Chapter 8
Internet of Things Security and Privacy

8.1 Introduction

The Internet of Things (IoT) promises to make our lives more convenient by turning each physical object in our surrounding environment into a smart object that can sense the environment, communicate with the remaining smart objects, perform reasoning, and respond properly to changes in the surrounding environment. However, the conveniences that the IoT brings are also associated with new security risks and privacy issues that must be addressed properly. Ignoring these security and privacy issues will have serious effects on the different aspects of our lives including the homes we live in, the cars we ride to work, and even the effects that will reach our own bodies.

If your home does not already have a smart meter, it will soon have multiple of those meters that are dedicated to monitor and control the power consumption, the heating, and the lighting of your house. This is not to mention the smart gadgets that will be found all over your house such as the smart camera that notifies your smartphone during business hours when movement is detected, the smart door that opens remotely, and the smart fridge that notifies you when you are short of milk. Imagine now the level of control that an attacker can gain by hacking those smart meters and gadgets if the security of those devices was overlooked. In fact, the damage caused by cyberattacks in the IoT era will have a direct impact on all the physical objects that you use in your daily life. The same applies to your smart car as the number of integrated sensors continues to grow rapidly and as the wireless control capabilities increase significantly over time, giving an attacker who hacks the car the ability to control the windshield wipers, the radio, the door lock, and even the brakes and the steering wheel of your car. Our bodies will not also be safe from cyberattacks. In fact, researchers have shown that an attacker can control remotely the implantable

A. Rayes, S. Salam, *Internet of Things from Hype to Reality*,
https://doi.org/10.1007/978-3-030-90158-5_8

and wearable health devices (e.g., insulin pumps and heart pacemakers) by hacking the communication link that connects them to the control and monitoring system. This gives the attacker, for example, the ability to tune the injected insulin dose causing serious health problems that may even cause death to patients wearing those smart health devices. In fact, such concerns have made doctors disable the wireless capability of the heart pacemaker of Dick Cheney, the former US vice president, in order to protect him from such malicious attacks.

The security risks are also extremely serious when IoT devices are used in business enterprises. If an attacker hacks any of those smart objects that are used in a big enterprise, then the sensing capabilities that those smart objects have can be used by the attacker to spy on the enterprise. Such cyberattacks can also be used to steal sensitive information such as the company earnings report and credit card information. In fact, these stealing attacks are common in big enterprises such as the largest financial hacking case in the US history, which took place in 2013, where a group of five hackers stole $160 million from credit cards and over hundreds of millions in criminal loot.

Maintaining users' privacy in IoT is also crucial as there is an enormous amount of information that an outsider can learn about people's life by eavesdropping on the sensed data that their smart house appliances and wearable devices report. In fact, people will be living in a "Big Brother" world where smart things record our daily activities anytime and everywhere. The advances in the fields of facial, speech, and human activity recognition amplify the amount of information that the sensed data can reveal if it falls in the wrong hands. Even if your IoT objects are merely reporting metadata, you would be surprised by the amount of information that an outsider can learn about your personal life when aggregating the metadata collected from multiple hacked objects that surround you over time. It is thus essential to find solutions to preserve people's privacy in the IoT era.

The objective of this chapter is to shed the light on some of the security and privacy issues that the IoT paradigm is exposed to. We also survey the techniques that were proposed to address these issues. Some of the discussed techniques prevent security breaches from taking place, while others try to detect malicious behavior and trigger an appropriate mitigating countermeasure. The rest of the chapter is organized as follows. Section 8.2 identifies the new security challenges that are encountered in the IoT paradigm. Section 8.3 identifies the IoT security requirements. Section 8.4 briefly describes the three domains in the IoT architecture. Sections 8.5–8.7 survey the security attacks and countermeasures at the cloud domain, the fog domain, and the sensing domain, respectively. Section 8.8 discusses approaches for securing IoT Devices. The section starts by providing several examples of IoT devices used in security attacks, and then discusses solutions including MUD and DICE. Finally, Sect. 8.9 summarizes the chapter and provides directions for future work related to the area of IoT security.

8.2 IoT Security Challenges

IoT has unique characteristics and constraints when it comes to designing efficient defensive mechanisms against cybersecurity threats that can be summarized by:

1. *Multiple Technologies*: IoT combines multiple technologies such as radio-frequency identification (RFID), wireless sensor networks, cloud computing, virtualization, etc. Each of these technologies has its own vulnerabilities. The problem with the IoT paradigm is that one must secure the chain of all of those technologies as the security resistance of an IoT application will be judged based on its weakest point which is usually referred to by Achilles' heel.
2. *Multiple Verticals*: The IoT paradigm will have numerous applications (also called verticals) that span eHealth, industrial, smart home gadgets, smart cities, etc. The security requirements of each vertical are quite different from the remaining verticals.
3. *Scalability*: According to Cisco, 26.3 billion smart devices will be connected to the Internet by 2020. This huge number makes scalability an important issue when it comes to developing efficient defensive mechanisms. None of the previously proposed centralized defensive frameworks can work anymore with the IoT paradigm, where the focus must be switched to finding practical decentralized defensive security mechanisms. An IoT solution needs to scale cost-effectively, potentially to hundreds of thousands or even millions of endpoints.
4. *Availability:* Availability refers to characteristic of a system or subsystem that is continuously operational for a desirably long period of time. It is typically measured relative to "100% operational" or "never failing." A widely held but difficult-to-achieve standard of availability for a system or product is known as "five 9 s" (available 99.999% of the time in a given year) availability. Security plays a major rule in high availability as network administrators often hesitate to use needed threat-response technology functions (e.g., network discovery as illustrated in Chap. 7) for fear that such functions will take down critical systems. Even a simple port scan causes some IoT devices to stop working, and the cost of downtime can far exceed the cost of remediating all but the most severe incidents. In some instances, network administrators would rather have no cybersecurity protection rather than risk an outage due to a false positive. This leaves them blind to threats within their control networks. Companies often add redundancy to their systems so that failure of a component does not impact the entire system.
5. *Big Data*: Not only the number of smart objects will be huge, but also the data generated by each object will be enormous as each smart object is expected to be supplied by numerous sensors, where each sensor generates huge streams of data over time. This makes it essential to come up with efficient defensive mechanisms that can secure these large streams of data.
6. *Resource Limitations*: The majority of IoT end devices have limited resource capabilities such as CPU, memory, storage, battery, and transmission range. This makes those devices a low-hanging-fruit for denial of service (DoS) attacks

where the attacker can easily overwhelm the limited resource capabilities of those devices causing a service disruption. In addition to that, the resource limitations of those devices raise new challenges when it comes to developing security protocols especially with the fact that the traditional and mature cryptography techniques are known to be computationally expensive.

