrm{M}_{\mathrm{container}}$ is the modeled power usage for the same workload read within the container. Note that both the host and container consume power at an idle state with trivial differences. We use a constant $\square_{\mathrm{diff}}$ as the modifier reflecting the difference in power consumption at an idle state for a host and a container. The results, illustrated in Figure 3.8, show that our power modeling is accurate as the error values of all the tested benchmarks are lower than 0.05.

3.5.2 Security

We also evaluate our system from the security perspective. With the power-based namespace enabled, the container should only retrieve the power consumed within the container and be unaware of the host's power status. We launch two containers in our testbed for comparison. We run the SPECCPU2006 benchmark in one container and leave the other one idle. We record the power usage per second of both containers and the host. We show the results of *401.bzip2* in Figure 3.9. All other benchmarks exhibit similar patterns.

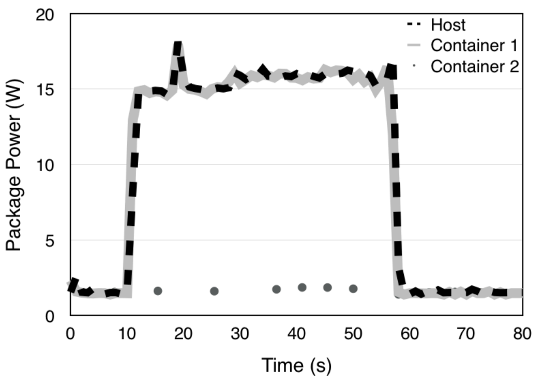


Figure 3.9: Transparency: A Malicious Container (Container 2) is Unaware of the Power Condition for the Host.

When both containers have no workload, their power consumption is at the same level as that of the host, e.g., from 0s to 10s. Once container 1 starts a workload at 10s, we can find that the power consumption of container 1 and the host surges simultaneously. From 10s to 60s, container 1 and the host have a similar power usage pattern, whereas container 2 is still at a low power consumption level. Container 2 is unaware of the power fluctuation on the whole system because of the isolation enforced by the power-based namespace. This indicates that our system is effective for isolating and partitioning power consumption for multiple containers, and thus it can neutralize the *synergistic power attacks*.

3.5.3 Performance

We use UnixBench to compare the performance overhead before and after enabling our system. Table 3.3 lists all results.

As the results show, CPU benchmarks such as *Dhrystone* (testing integer and string operations) and *Whetstone* (testing float point arithmetic performance) incur negligible overhead. Other benchmarks like *shell scripts*, *pipe throughput*, and *system call* also trigger little overhead. The *pipe-based context switching* does incur a 61.53% overhead in the case of one parallel copy, but it decreases to 1.63% for 8 parallel copies. We anticipate that inter-cgroup context switching involves enabling/disabling the performance event monitor, whereas intra-cgroup context switching does not involve any such overhead. This could explain why 8 parallel copies can maintain a similar

Table 3.3: Performance Results of Unix Benchmarks

	1 Parallel Copy			8 Parallel Copies		
Benchmarks	**Original**	**Modified**	**Overhead**	**Original**	**Modified**	**Overhead**
Dhrystone 2 using register variables	3,788.9	3,759.2	0.78%	19,132.9	19,149.2	0.08%
Double-Precision Whetstone	926.8	918.0	0.94%	6,630.7	6,620.6	0.15%
Execl Throughput	290.9	271.9	6.53%	7,975.2	7,298.1	8.49%
File Copy 1024 bufsize 2000 maxblocks	3,495.1	3,469.3	0.73%	3,104.9	2,659.7	14.33%
File Copy 256 bufsize 500 maxblocks	2,208.5	2,175.1	0.04%	1,982.9	1,622.2	18.19%
File Copy 4096 bufsize 8000 maxblocks	5,695.1	5,829.9	-2.34%	6,641.3	5,822.7	12.32%
Pipe Throughput	1,899.4	1,878.4	1.1%	9,507.2	9,491.1	0.16%
Pipe-based Context Switching	653.0	251.2	61.53%	5,266.7	5,180.7	1.63%
Process Creation	1416.5	1289.7	8.95%	6618.5	6063.8	8.38%
Shell Scripts (1 concurrent)	3,660.4	3,548.0	3.07%	16,909.7	16,404.2	2.98%
Shell Scripts (8 concurrent)	11,621.0	11,249.1	3.2%	15,721.1	15,589.2	0.83%
System Call Overhead	1,226.6	1,212.2	1.17%	5,689.4	5,648.1	0.72%
System Benchmarks Index Score	2,000.8	1,807.4	9.66%	7,239.8	6,813.5	7.03%

performance level with the power-based namespace disabled. In addition, context switching only contributes to a very small portion of the whole-system performance overhead, so there is trivial impact for the normal use. As demonstrated in the last row of Table 3.3, the overall performance overheads for the UnixBench are 9.66% for one parallel copy and 7.03% for 8 parallel copies, respectively. Our system's performance depends heavily on the implementation of *perf_event* cgroup and could improve with the advancement of a performance monitoring subsystem.

3.6 Discussion

3.6.1 Synergistic Power Attacks without the RAPL Channel

We also notice that servers in some container clouds are not equipped with RAPL or other similar embedded power meters. Those servers might still face power attacks. Without power-capping tools like RAPL, those servers might be vulnerable to host-level power attacks on a single machine. In addition, if power data is not directly available, advanced attackers will try to approximate the power status based on the resource utilization information, such as the CPU and memory utilization, which is still available in the identified information leakages. It would be better to make system-wide performance statistics unavailable to container tenants.

3.6.2 Complete Container Implementation

The root cause for information leakage and synergistic power attack is the incomplete implementation of the isolation mechanisms in the Linux kernel. It would be better to introduce more security features, such as implementing more namespaces and control groups. However, some system resources are still difficult to be partitioned, e.g., interrupts, scheduling information, and temperature. People also argue that the complete container implementation is no different from a virtual machine, and loses all the container's advantages. It is a trade-off for containers to deal with. The question of how to balance security, performance, and usability in container clouds needs further investigation.