7. *Remote Locations*: In many IoT verticals (e.g., smart grid, railways, roadsides), IoT devices, epically sensors, will be installed in unmanned locations that are difficult to reach. Attackers can interfere with these devices without being seen. Cyber and physical security monitoring systems must be installed in safeguarded location, operate in extreme environmental conditions, fit in small spaces, and operate remotely for routine updates and maintenance avoiding delayed and expensive visits by network technicians.
8. *Mobility*: Smart objects are expected to change their location often in the IoT paradigm. This adds extra difficulties when developing efficient defensive mechanisms in such dynamic environments.
9. *Delay-Sensitive Service*: The majority of IoT applications are expected to be delay-sensitive, and thus one should protect the different IoT components from any attack that may degrade their service time or may cause a service disruption.

8.3 IoT Security Requirements

We summarize in this section the security requirements for IoT. These requirements include:

- *Confidentiality*: ensures that the exchanged messages can be understood only by the intended entities.
- *Integrity*: ensures that the exchanged messages were not altered/tampered by a third party.
- *Authentication*: ensures that the entities involved in any operation are who they claim to be. A masquerade attack or an impersonation attack usually targets this requirement where an entity claims to be another identity.
- *Availability*: ensures that the service is not interrupted. Denial of service attacks target this requirement as they cause service disruption.
- *Authorization*: ensures that entities have the required control permissions to perform the operation they request to perform.
- *Freshness*: ensures that the data is fresh. Replay attacks target this requirement where an old message is replayed in order to return an entity into an old state.
- *Non-repudiation*: ensures that an entity cannot deny an action that it has performed.
- *Forward Secrecy*: ensures that when an object leaves the network, it will not understand the communications that are exchanged after its departure.
- *Backward Secrecy*: ensures that any new object that joins the network will not be able to understand the communications that were exchanged prior to joining the network.

8.4 IoT Three-Domain Architecture

Before introducing IoT security issues, we briefly describe in this section the three-domain architecture that we consider in our security analysis.

As illustrated in Figs. 8.1 and 8.2, the architecture is made up of the following three domains:

1. *IoT Sensing Domain*: This domain is made up of all the smart objects that have the capability to sense the surrounding environment and report the sensed data to one of the devices in the fog domain. The smart objects in the sensing domain are expected to change their location over time.

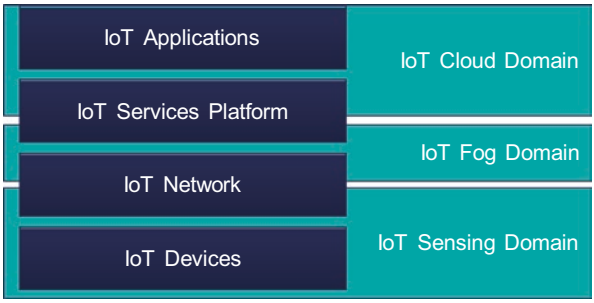

Fig. 8.1 Mapping of IoT domains

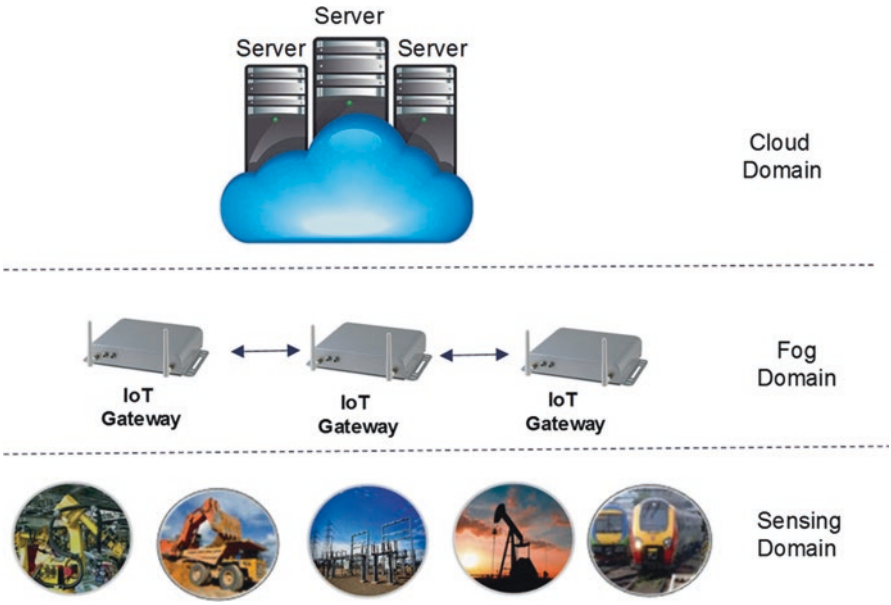

Fig. 8.2 The IoT domains

2. *Fog Domain*: This domain consists of a set of fog devices that are located in areas that are highly populated by many smart objects. Each fog device is allocated a set of smart objects where the allocated objects report their sensed data to the fog device. The fog device performs operations on the collected data including aggregation, preprocessing, and storage. Fog devices are also connected with each other in order to manage the communication among the smart objects and in order to coordinate which fog device will be responsible for handling which object as objects change their location over time. Each fog device is also connected to one or multiple servers in the cloud domain.
3. *Cloud Domain*: This domain is composed of a large number of servers that host the applications that are responsible for performing the heavy-computational processing operations on the data reported from the fog devices.

We analyze in the following sections the security attacks and countermeasures at each one of those three domains. We follow a top-down order where we describe the attacks and countermeasures that are encountered at the cloud domain, the fog domain, and the sensing domain. For each one of those domains, we identify the most popular security attacks and then describe how these attacks are launched, what vulnerabilities they exploit, and what countermeasure techniques can be used to prevent, detect, or mitigate those attacks.

8.5 Cloud Domain Attacks and Countermeasures

As mentioned earlier, the cloud domain holds the IoT applications that are performing different operations on the data collected by the IoT objects. Each IoT application is dedicated one or multiple virtual machines (VMs) where each VM is assigned to one of the servers in the cloud data center and gets allocated certain amount of CPU and memory resources in order to perform certain computing tasks. The cloud data center is made up of thousands of servers where each server has certain CPU, memory, and storage capacities, and thus each server has a limit on the number of VMs that it can accommodate. The servers in the cloud data center are virtualized which allows multiple VMs to be assigned to the same server as long as the server has enough resource capacity to support the resource requirements of each hosted VM. Figure 8.3 shows an illustration of how multiple VMs can be assigned to the same server, thanks to virtualization (more details on virtualization were discussed in Chap. 6). Each IoT application is hosted on a VM that has its own operating system (OS). The hypervisor (sometimes also called the virtual machine manager) monitors those running VMs and manages how these VMs share the server's hardware. The hypervisor also provides the logical separation among the VMs and also separates each VM from the underlying hardware. The hypervisor has also a migration module that manages how to move a VM that is currently hosted on the server to another server. The migration module also manages the reception of a VM that is moved from other servers.

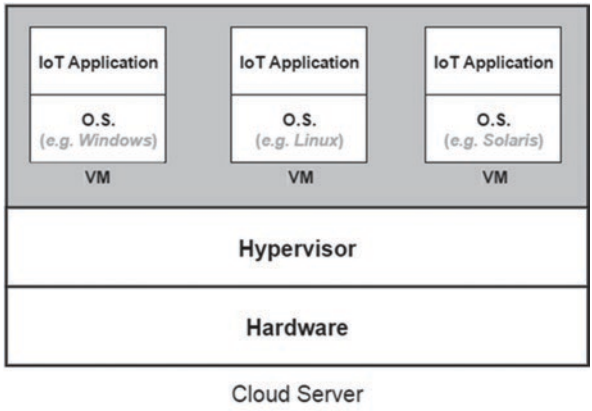

Fig. 8.3 Illustration of how multiple IoT applications can be hosted on the same server, thanks to virtualization

Cloud computing is considered a high-risk environment for many businesses and consumers as they feel its perimeter cannot be defined nor controlled. In addition, many government agencies must comply with regulatory statutes, such as the Health Insurance Portability and Accountability Act (HIPAA), the Sarbanes-Oxley Act of 2002 (SOX), and the Federal Information Security Management Act (FISMA). The IoT applications running in the cloud domain are susceptible to numerous security attacks. We summarize next the most popular ones:

1. *Hidden-Channel Attacks:* Although there is a logical separation among the VMs running on the same server, there are still some hardware components that are shared among those VMs such as the cache. This opens opportunities for data leakage across the VMs that reside on the same server. Three steps are followed by the attacker in order to leak information from a target VM. These three steps are explained next:

   (a) ***Step1*: *Mapping Target VM***: The first step toward launching an attack against a VM in a cloud data center is to locate where the target VM resides. A cloud data center is typically divided into multiple management units called clusters, where each cluster is located in a certain geographical location and is made up of thousands of servers. Each cluster is divided into multiple zones (sometimes called "pods") where each zone consists of a large number of servers. Although clients have the choice to specify in which cluster their VM resides, they do not have control on selecting the zone or the server within the zone where their VM will reside as this decision is made based on the cloud provider's scheduling algorithm which is not released publicly. In order to know where a target VM resides, the attacker needs only to know the external IP address of that VM where each VM hosted on the cloud has usually two IP addresses: an external address used to communicate with any entity that is located outside the cloud cluster and an internal address used only within the cloud cluster and is only visible within the cloud cluster. The attacker can infer based on the VM's external

IP address on what cluster the VM resides, as cloud clusters are usually placed in different geographical locations and have different IP addresses. Now in order to identify in what zone within the cluster the target VM resides, the attacker needs to know the target VM's internal IP address as the internal IP addresses for all VMs within the same zone have the same network prefix. In order to identify the VM's internal IP address, the attacker rents a VM in the same cluster as the one where the VM resides. The rented VM is then used to query the DNS server of the cloud cluster where the internal IP address of the target VM can be fetched. By observing the internal IP address of the target VM in the DNS query, the attacker can tell what zone within the cloud cluster the VM is hosted in.

(b) ***Step2*: *Malicious VM Placement***: having identified on what cluster and on what zone the target VM resides, the next step toward launching an attack against the target VM is to place a malicious VM on the same server where the target VM resides. In order to do that, the attacker rents a VM in the same cluster as the target VM. The cloud provider's scheduling algorithm places the rented VM on one of the servers within one of the cluster's zones. The attacker performs a traceroute from the rented VM to the target VM where the routing path that separates the rented VM and the target VM is identified. If the identified routing path shows multiple hops that separate the target VM and the rented VM, then the attacker knows that the rented VM was not placed on the same server as the target VM. The attacker then releases the rented VM and requests a new one. The cloud provider's scheduling algorithm selects a server to host the requested VM. The attacker performs a traceroute from the new rented VM to the target VM in order to know whether or not the target VM and the new rented VM reside on the same server. The attacker continues releasing then renting new VMs and performing a traceroute until he/she identifies that the cloud provider's scheduling algorithm has placed the rented VM on the same server as the target VM.