3.7 Related Work

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

3.7.1 Performance and Security Research on Containers

Since containers have recently become popular, researchers are curious about the performance comparison between containers and hardware virtualization. Felter et al. compared the performance of *Docker* and *KVM* by using a set of benchmarks covering CPU, memory, storage, and networking resources [43]. Their results show that *Docker* can achieve equal or better performance than *KVM* in all cases. Spoiala et al. [106] used the *Kurento Media Server* to compare the performance of *WebRTC* servers on both *Docker* and *KVM*. They also demonstrated that *Docker* outperforms *KVM* and could support real-time applications. Morabito et al. [85] compared the performance between traditional hypervisor and OS-level virtualization with respect to computing, storage, memory, and networks. They conducted experiments on *Docker*, *LXC*, and *KVM* and observed that Disk I/O is still the bottleneck of the *KVM* hypervisor. All of these works demonstrate that container-based OS-level virtualization can achieve a higher performance than hardware virtualization. Besides performance, the security of a container cloud is always an important research area. Gupta [56] gave a brief overview of *Docker* security. Bui [29] also performed an analysis on *Docker* containers, including the isolation problem and corresponding kernel security mechanisms. They claimed that *Docker* containers are fairly secure with default configurations.

Grattafiori et al. [53] summarized a variety of potential vulnerabilities of containers. They also mentioned several channels in the memory-based pseudo file systems. Previous research efforts on the performance and security of containers encourage us to investigate more on how containers can achieve the same security guarantees as hardware virtualization, but with trivial performance trade-offs. We are among the first to systematically identify the information leakage problem in containers and investigate potential container-based power attack threats built upon these leakage channels.

3.7.2 Cloud Security and Side/Covert Channel Attacks

Cloud security has received much attention from both academia and industry. Co-residence detection in the cloud settings is the most closely related research topic to our work. Co-residence detection was first proposed by Ristenpart et al. [98]. They demonstrated that an attacker can place a malicious VM co-resident with a target VM on the same sever and then launch side-channel and covert-channel attacks. Two previous works [112, 122] show that it is still practical to achieve co-residence in existing mainstream cloud services. To verify co-residence on the same physical server, attackers typically leverage side channels or covert channels [124, 125], e.g., one widely adopted approach is to use cache-based covert channels [65, 111, 121]. Multiple instances locating on the same package share the last-level caches. By using some dedicated operations, such as *cflush* [127], attackers can detect co-residence by measuring the time of cache accessing. Liu et al. [76] demonstrated that l3 cache side-channel attacks are practical for cross-core and cross-VM attacks. Zhang et al. conducted real side-channel attacks on the cloud [130, 131] and proposed several defense mechanisms to mitigate those attacks [120, 129, 132]. In particular, they demonstrated that cross-tenant side-channel attacks can be successfully conducted in PaaS with co-resident servers [131]. Besides the cache-based channel, memory bus [118] and memory deduplication [119] have also proved to be effective for covert-channel construction. Different from existing research efforts on side/covert channels, we discover a system-wide information leakage in the container cloud settings and design a new methodology to quantitatively assess the capacity of leakage channels for co-residence detection. In addition, compared to the research on minimizing the kernel attack surface for VMs [54], we proposed a two-stage defense mechanism to minimize the space for information leakages and power attacks on container clouds.

System status information, such as core temperature and system power consumption, have also been used to build side/covert channels. Thiele et al. [26, 82] proposed a thermal covert channel based on the temperature of each core and tested the capacity in a local testbed. Power consumption could also be abused to break AES [67]. In our work, we do not use the power consumption data as a covert channel to transfer information. Instead, we demonstrate that adversaries may leverage the host power consumption leakage to launch more advanced power attacks.

3.7.3 Power Modeling

When hardware-based power meter is absent, power modeling is the approach to approximating power consumption. Russell et al. [99] and Chakrabarti et al. [31] proposed instruction-level power modeling. Their works indicate that the number of branches affects power consumption. There are several works of approximating power consumption for VMs. Both works [63, 72] demonstrate that VM-level power consumption can be estimated by CPU utilization and last-level cache miss. Mobius et al. [83] broke the power consumption of VM into CPU, cache, memory, and disk. BITWATTS [35] modeled the power consumption at a finer-grained process level. Shen et al. [102] proposed a power container to account for energy consumption of requests in multi-core systems. Our defense against the synergistic power attack is mainly inspired by the power modeling approach for VMs. We propose a new power partitioning technique to approximate the per-container power consumption and reuse the RAPL interface, thus addressing the RAPL data leakage problem in the container settings.

3.8 Chapter Summary

Container cloud services have become popular for providing lightweight OS-level virtual hosting environments. The emergence of container technology profoundly changes the eco-system of developing and deploying distributed applications in the cloud. However, due to the incomplete implementation of system resource partitioning mechanisms in the Linux kernel, there still exist some security concerns for multiple container tenants sharing the same kernel. In this chapter, we first present a systematic approach to discovering information leakage channels that may expose host information to containers. By exploiting such leaked host information, malicious container tenants can launch a new type of power attack that may potentially jeopardize the dependability of power systems in the data centers. Additionally, we discuss the root causes of these information leakages and propose a two-stage defense mechanism. Our evaluation demonstrates that the proposed solution is effective and incurs trivial performance overhead.

Chapter 4

Sudden Death of Live Migration and Its Security Implications

4.1 Introduction

In this chapter, 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. We emphasize that the root cause of such a vulnerability is rooted in the fundamental design of *post-copy* rather than the actual implementation. 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 [30] 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 environments. We implement a prototype by modifying QEMU and the Linux kernel.

The rest of this chapter is organized as follows. Section 4.2 introduces the background of VM live migration and describes the reset threat to TCP connections. Section 4.3 presents the vulnerability in VM live migration process, and details the method to reset the migration connection within heavy networking traffic. Section 4.4 shows the evaluation of our attack on VM live migration from different aspects. Section 4.5 presents two general defense mechanisms. Section 4.6 surveys related work, and finally Section 4.7 concludes.

4.2 Background

4.2.1 VM Live Migration

Clark et al. [34] 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. 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* [34] and *post-copy* [58, 59], are widely deployed.

4.2.1.1 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. Once the iterative copying is no longer beneficial, 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.

4.2.1.2 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.

4.2.2 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.

4.2.2.1 Blind Reset Attack

RFC 5961 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 [114]: 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, 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 $1/2^{32}$.

4.2.2.2 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 [40, 41, 50, 93, 94]. Besides, a more recent work [30] 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. 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 [30] 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 [95] 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.

4.3 Threats in VM Live Migration

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 environment. Hence, it is generally believed that the existing approach is secure enough, and many protection mechanisms are not taken seriously. In this section, we argue that the TCP-RST attack could lead to devastating and unrecoverable consequences for today's *post-copy*-based VM live migrations. The termination of a TCP connection would cause

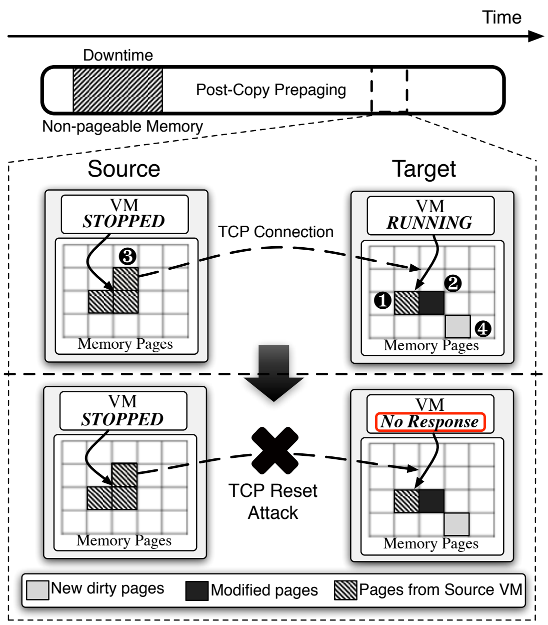


Figure 4.1: Post-copy VM Live Migration.

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.

4.3.1 Vulnerability Overview

4.3.1.1 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. Figure 4.1 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 the RST attack succeeds in terminating the TCP connection at this point, in current hypervisor implementation, the VM on the target server would be irresponsive immediately. 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 4.1). 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.

4.3.1.2 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.

4.3.2 Attacking VM Live Migration

4.3.2.1 Threat Model

The attacker sets it target on destroying VM live migration in the cloud environment by resetting the TCP connection. We assume that the attacker knows several IP addresses of the servers in the target cloud. 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 4.2. The attacker can

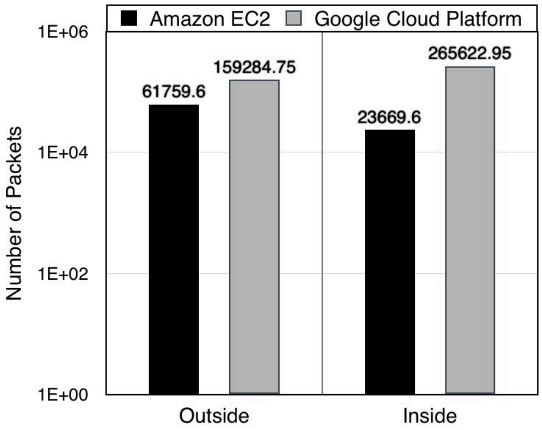


Figure 4.2: Number of RST packets per second in clouds.

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 connection. 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.

4.3.2.2 TCP RST Packets in Clouds

The ability to send crafted RST packets is the pre-requisite for launching TCP reset attacks. We 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 TCP RST 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 4.2 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 [39, 71], since the security vulnerabilities of VMMs have been reported on a regular basis due to the complexity of VMMs.

4.3.2.3 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. As mentioned in Section 4.2, 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 completely 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 [30] could be utilized to explore the rest of the tuples.

4.3.2.4 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 [33, 50, 51, 93, 94]. 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 [30], as in the case of $\mathbb{I}$ illustrated in Figure 4.3. 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

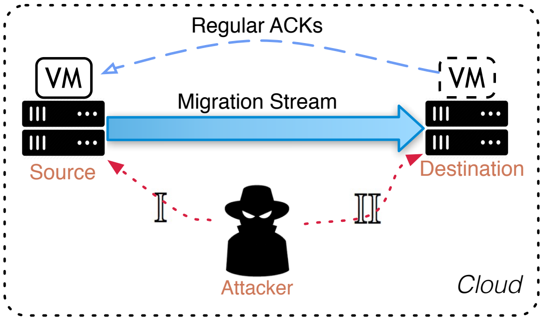
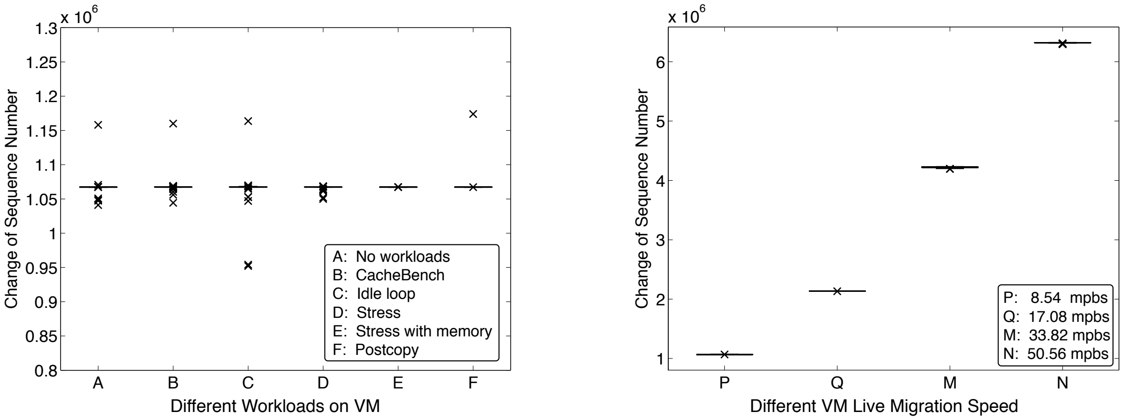


Figure 4.3: Attack Scenarios.

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 packets in that block.

Side-channel reset attacks on the destination. Attacking the target server (as in the case of $\mathbb{II}$ in Figure 4.3) is much more difficult than the source server case. The heavy traffic makes the sequence number vary rapidly, leaving the previous method [30] 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 4.4(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



(a) Migration Methods and Workloads.

(b) Migration Speed.

Figure 4.4: Factors Affecting the Change of Sequence No. of VM Live Migration.

to 17.08 Mbps, 33.82 Mbps, and 50.56 Mbps. The results in Figure 4.4(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, 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$\sim$10. This procedure, *step profiling*, could obtain an approximate value of the speed by dividing the increased size by time consumption.