(c) ***Step3*: *Cross-VM Data Leakage***: Having placed a malicious VM on the same server as the target VM, the attacker now tries to learn some information about the target VM by exploiting the fact that although VMs are separated logically, thanks to virtualization, they still share certain parts of the server's hardware such as the instruction cache and the data cache. The attacker can now, for example, learn what lines of cache (data or instruction) the target VM has accessed recently. This can be done as follows. When the shared cache is assigned to the malicious VM that is under the control of the attacker, the attacker fills the whole shared cache by dummy data. The malicious VM then yields the shared cache to the target VM which performs some data access operations. The malicious VM sends an interrupt after a short time from yielding the cache to the target VM asking to assess the cache so that the target VM yields the cache for the malicious VM. Now the malicious VM probes the different lines of the cache asking to fetch the dummy data that were previously filled in the cache. By observing the time it takes to access each chunk of the dummy data, the malicious VM can tell

which chunks of the dummy data were fetched from the cache and which chunks were fetched from memory as they were replaced by data that was accessed by the target VM. This gives information to the malicious VM about what addresses the target VM has accessed recently. Knowing what addresses the target VM accesses over time can help the malicious VM recover parts of the security keys that the target VM is using.

(d) Different countermeasures can be taken to prevent hidden-channel attacks from taking place. The first twos steps needed to launch this attack (mapping the target VM and placing a malicious VM on the same server as the target VM) can be prevented by not allowing the VMs hosted in the cloud data center to send probing packets such as traceroute packets. Preventing data from being leaked across VMs that are hosted on the same server can be achieved by one of the following techniques:

- *Hard Isolation*: The basic idea behind this preventive technique is to maintain high levels of isolation among the VMs. One way to do this is to separate the cache dedicated for each VM through hardware or software. Another way to achieve hard isolation is by assigning only one VM to each server. Although this completely prevents data leakages across VMs, it is not a practical solution as it leaves the servers within the cloud data center underutilized. A better way to achieve hard isolation is by letting each cloud client specify a list of trusted cloud users called the *white list*. The cloud client is fine with sharing the server with only the VMs belonging to the *white list users*. New scheduling algorithms are needed in that case in order to decide on what server each VM should be placed such that the security constraints of each VM that are specified by the white and black lists are met. A key limitation of this technique is that each VM must have a list of identified untrusted VMs.
- *Cache Flushing*: This technique flushes the shared cache every time the allocation of the cache is switched from a VM to another. The downside of this countermeasure is that the VMs running on the server will experience frequent performance degradation as the shared cache will be emptied every time a switch from a VM to another occurs, which increases the time needed to access and fetch data.
- *Noisy Data Access Time*: This technique adds random noise to the amount of time needed to fetch data, which makes it hard to tell whether or not the data was fetched from the cache or from the memory. By doing this, it becomes harder for a malicious VM to identify what segments of the cache were populated by another VM that shares the same server. Of course this has a price as the fetched data gets delayed a little bit due to the noise (variable time delay) that is added to the time needed to fetch the data.
- *Limiting Cache Switching Rate*: A mitigation technique to limit the amount of data that can be leaked across VMs can be achieved by limiting how often the cache is switched from a VM to another. The idea here

is that if the cache is not switched from a VM to another too soon, then the content of the cache will be modified a lot by the VM that possess the cache. This makes it hard for another VM to attain fine-grained knowledge of what data the previous VM has accessed when probing the cache.

2. *VM Migration Attacks*: The virtualization technology supports live *VM migration*, which allows moving a VM transparently from a server to another. The term *live* refers here to the fact that the application running on the VM is disrupted for a very short duration due to this migration where the disruption is as low as hundreds of milliseconds. Before delving into the security issues that VM migration brings, we explain briefly the mechanism for performing VM migration and the scenarios where VM migration is usually performed.

   The mechanism of moving a VM from a source server to a destination server is done by copying the VM's memory content. The VM's hard disk content does not need to be copied as it is usually stored on a network-attached storage (NAS) device and can be accessed from any location within the cloud cluster. If the destination server where the VM will be moved to lies on the same local network as the source server, then the VM keeps the same IP address even after migration in order to avoid the need for communication redirection. Maintaining the same IP address even after moving to another server is done after copying the memory content of the VM by sending a gratuitous ARP reply packet that informs the routing devices within the cloud about the VM's new physical address, so that any packet destined to the VM's IP address gets routed to the VM's new location on the destination server. Each server has a dedicated module in the hypervisor called the VM migration module that is responsible for sending the VM content for the source server or receiving the VM's memory content for the destination server.

   VM migration is very useful in multiple scenarios. Consider, for example, the case when a server that is hosting some VMs needs to be taken offline for maintenance or for patch installation. VM migration can be used in this case to move all the VMs currently running on the server into other servers so that the server can be taken down for maintenance without terminating the running VMs that are hosted on that server. VM migration is also a very useful tool for managing the servers in the cloud data center where it can be used to balance the workload among the servers or to consolidate the scheduled VMs on fewer number of powered servers so that a larger number of servers can be powered down to save energy. However, the conveniences that VM migration brings raise new security threats. The attacks that exploit VM migration can be divided into two subcategories based on the target plane:

   (a) *Control Plane Attacks*: These attacks target the module that is responsible for handling the migration process on a server which is called the migration module that is found in the hypervisor. By exploiting a bug in the migration module software, the attacker can hack the server and take full control over the migration module. This gives the attacker the ability to launch malicious activities including:

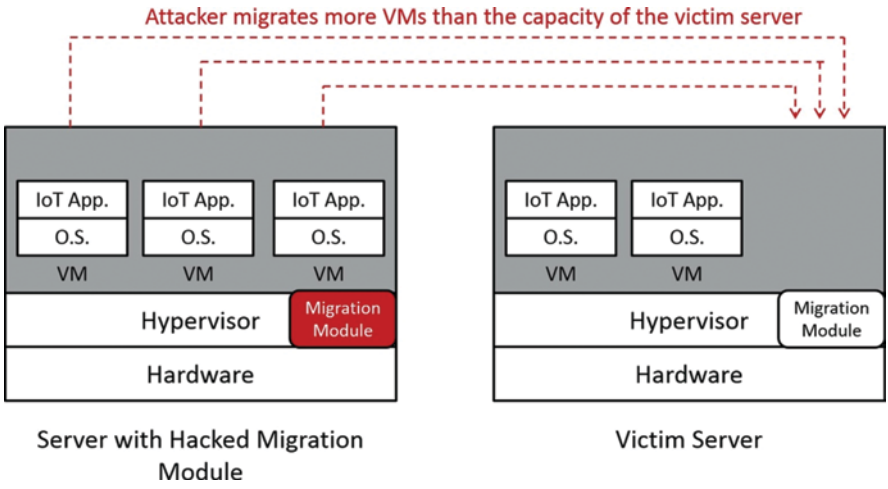

Fig. 8.4 Illustration of the migration flooding attack

- *Migration Flooding*: This attack is illustrated in Fig. 8.4 where the attacker moves all the VMs that are hosted on the hacked server to a victim server that does not have enough resource capacity to host all the moved VMs. This causes a denial of service of the applications running in the VMs of the victim server as there will not be enough resources to satisfy the demands of all the hosted VMs leading into VM performance degradation and VM crashes.
- *False Resource Advertising*: The hacked server claims that it has a large resource slack (a large amount of free resources). This attracts other servers to off-load some of their VMs to the hacked server so that the cloud workload gets distributed over the cloud servers. After moving VMs from other servers to the hacked server, the attacker can exploit other vulnerabilities to break into the offloaded VMs as now these VMs are placed on a server that is under the control of the attacker.

(b) *Data Plane Attacks*: These constitute the second type of VM migration attacks, and those attacks target the network links over which the VM is moved from a server to another. Examples of data plane attacks include:

- *Sniffing Attack*: where an attacker sniffs the packets that are exchanged between the source and destination and reads the migrated memory pages.
- *Man-in-the-Middle Attack*: the attacker fabricates a gratuitous ARP reply packet similar to the one that is usually sent when a VM moves from a server to another. This fabricated ARP packet informs the routing devices that the physical address where the victim VM resides was changed to become the physical address of the attacker's malicious VM. Now the incoming packets that are destined to the victim get routed to the new physical address where the attacker resides. The attacker can then pas-

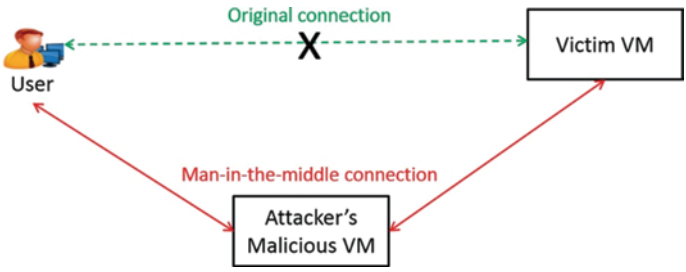

Fig. 8.5 Man-in-the-middle attack

sively monitor the received packets while continuing to forward them to the actual physical address where the victim VM resides so that the victim does not detect that any malicious activity is going on. The attacker can also modify the content of the received packets if the integrity of the packets is not protected by any security mechanism. An illustration of the man-in-the-middle attack is shown in Fig. 8.5.

- Having explained the VM migration attacks, we now discuss the possible countermeasures. Unfortunately, little attention was given to secure VM migration where the focus was more on how to optimize the performance degradation or the energy overhead associated with those migrations. In order to secure VM migration, mutual authentication should be performed between the server initiating the migration and the server that will be hosting the migrated VM. The control messages that are exchanged between the servers to manage the migration should also be encrypted and signed by the entity that is generating those control messages in order to avoid altering the content of those control messages and in order to prevent other entities from fabricating fake control messages. Sequence numbers or timestamps should also be included in the exchanged control messages in order to prevent a malicious entity from replaying an old control message that was sent earlier. Also, gratuitous ARP Reply packets that update the physical address of the VM should be accepted only after authentication in order to prevent man-in-the-middle attacks. The reader interested in learning more about VM migration attacks and countermeasures is referred to [19] for further information on this topic.

3. *Theft-of-Service Attack*: In this attack a malicious VM misbehaves in a way that makes the hypervisor assigns to it more resources than the share it is supposed to obtain. This extra allocation of resources for the malicious VM comes at the expense of the other VMs that share the same server as the malicious VM, where these victim VMs get allocated less share of resources than what they should actually obtain, which in turn degrades their performance.

   Xen is a well-known hypervisor that is susceptible to this attack. One of the main roles of Xen hypervisor is to decide to which VM among the ones running

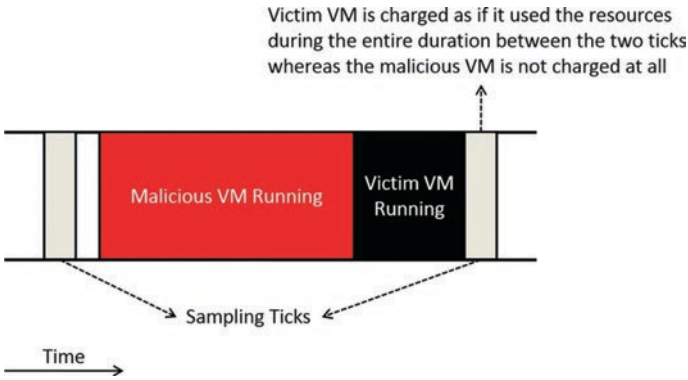

Fig. 8.6 Illustration of the theft-of-service attack

on the server each physical core should be assigned to over time. In order to do that, Xen samples every 10 ms to check the VMs that are utilizing the cores. Xen then assumes that the VM that is detected to be using one of the cores at the sampling time has been using the server's core during the entire 10 ms. The hypervisor then calculates how much time each VM has been assigned the cores. VMs that utilized the cores less than the remaining VMs are given higher priority to utilize the server's core in the future in order to guarantee a fair allocation of the shared resources.

The fact that Xen performs periodic sampling can be exploited by a malicious VM by using one of the cores at times other than the sampling time. As illustrated in Fig. 8.6, the malicious VM can yield the acquired core to another VM shortly before the sampling tick. The hypervisor then assumes that the other VM that has yielded the core has been using the core during the entire 10 ms. The malicious VM does not get logged as using the core and thus keeps having high priority to use the cores in the future.