Lines 11$\sim$24 present the next procedure, *micro tuning*. This step is the most critical one, and

Algorithm 1 Identify and stalk the correct sequence number range

```
 1: procedure Sequence_Number_Stalking($s'_l$, $s'_u$)
 2:     $step_o \leftarrow$ 5 × ($s'_u$ - $s'_l$);
 3:     $s_{u,l} \leftarrow s'_{u,l} + step_o$;
 4:     range ← ($s_u$ - $s_l$);
 5: Step profiling:
 6:     while $range\_hit()$ do
 7:         $count \leftarrow count + 1$;
 8:         wait(1);
 9:     step ← $step_o$/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 ← $step_o$/(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.
```

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 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.

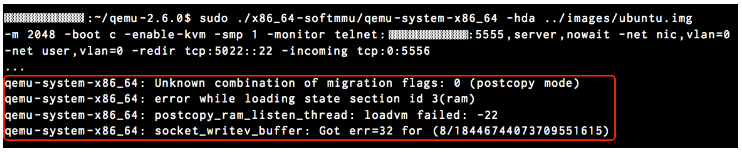
4.4 Evaluations

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.

4.4.1 Attacking Consequences

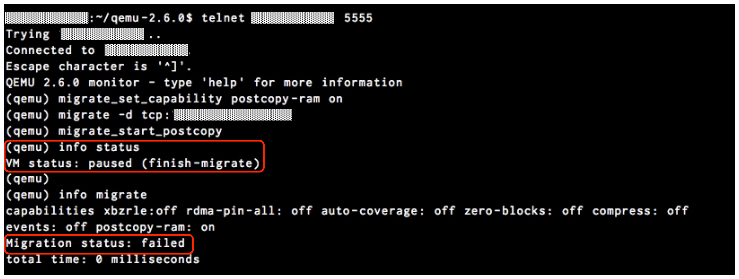
We first demonstrate that the TCP reset attack can cause the devastating consequences we expect. Our experiments are conducted on QEMU version 2.6. We also test QEMU 2.8 and 2.9, which give similar results. We use two Ubuntu servers with Linux kernel 4.4 with independent, public IP addresses.

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 4.5(a). In some cases, the destination VM would not be crashed, but totally unresponsive. Figure 4.5(b) shows the status of the source



```
[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).

Figure 4.5: Attack on Post-copy Live Migration.

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.

4.4.2 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.

4.4.2.1 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

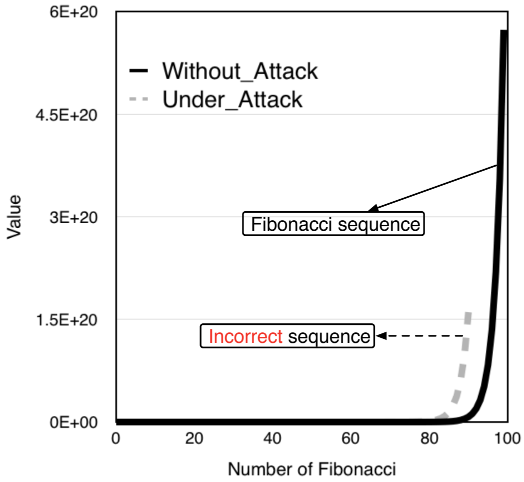


Figure 4.6: Memory Inconsistence:
Incorrect Fibonacci Number.

F_{n-1}. Based on a counter (used as the index) maintained in the toy server, the toy 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.

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 4.6, 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 on the VM, 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

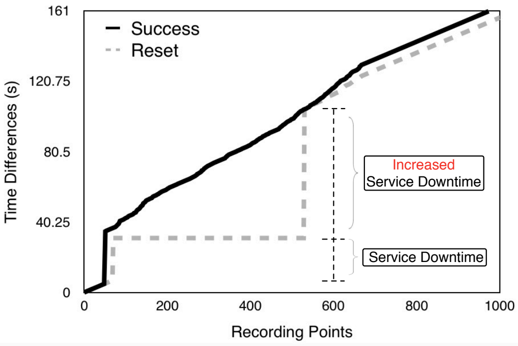


Figure 4.7: Downtime Increases.

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 Fibonacci 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.

4.4.2.2 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 4.7 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

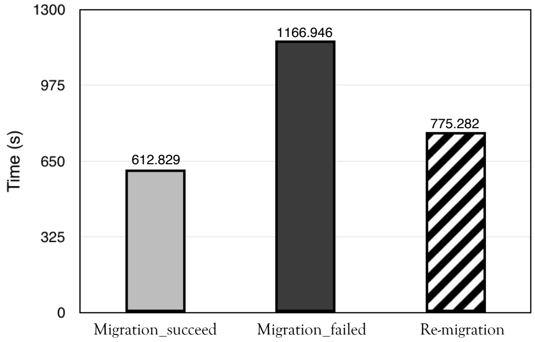


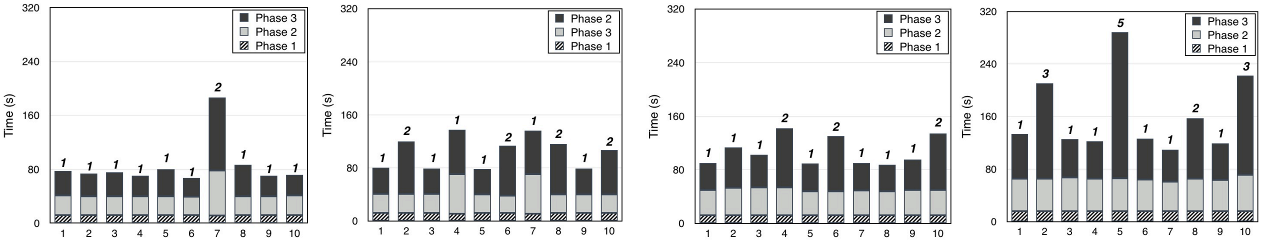
Figure 4.8: Performance Degradation.

the re-migration starts immediately in this experiment. Otherwise, the increase of downtime could be much more serious.

4.4.2.3 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 4.8. 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.



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

Figure 4.9: Attacks on VM Live Migration with different throughput.

4.4.3 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 [30], 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 4.9. 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 4.3, 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 4.9(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.

4.5 Defense

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. 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. 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 to 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.

4.6 Related Work

4.6.1 VM Live Migration

VM live migration with *pre-copy* was first proposed in 2005 [34]. Then, several new approaches based on *pre-copy* were proposed to improve performance, including the Fast Transparent Migration [88], Delta Compression Techniques [109], Sandpiper [115], and MiG [97]. Hines [58, 59] presented the design, implementation, and evaluation of post-copy VM live migration in 2009. Ali [81] designed XvMotion, allowing VM migration over long distances. Instead of improving the performance, our work aims to avoid performance downgrades by fixing potential vulnerabilities.

Perez et al. [92] 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. [70] showed that the RTT (round trip time) of ICMP packets could be utilized by outside attackers to detect the migration process. Fiebig [44] 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 (network time protocol). 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 [90, 108, 128]. Oberheide et al. [89] 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.

4.6.2 TCP Attacks

The security of TCP, especially in off-path attacks, always receives much attention. Gilad et al. [50] 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 [33,93,94] abuse OS packet counters to determine whether the guessed sequence number is correct. Gilad et al. [51] 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 chapter. In addition to cracking the sequence number, there are also several other security issues for TCP. Alexander et al. [24] presented a novel technique for estimating the RTT latency between two off-path hosts. Ensafi et al. [41] showed that it is even possible for an attacker to port scan a network from outside the firewall. Abramov et al. [22] 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.

4.6.3 Denial-of-Service Attacks on Clouds

DoS attacks on clouds are also closely related to our work. For instance, power attacks [123] can cause power outages in data centers, and lead to forced shutdowns for servers on the same rack or on the same power distribution unit. For those DoS attacks [49], 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 [121], memory bus [118], networking [98], and system process statistics [112] 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. [130, 131] showed several approaches for extracting private keys and collecting potentially sensitive application data on co-resident instances. 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.

4.7 Chapter Summary

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 chapter, 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 VM live migration approaches. In addition, the termination of 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. Finally, we conduct a large-scale measurement to investigate the potential TCP reset threats in two commercial clouds. Our results show that attackers can send a large amount of TCP RST packets per second, and many instances are still vulnerable to the TCP reset threats.

Chapter 5

Conclusion and Future Work

This dissertation presents a study on the potential security vulnerabilities in today's clouds and data centers from low-level infrastructures like cooling systems and power supply facilities to high-level isolation techniques including virtual machines and Linux containers. In particular, we first reveal a new security vulnerability in a cloud's cooling infrastructure, named as reduced cooling redundancy that could be abused to cause cooling failures in a data center. Then, we present a systematic study on the information leakage problem and its security implications in public commercial multi-tenancy container clouds. Finally, we investigate security threats that exist in the process of VM live migration and present detailed techniques to cause memory inconsistency for the post-copy based approach. The three major contributions of this dissertation are summarized as follows.

- While data centers have widely adopted aggressive energy saving policies such as raising supply air temperature of cooling systems and power over-subscription, we reveal that those policies lead to reduced cooling redundancies that can be exploited to lower servers' reliability, raise cooling cost, and even cause massive power outages. We conduct extensive physical measurement studies to understand the thermal characteristics of a single server, and propose various attack vectors on both virtualized and non-virtualized environments. We further perform data center level simulations based on CFD analysis for the feasibility of launching thermal attacks on the rack and data center levels. While this thermal vulnerability cannot be completely fixed without increasing cooling redundancies, we propose a high-level dynamic thermal-aware load balancing to mitigate the threat.