Two countermeasures were proposed to handle this attack. The first countermeasure is to log more accurately the start and end time when each VM was utilizing the cores using accurate clocks. Another solution is to randomize the sampling times.

4. *VM Escape Attack*: Virtual machines are designed in a way that isolate each VM from the other VMs running on the same server, which prevents VMs from accessing data that belongs to other VMs that reside on the same server. However, in reality software bugs can be exploited to break this isolation. If a VM escapes the hypervisor layer and reaches the server's hardware, then the malicious VM can gain root access to the whole server where it resides. This gives the VM full control on all the VMs hosted on the hacked server. Different techniques were proposed to prevent a malicious VM from bypassing the hypervisor layer and obtaining the root privileges. An example of such techniques is CloudVisor which basically adds an extra isolation layer between the hardware and the hypervisor through nested virtualization that prevents the malicious VM from

obtaining the root privileges even if it bypasses the hypervisor layer. Other architecture solutions were also proposed to avoid VM escape attacks and could be found in [28].

5. *Insider Attacks*: In all the previously discussed attacks, we were treating the administrators of the cloud data center as trusted entities, and we were focusing only on the attacks that are originating from other malicious VMs that are hosted in the cloud data center. However, some sensitive applications may have serious concerns about hosting their collected information on the cloud data center in the first place as the cloud data center administrators will in that case have the ability to access and modify the collected data. Different techniques were proposed to protect the data from these insider attacks. Homomorphic encryption is a form of encryption that can be used to prevent such attacks as it allows the cloud servers to perform certain computing operations on encrypted input data to generate an encrypted result. This encrypted result when decrypted matches the result of performing the computational operation on the unencrypted input data. Applying homomorphic encryption in the IoT paradigm allows cloud servers to perform the necessary processing operations on the encrypted data that is collected from the smart devices without giving the cloud servers the ability to interpret neither the input data nor the result as they are both encrypted using a secret key that is not shared with the cloud. Only the smart objects and the user running the IoT application can interpret these data as they have the key needed for decryption. Another form of protection against insider attacks is to chop the data collected by the smart object into multiple chunks and then to use a secret key to perform certain permutations on those chunks before sending the data to the cloud servers. This allows storing the data on the cloud servers in an uninterpretable form for the cloud administrators. Only authorized entities that have the secret key can return the stored data to an interpretable form by performing the correct permutations.

For convenience, Table 8.1 summarizes all the cloud domain attacks that were discussed in this section. The second, third, and fourth columns of Table 8.1 describe, respectively, the vulnerability that causes this attack, what security requirement each attack violates, and what are the countermeasures that can be used to prevent or detect and mitigate each attack.

8.6 Fog Domain Attacks and Countermeasures

Recall that the fog domain is made up of a set of fog devices where each fog device collects the sensing data that is reported from a set of smart objects. The fog device performs different operations on the collected data which include data aggregation, data preprocessing, and data storage. The fog device may also perform some reasoning operations on the collected data. After processing and aggregating the collected data, the fog device forwards these data to the cloud domain. It is worth

Table 8.1 Summary of the security attacks in the cloud domain

Attack	Vulnerability reason	Security violation	Countermeasures
Hidden-channel attack	Shared hardware components (e.g., cache) among the server's VMs	Confidentiality	Hard isolation Cache flushing Noisy data access time Limiting cache switching rate
VM migration attacks	VM migration software bugs VM migration is performed without authentication Memory pages copied in clear	Confidentiality Integrity Availability	Server authentication Encrypting migrated memory pages
Theft-of-service attack	Periodic sampling of VMs' used resources	Availability Non-repudiation	Fine-grain sampling using high precision clocks Random sampling
VM escape attack	Hypervisor software bugs	Confidentiality Availability Integrity	Add an isolation domain between the hypervisor and hardware
Insider attacks	Lack of trust in cloud administrators	Confidentiality Integrity	Homomorphic encryption Secret storage through data chopping and permutation based on a secret key

mentioning that not only fog devices are connected with the cloud domain, but also fog devices are usually connected with each other in order to allow the fog devices connecting different smart objects to communicate directly with each other and in order to coordinate assigning objects to fog devices as their location changes. Fog devices can be independent components or could be built on top of existing gateways. Each fog device provides computing resources to be used by the IoT smart objects that are located close to the fog device. These computing resources are virtualized in order to allow the connected objects to share the computing resources that are offered by the fog device where each object or set of connected objects are allocated a virtual machine that performs the necessary data processing operations.

One can see that the computing capabilities provided by fog devices are very similar to the computing services provided by the servers in the cloud as they are both virtualized environments. The high similarities between the fog domain and the cloud domain make the fog domain susceptible to all the cloud domain attacks that were described in Sect. 8.5.

Although the fog domain is highly similar to the cloud domain, there are three key differences that distinguish fog devices from cloud servers:

1. *Location*: Unlike cloud servers which are usually located far from smart objects, fog devices are placed in areas with high popular access and thus are placed close to the smart objects. This placement plays an important role in giving the fog devices the ability to respond quickly to changes in the reported data. This also gives the fog devices the ability to provide location-aware services as smart objects connect to the closest fog device, and thus each fog device knows the location of the objects connected to it.

2. *Mobility*: Since the location of the smart object may change over time, then the VMs created to handle those objects at the fog domain must be moved from a fog device into another, in order to keep the processing that is performed in the fog device close to the object that is generating data.
3. *Lower Computing Capacity*: The fog devices that are installed in a certain location are expected to have a lower computing capacity when compared to capacities offered by cloud data centers as the latter are made of thousands of servers.