- Relying on multiple building blocks (e.g., *namespace*) in the Linux kernel, containers can achieve near native performance with resource isolation. However, the incomplete or buggy implementation might lead to the expose of system-wide sensitive information among containers. We present the first systematic study on the information leakage problem in a container: we uncover the existence of more than thirty leakage channels in five commercial multi-tenancy container clouds. We demonstrate that those in-container interfaces allow attackers to retrieve the information of host OSes and co-located containers, verify co-residence, and launch advanced cloud-based attacks such as synergistic power attack. Finally, we discuss the root cause of the information leakage in depth, and propose a two-stage defense mechanism, which includes a power-based namespace implemented in the Linux kernel. The experimental results show that our solution is effective and incurs trivial performance overhead.

- Acting as a regular tool for system and resource management in cloud services, VM live migration must be fully secured to ensure minimal disruptions. We expose that the fundamental of VM live migration might be insecure: the entire procedure relies upon reliable TCP connectivity to transfer all VM states, which is actually vulnerable to malicious attacks. We demonstrate that attackers can cause devastating consequences to VMs in the live migration by intentionally aborting the underlying TCP connection. Those include unrecoverable memory inconsistency for a post-copy-based VM live migration, a significant increase in downtime and performance degradation of the running VM. In particular, we present the algorithms in detail to mount TCP-reset attacks on VM live migration by exploiting multiple vulnerabilities. The experimental results on QEMU+KVM environments show that the state-of-art live migration techniques (e.g., post-copy) suffer such a vulnerability. To address this problem, we propose and implement effective defense mechanisms to secure the process by modifying the Linux kernel.

In our future work, we will keep focusing on defense against the emerging threats in the cloud and data center environments. We will conduct a deep study on advanced exploitation of the information leakage channels in containers, whose damage might be more than that incurred by DDoS attacks: they could be exploited as side channels to infer private information of running applications of other tenants. Meanwhile, attackers can utilize them as covert channels to deliver illegal commands. From the defense perspective, we will systematically explore the information leakage through a whole-kernel program analysis. This analysis will include but not limit to memory-based pseudo file systems, system calls, input/output control, and network I/O. Moreover, we will attempt to propose proactive thermal control systems based on real-time server temperature prediction to defend against thermal attacks. Through the combination of (1) offline physical server placement adjustment, (2) online prediction-based defense, and (3) the exploitation of energy storage devices, our proposed proactive systems can make data centers more resilient to emerging threats like power and thermal attacks.

Bibliography

[1] Amazon EC2 Dedicated Hosts Pricing. http://aws.amazon.com/ec2/dedicated-hosts/pricing/.

[2] Amazon outage cost S&P 500 companies $150M. https://www.axios.com/amazon-outage-cost-150m-for-s-p-500-companies-2295532812.html.

[3] Burstable Performance Instances. https://aws.amazon.com/ec2/instance-types/#burst.

[4] Computational Fluid Dynamics: ANSYS CFX and FLUENT CFD Software. http://www.caeai.com/cfd-software.php.

[5] Containers Not Virtual Machines Are the Future Cloud. http://www.linuxjournal.com/content/containers.

[6] Delta Air Lines says the total bill for its devastating computer outage will come to $150 million. http://money.cnn.com/2016/09/07/technology/delta-computer-outage-cost/index.html.

[7] Efficiency: How we do it. https://www.google.com/about/datacenters/efficiency/internal/temperature.

[8] ElasticHosts: Linux container virtualisation allows us to beat AWS on price. http://www.computerworlduk.com/news/it-leadership/elastichosts-linux-container-virtualisation-allows-us-beat-aws-on-price-3510973/.

[9] Electromigration. https://en.wikipedia.org/wiki/Electromigration.

[10] Enterprises Still Failing to Cut Data Center Power, Cooling Costs. http://www.eweek.com/it-management/enterprises-still-failing-to-cut-data-center-power-cooling-costs.html.

[11] Go!Temp. http://www.vernier.com/products/sensors/temperature-sensors/go-temp/.

[12] Heatwave, Cooling Failure Bring iiNet Data Center Down in Perth. http://www.datacenterknowledge.com/archives/2015/01/06/heatwave-cooling-failure-bring-iinet-data-center-down-in-perth/.

[13] IBM Cloud metering and billing. https://www.ibm.com/developerworks/c-loud/library/cl-cloudmetering/.

[14] OnDemand Pricing Calculator. http://vcloud.vmware.com/service-offering/pricing-calculator/on-demand.

[15] Overheating brings down Microsoft data center. http://www.datacenterdynamics.com/content-tracks/power-cooling/overheating-brings-down-microsoft-data-center/74543.fullarticle.

[16] Perf Wiki. https://perf.wiki.kernel.org/index.php/Main_Page.

[17] Prime95 Version 28.9. http://www.mersenne.org/download/ .

[18] Travel nightmare for fliers after power outage grounds Delta. http://money.cnn.com/2016/08/08/news/companies/delta-system-outage-flights/.

[19] Truck Crash Knocks Rackspace Offline. http://www.datacenterknowledge.com/archives/2007/11/13/truck-crash-knocks-rackspace-offline/.

[20] Univesity of Pennsylvania Data Center Overheats. http://www.datacenterknowledge.com/archives/2010/01/20/u-of-penn-data-center-overheats/.

[21] Wikipedia's Data Center Overheats. http://www.datacenterknowledge.com/archives/2010/03/25/downtime-for-wikipedia-as-data-center-overheats/.

[22] Raz Abramov and Amir Herzberg. TCP Ack storm DoS attacks. *Computers & Security*, 2013.

[23] Faraz Ahmad and TN Vijaykumar. Joint Optimization of Idle and Cooling Power in Data Centers While Maintaining Response Time. In *ACM Sigplan Notices*, 2010.

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

[25] Luiz Barroso and Urs Hölzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 2007.

[26] Davide Bartolini, Philipp Miedl, and Lothar Thiele. On the Capacity of Thermal Covert Channels in Multicores. In *ACM EuroSys*, 2016.

[27] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *IFIP/IEEE IM*, 2007.

[28] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. In *ACM VEE*, 2007.

[29] Thanh Bui. Analysis of Docker Security. *arXiv preprint arXiv:1501.02967*, 2015.

[30] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V Krishnamurthy, and Lisa M Marvel. Off-Path TCP Exploits: Global Rate Limit Considered Dangerous. In *USENIX Security*, 2016.

[31] Chaitali Chakrabarti and Dinesh Gaitonde. Instruction Level Power Model of Microcontrollers. In *IEEE ISCAS*, 1999.

[32] Vikas Chandra and Robert Aitken. Impact of Technology and Voltage Scaling on the Soft Error Susceptibility in Nanoscale CMOS. In *IEEE DFTVS*, 2008.

[33] Qi Alfred Chen, Zhiyun Qian, Yunhan Jack Jia, Yuru Shao, and Zhuoqing Morley Mao. Static Detection of Packet Injection Vulnerabilities: A Case for Identifying Attacker-controlled Implicit Information Leaks. In *ACM CCS*, 2015.

[34] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *USENIX NSDI*, 2005.

[35] Maxime Colmant, Mascha Kurpicz, Pascal Felber, Loïc Huertas, Romain Rouvoy, and Anita Sobe. Process-level Power Estimation in VM-based Systems. In *ACM EuroSys*, 2015.

[36] Ayse Coskun, Richard Strong, Dean Tullsen, and Tajana Rosing. Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors. In *ACM SIGMETRICS*, 2009.

[37] Tathagata Das, Pradeep Padala, Venkata N Padmanabhan, Ramachandran Ramjee, and Kang G Shin. LiteGreen: Saving Energy in Networked Desktops Using Virtualization. In *USENIX ATC*, 2010.

[38] Nosayba El-Sayed, Ioan Stefanovici, George Amvrosiadis, Andy Hwang, and Bianca Schroeder. Temperature Management in Data Centers: Why Some (Might) Like it Hot. *ACM SIGMETRICS*, 2012.

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

[40] Roya Ensafi, Jeffrey Knockel, Geoffrey Alexander, and Jedidiah R Crandall. Detecting Intentional Packet Drops on the Internet via TCP/IP Side Channels. In *International Conference on Passive and Active Network Measurement*, 2014.

[41] Roya Ensafi, Jong Chun Park, Deepak Kapur, and Jedidiah R Crandall. Idle Port Scanning and Non-interference Analysis of Network Protocol Stacks Using Model Checking. In *USENIX Security*, 2010.

[42] Xiaobo Fan, Wolf Weber, and Luiz Barroso. Power Provisioning for a Warehouse-Sized Computer. In *IEEE ISCA*, 2007.

[43] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An Updated Performance Comparison of Virtual Machines and Linux Containers. In *IEEE ISPASS*, 2015.

[44] Sebastian Fiebig, Melanie Siebenhaar, Christian Gottron, and Ralf Steinmetz. Detecting VM Live Migration using a Hybrid External Approach. In *CLOSER*, 2013.

[45] Xing Fu, Xiaorui Wang, and Charles Lefurgy. How Much Power Oversubscription is Safe and Allowed in Data Centers. In *ACM ICAC*, 2011.

[46] Karthik Ganesan, Jungho Jo, W Bircher, Dimitris Kaseridis, Zhibin Yu, and Lizy John. System-Level Max Power (SYMPO): A Systematic Approach for Escalating System-Level Power Consumption using Synthetic Benchmarks. In *ACM PACT*, 2010.

[47] Karthik Ganesan and Lizy John. MAximum Multicore POwer (MAMPO): An Automatic Multithreaded Synthetic Power Virus Generation Framework for Multicore Systems. In *ACM SC*, 2011.

[48] Xing Gao, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds. In *IEEE DSN*, 2017.

[49] Xing Gao, Zhang Xu, Haining Wang, Li Li, and Xiaorui Wang. Reduced Cooling Redundancy: A New Security Vulnerability in a Hot Data Center. In *NDSS*, 2018.

[50] Yossi Gilad and Amir Herzberg. Off-Path Attacking the Web. In *USENIX WOOT*, 2012.

[51] Yossi Gilad and Amir Herzberg. Spying in the Dark: TCP and Tor Traffic Analysis. In *PETS*, 2012.

[52] Íñigo Goiri, Thu Nguyen, Ricardo Bianchini, and Íñigo Presa. CoolAir: Temperature-and Variation-Aware Management for Free-Cooled Datacenters. In *ACM ASPLOS*, 2015.

[53] Aaron Grattafiori. NCC Group Whitepaper: Understanding and Hardening Linux Containers, 2016.

[54] Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. FACE-CHANGE: Application-Driven Dynamic Kernel View Switching in a Virtual Machine. In *IEEE/IFIP DSN*, 2014.

[55] Part Guide. Intel® 64 and IA-32 Architectures Software Developer's Manual, 2011.

[56] Udit Gupta. Comparison between security majors in virtual machine and linux containers. *arXiv preprint arXiv:1507.07816*, 2015.

[57] Jahangir Hasan, Ankit Jalote, TN Vijaykumar, and Carla Brodley. Heat Stroke: Power-Density-Based Denial of Service in SMT. In *IEEE HPCA*, 2005.

[58] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. Post-Copy Live Migration of Virtual Machines. *ACM SIGOPS Operating Systems Review*, 2009.

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

[60] Wei Huang, Malcolm Ware, John Carter, Elmootazbellah Elnozahy, Hendrik Hamann, Tom Keller, Arles Lefurgy, Jian Li, Karthick Rajamani, and Juan Rubio. Tapo: Thermal-Aware Power Optimization Techniques for Servers and Data Centers. In *IEEE IGCC*, 2011.

[61] Mohammad Islam, Shaolei Ren, and Adam Wierman. Exploiting a Thermal Side Channel for Power Attacks in Multi-Tenant Data Centers. In *ACM CCS*, 2017.

[62] JEDEC Solid State Technology Association. Failure Mechanisms and Models for Semiconductor Devices. *JEDEC Publication JEP122-B*, 2003.

[63] Zhixiong Jiang, Chunyang Lu, Yushan Cai, Zhiying Jiang, and Chongya Ma. VPower: Metering Power Consumption of VM. In *IEEE ICSESS*, 2013.

[64] Andrew Kahng, Siddhartha Nath, and Tajana Rosing. On Potential Design Impacts of Electromigration Awareness. In *IEEE ASP-DAC*, 2013.

[65] Mehmet Kayaalp, Dmitry Ponomarev, Nael Abu-Ghazaleh, and Aamer Jaleel. A High-Resolution Side-Channel Attack on Last-Level Cache. In *IEEE DAC*, 2016.

[66] AJ Kleinosowski, Phil Oldiges, Richard Williams, and Paul Solomon. Modeling Single-Event Upsets in 65-nm Silicon-on-Insulator Semiconductor Devices. *IEEE Transactions on Nuclear Science*, 53(6), 2006.

[67] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Annual International Cryptology Conference*, 1999.

[68] Panagiotis Kokkinos, Dimitris Kalogeras, Anna Levin, and Emmanouel Varvarigos. Survey: Live migration and disaster recovery over long-distance networks. *ACM Computing Surveys*, 2016.

[69] Joonho Kong, Johnsy John, Eui Chung, Sung Chung, and Jie Hu. On the Thermal Attack in Instruction Caches. *IEEE Transactions on Dependable and Secure Computing*, 2010.

[70] André König and Ralf Steinmetz. Detecting Migration of Virtual Machines. In *EuroView 2011*, 2011.

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

[72] Bhavani Krishnan, Hrishikesh Amur, Ada Gavrilovska, and Karsten Schwan. VM Power Metering: Feasibility and Challenges. *ACM SIGMETRICS Performance Evaluation Review*, 2011.

[73] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Power Capping: A Prelude to Power Shifting. *Cluster Computing*, 2008.

[74] Chao Li, Zhenhua Wang, Xiaofeng Hou, Haopeng Chen, Xiaoyao Liang, and Minyi Guo. Power Attack Defense: Securing Battery-Backed Data Centers. In *IEEE ISCA*, 2016.

[75] Li Li, Wenli Zheng, Xiaodong Wang, and Xiaorui Wang. Coordinating Liquid and Free Air Cooling with Workload Allocation for Data Center Power Minimization. In *USENIX ICAC*, 2014.

[76] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-Level Cache Side-Channel Attacks are Practical. In *IEEE S&P*, 2015.

[77] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and Cooling Aware Workload Management for Sustainable Data Centers. In *ACM SIGMETRICS*, 2012.

[78] Diego Llanos and Belén Palop. TPCC-UVa: An Open-Source TPC-C Implementation for Parallel and Distributed Systems. In *IEEE IPDPS*, 2006.

[79] Kai Ma and Xiaorui Wang. PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs. In *ACM PACT*, 2012.

[80] Ioannis Manousakis, Íñigo Goiri, Sriram Sankar, Thu Nguyen, and Ricardo Bianchini. CoolProvision: Underprovisioning Datacenter Cooling. In *ACM SoCC*, 2015.