These characteristics raise new security threats that are specific to the fog domain and that distinguish it from the cloud domain. The security threats that are specific to the fog domain are the following:

- *Authentication and Trust Issues*: The fact that fog devices do not require a large facility space or a high number of servers compared to cloud data centers will encourage many small and less-known companies to install virtualized fog devices in dense areas and to offer these computing resources to be rented by the smart objects that are near the installed fog devices. Unlike cloud data centers which are offered by well-known companies, fog devices are expected to be owned by multiple and less-known entities. An important security concern that needs then to be taken into account when assigning a smart object to a fog device is to authenticate first the identity of the owner of the fog device. Authentication is not enough, as the smart object also needs to decide whether or not the owner of the fog device can be trusted. Trust is an important aspect as a smart object will be assigned to different fog devices belonging to different entities as their location may change over time. Reputation systems such as those that were proposed in peer-to-peer networks in or to rank cloud providers in can be used to select a trustworthy fog device among the available ones in the area surrounding each smart object.
- *Higher Migration Security Risks*: Although VM migration is common in both the cloud and the fog domains, there is an important difference between the migration in the cloud domain and that in the fog domain. While the migrated VMs in the cloud domain are carried over the cloud data center's internal network, the migrations from a fog device into another are carried over the Internet. Thus there is a higher probability that the migrated VMs get exposed to compromised network links or network routers when moving a VM from a fog device into another. This makes it vital to encrypt the migrated VM and to authenticate the VM migration messages that are exchanged among the fog devices.
- *Higher Vulnerability to DoS Attacks*: Since fog devices have lower computing capacities, this makes them a low-hanging-fruit for denial of service (DoS) attacks where attackers can easily overwhelm fog devices when compared to the cloud data centers, where a huge number of servers that have high computing capacity are available.
- *Additional Security Threats Due to Container Usage*: In order to provide the computing needs for a larger number of connected objects, the fog device may use containers rather than VMs to allocate the resource demands for each connected object. The main difference between a container-based virtualization and

full virtualization is the fact that containers share not only the same hardware but also the same operating system with the other containers that are hosted on the same fog device (refer to Chap. 6). This is unlike the full virtualization (which was illustrated in Fig. 8.3) where only the hardware is shared among multiple VMs and each VM has its own operating system. The low overhead of containers allows larger number of objects to be served by the fog device. However, sharing the same operating system among the containers dedicated for objects that belong to different users raises serious security concerns as the opportunities for data leakage and for hijacking the fog device increase significantly. The industry needs to address these gaps in container security to enable IoT applications at scale.

- *Privacy Issues*: We mentioned before that each smart object will be connected to one of the fog devices that are close to it. This means that the fog device can infer the location of all the connected smart objects. This allows the fog device to track users or to know their commuting habits which may break the privacy of the users carrying those objects. New mechanisms should be developed in order to make it harder for fog devices to track the location of the smart objects over time. Furthermore, the advancement in wireless signal processing has made it possible now to identify the presence of humans and track their location, their lip movement, and their heartbeats by capturing and analyzing the wireless signals that are exchanged between the sensing objects and the fog domain. This advancement makes it possible for any entity to install a reception device close to your home that analyzes the wireless signals that are emitted from your home in order to spy on your daily activities. The work in [47] is among the first papers that identified these risks where the authors in that paper propose a device called an obfuscator that prevents leaking such information by emitting signals that make it hard for an unauthorized receiver to infer the amplitude, the frequency, and the time shift of the originally exchanged signals. The obfuscator does not only prevent such leakages but also acts as a relay that rebroadcasts some of the sent messages which increases the transmission rate between the sensing objects and the fog domain.

8.7 Sensing Domain Attacks and Countermeasures

The sensing domain contains all the smart objects, where each object is equipped with a number of sensors that allow the object to perceive the world. The smart object is also supplied with a communication interface that allows it to communicate with the outer world. The smart object reports the sensed data to one of the fog devices in the fog domain. This is done by either creating a direct connection with the fog device if the smart object is directly connected by wires or has the wireless transmission capability to reach that fog device or in a multi-hop fashion where the smart object relies on other smart objects that lie along the path to the fog device to deliver the sensed data (as illustrated in Fig. 8.7).

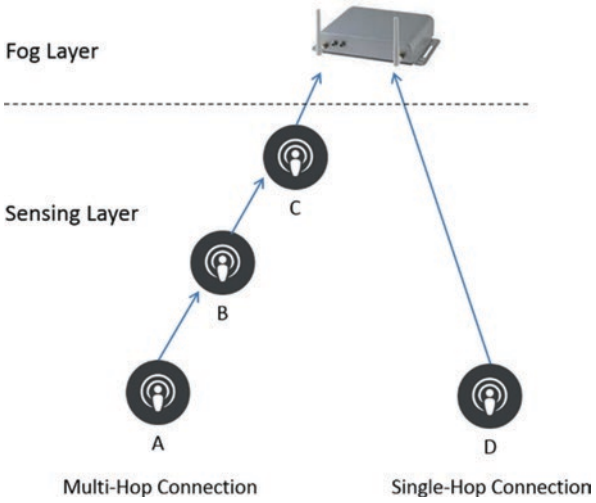

Fig. 8.7 Multi-hop versus direct connection between the smart object and the fog device

The sensing domain is susceptible to multiple attacks. We summarize next some of the most well-known ones:

1. *Jamming Attack*: This attack causes a service disruption and takes one of two forms:
   (a) *Jamming the Receiver*: This attack targets the physical domain in the OSI stack of the receiver (where the receiver is the fog device in the case of a direct connection or another object in the case of a multi-hop connection) where a malicious user (called the jammer) emits a signal (called the jamming signal) that interferes with the legitimate signals that are received at the receiver side. The interference degrades the quality of the received signal causing many errors. As a result, the receiving end does not acknowledge the reception of these damaged packets and waits for the sender to retransmit those packets.
   (b) *Jamming the Sender*: Unlike the previous attack, this type targets the data link layer at the OSI layer of the sending object where the jammer in this attack sends a jamming signal that prevents the neighboring objects from transmitting their packets as they sense the wireless channel to be busy and back off waiting for the channel to become idle.

   There are different jamming strategies that a jammer may follow to launch a jamming attack. The most well-known ones are summarized next:

- *Constant Jamming*: The attacker continuously transmits a random jamming signal all the time. The main limitation of this attack is that it can be detected easily by observing random bits that do not follow the pattern dictated by the MAC protocol. Another main limitation is the fact that it requires the jamming device to be connected to a source of power as it requires lots of energy.