[81] Ali José Mashtizadeh, Min Cai, Gabriel Tarasuk-Levin, Ricardo Koller, Tal Garfinkel, and Sreekanth Setty. XvMotion: Unified Virtual Machine Migration over Long Distance. In *USENIX ATC 14*, 2014.

[82] Ramya Jayaram Masti, Devendra Rai, Aanjhan Ranganathan, Christian Müller, Lothar Thiele, and Srdjan Capkun. Thermal Covert Channels on Multi-core Platforms. In *USENIX Security*, 2015.

[83] Christoph Mobius, Waltenegus Dargie, and Alexander Schill. Power Consumption Estimation Models for Processors, Virtual Machines, and Servers. *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[84] Justin Moore, Jeffrey Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers. In *USENIX ATC*, 2005.

[85] Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. Lightweight Virtualization: A Performance Comparison. In *IEEE IC2E*, 2015.

[86] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L Scott. Proactive Fault Tolerance for HPC with Xen Virtualization. In *ACM ICS*, 2007.

[87] Ripal Nathuji and Karsten Schwan. Virtualpower: Coordinated Power Management in Virtualized Enterprise Systems. In *ACM SOSP*, 2007.

[88] Michael Nelson, Beng-Hong Lim, Greg Hutchins, et al. Fast Transparent Migration for Virtual Machines. In *USENIX ATC*, 2005.

[89] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Empirical Exploitation of Live Virtual Machine Migration. In *Proc. of BlackHat DC convention*, 2008.

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

[91] Nathanael Paul, Sudhanva Gurumurthi, and Evans David. Thermal Attacks on Storage Systems. In *IEEE MSST*, 2006.

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

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

[94] Zhiyun Qian, Z Morley Mao, and Yinglian Xie. Collaborative TCP Sequence Number Inference Attack: How to Crack Sequence Number Under a Second. In *ACM CCS*, 2012.

[95] Alan Quach, Zhongjie Wang, and Zhiyun Qian. Investigation of the 2016 Linux TCP Stack Vulnerability at Scale. In *ACM SIGMETRICS*, 2017.

[96] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No Power Struggles: Coordinated Multi-Level Power Management for the Data Center. In *ACM ASPLOS*, 2008.

[97] Anshul Rai, Ramachandran Ramjee, Ashok Anand, Venkata N Padmanabhan, and George Varghese. MiG: Efficient Migration of Desktop VMs Using Semantic Compression. In *USENIX ATC*, 2013.

[98] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *ACM CCS*, 2009.

[99] Jeffry T Russell and Margarida F Jacome. Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors. In *IEEE ICCD*, 1998.

[100] Peter Sacco. Data Center Cooling Best Practices.

[101] Sriram Sankar, Mark Shaw, and Kushagra Vaid. Impact of Temperature on Hard Disk Drive Reliability in Large Datacenters. In *IEEE/IFIP DSN*, 2011.

[102] Kai Shen, Arrvindh Shriraman, Sandhya Dwarkadas, Xiao Zhang, and Zhuan Chen. Power Containers: An OS Facility for Fine-Grained Power and Energy Management on Multicore Servers. *ACM ASPLOS*, 2013.

[103] Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. Server-Storage Virtualization: Integration and Load Balancing in Data Centers. In *ACM/IEEE SC*, 2008.

[104] Matt Skach, Manish Arora, Chang-Hong Hsu, Qi Li, Dean Tullsen, Lingjia Tang, and Jason Mars. Thermal Time Shifting: Leveraging Phase Change Materials to Reduce Cooling Costs in Warehouse-Scale Computers. In *IEEE ISCA*, 2015.

[105] Sergei Skorobogatov. Local Heating Attacks on Flash Memory Devices. In *IEEE HOST*, 2009.

[106] Cristian Constantin Spoiala, Alin Calinciuc, Corneliu Octavian Turcu, and Constantin Filote. Performance comparison of a WebRTC server on Docker versus Virtual Machine. In *IEEE DAS*, 2016.

[107] Jayanth Srinivasan, Sarita V Adve, Pradip Bose, and Jude A Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *IEEE ISCA*, 2004.

[108] Degang Sun, Jie Zhang, Wei Fan, Tingting Wang, Chao Liu, and Weiqing Huang. SPLM: Security Protection of Live Virtual Machine Migration in Cloud Computing. In *ACM SCC*, 2016.

[109] Petter Svärd, Benoit Hudzia, Johan Tordsson, and Erik Elmroth. Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines. In *ACM VEE*, 2011.

[110] Qinghui Tang, Tridib Mukherjee, Sandeep KS Gupta, and Phil Cayton. Sensor-Based Fast Thermal Evaluation Model for Energy Efficient High-Performance Datacenters. In *IEEE ISSNIP*, 2006.

[111] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient Cache Attacks on AES, and Countermeasures. *Journal of Cryptology*, 2010.

[112] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. A Placement Vulnerability Study in Multi-Tenant Public Clouds. In *USENIX Security*, 2015.

[113] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *IEEE Cloud*, 2009.

[114] Paul Watson. Slipping in the Window: TCP Reset attacks. 2004.

[115] Timothy Wood, Prashant J Shenoy, Arun Venkataramani, Mazin S Yousif, et al. Black-box and Gray-box Strategies for Virtual Machine Migration. In *USENIX NSDI*, 2007.

[116] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. Dynamo: Facebook's Data Center-Wide Power Management System. *IEEE ISCA*, 2016.

[117] Zhenyu Wu, Mengjun Xie, and Haining Wang. On Energy Security of Server Systems. *IEEE Transactions on Dependable and Secure Computing*, 2012.

[118] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In *USENIX Security*, 2012.

[119] Jidong Xiao, Zhang Xu, Hai Huang, and Haining Wang. Security Implications of Memory Deduplication in a Virtualized Environment. In *IEEE/IFIP DSN*, 2013.

[120] Qiuyu Xiao, Michael K. Reiter, and Yinqian Zhang. Mitigating Storage Side Channels Using Statistical Privacy Mechanisms. In *ACM CCS*, 2015.

[121] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An Exploration of L2 Cache Covert Channels in Virtualized Environments. In *ACM CCSW*, 2011.

[122] Zhang Xu, Haining Wang, and Zhenyu Wu. A Measurement Study on Co-Residence Threat Inside the Cloud. In *USENIX Security*, 2015.

[123] Zhang Xu, Haining Wang, Zichen Xu, and Xiaorui Wang. Power Attack: An Increasing Threat to Data Centers. In *NDSS*, 2014.

[124] Fan Yao, Miloš Doroslovački, and Guru Venkataramani. Are Coherence Protocol States Vulnerable to Information Leakage? In *IEEE HPCA*, 2018.

[125] Fan Yao, Guru Venkataramani, and Miloš Doroslovački. Covert timing channels exploiting non-uniform memory access based architectures. In *ACM GLSVLSI*, 2017.

[126] Fan Yao, Jingxin Wu, Suresh Subramaniam, and Guru Venkataramani. Wasp: Workload adaptive energy-latency optimization in server farms using server low-power states. In *IEEE CLOUD*, 2017.

[127] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security*, 2014.

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

[129] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K. Reiter. HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis. In *IEEE S&P*, 2011.

[130] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-VM Side Channels and Their Use to Extract Private Keys. In *ACM CCS*, 2012.

[131] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-Tenant Side-Channel Attacks in PaaS Clouds. In *ACM CCS*, 2014.

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

[133] Wenli Zheng, Kai Ma, and Xiaorui Wang. Exploiting Thermal Energy Storage to Reduce Data Center Capital and Operating Expenses. In *IEEE HPCA*, 2014.