- Deceptive Jamming: This is similar to the constant jamming with the exception that the jammer conceals its malicious behavior by transmitting legitimate packets that follow the structure of the MAC protocol rather than sending random bits.
- Reactive Jammi*ng*: This is a strategy for jamming the receiver that is suitable for the case when the jamming device has a limited power budget. The jammer in that case listens to the medium and transmits a jamming signal only after it senses that a legitimate signal is being transmitted in the medium. This is more power efficient than continuously transmitting signals as listening to the channel consumes less power than transmitting signals.
- *Random Jamming*: The jammer alternates between sending a jamming signal and remaining idle for random periods of time in order to hide the malicious activity.
- More sophisticated jamming attacks have also emerged that intend to increase the service disruption time, reduce the probability of detection, increase the abilities to recover from the countermeasure that the victim node may take, while also reducing the power that the jamming device requires. An example of a power efficient advanced jamming attack would be to jam only the acknowledgment packets that nodes exchange rather than jamming the whole transmitted data packets as the former are shorter than the latter and thus require less power to jam while causing the same damage.
- Different preventive and detective techniques were proposed to address jamming attacks. We summarize next the most popular ones:
- *Frequency Hopping*: This is a preventive technique where the sender and receiver switch from a frequency to another in order to escape from any possible jamming signal (IEEE 802.15.4 TSCH discussed in Chap. 5 is an example of a wireless technology that employs this technique). Switching from a frequency to another is based on a generated random sequence that is known only for the sender and receiver. If the jammer is aware of the use of this preventive strategy, then the jammer has to switch from a frequency to another trying to collide with the frequency used by the sender and receiver. The interaction between the hopping strategies of the legitimate nodes and that of the jammer in that case can be modelled as a two-player game, where game theory can be used to come up with a hopping strategy that reduces the chances of colliding with the frequency sequence of the jammer.
- *Spread Spectrum*: This technique uses a hopping sequence that converts the narrow band signal into a signal with a very wide band, which makes it harder for malicious users to detect or jam the resulting signal. This technique is also very efficient when the transmitted data are protected by an error-correction technique as it allows the reconstruction of the original signal even if few bits of the transmitted data were jammed by the attacker.
- *Directional Antennas*: The use of directional antennas can mitigate jamming attacks from being successful as the sender and receiver antennas will have less sensitivity to the noise coming from the random directions that are different from the direction that connects the sender and the receiver.
- *Jamming Detection*: Different detective techniques were proposed in the literature to detect jamming attacks. The receiver can detect that it is a victim of a

jamming attack by collecting features such as the received signal strength (RSS) and the ratio of corrupted received packets. Advanced machine learning technique can then be used to differentiate jamming attacks from the degradation caused by the poor quality of the channel due to normal changes in the wireless link. We point the reader to the survey in [2] for further information about jamming intrusion detection systems.

2. *Vampire Attack*: This attack exploits the fact that the majority of IoT objects have a limited battery lifetime where a malicious user misbehaves in a way that makes devices consume extra amounts of power so that they run out of battery earlier thereby causing a service disruption. The damage caused by this attack is usually measured by the amount of extra energy that objects consume compared to the normal case when no malicious behavior exists.

   We identify four types of vampire attacks based on the strategy used to drain power:

   (a) *Denial of Sleep*: Different data link layer protocols were proposed to reduce the power consumption of smart objects by switching them into sleep whenever they are not needed. Examples of these protocols include S-MAC and T-MAC protocols. The idea behind these protocols is to agree on a duty-cycle schedule where objects exchange control messages in order to synchronize their schedules so that they agree on transmitting signals at certain cycles while remaining asleep for the rest of the time. An adversary can now launch a denial of sleep attack which prevents objects from switching to sleep by simply sending control signals that change their duty-cycles keeping them active for longer durations. The adversary can still succeed in launching this attack even if the control messages that synchronize the duty-cycles of the objects are encrypted. When the control messages are encrypted, the adversary can capture one of those encrypted control messages and replay it (resend it) at a later point of time causing the nodes to change their synchronization and their schedules. The adversary needs in that case to use traffic analysis techniques that rely, for example, on the length of the packets and the rate at which packets are exchanged in order to distinguish the control messages from the data messages that the nodes exchange since the content that packets carry is hidden by encryption.

   (b) *Flooding Attack*: The adversary can flood the neighboring nodes with dummy packets and request them to deliver those packets to the fog device, where devices waste energy receiving and transmitting those dummy packets.

   (c) *Carrousel Attack*: This attack targets the network layer in the OSI stack and can be launched if the routing protocol supports *source routing*, where the object generating the packets can specify the whole routing path of the packets it wishes to send to the fog device. The adversary in that case specifies routing paths that include loops where the same packet gets routed back and fourth among the other objects wasting their power. Figure 8.8 illustrates this attack.

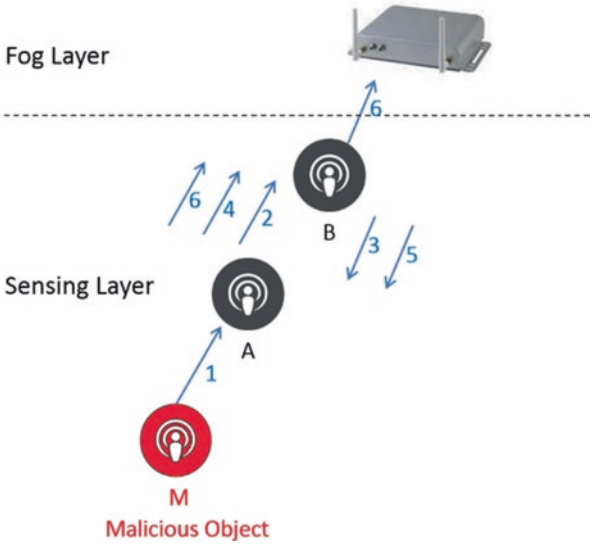

Fig. 8.8 Illustration of the carrousel attack where the numbered arrows show the path specified by the malicious objects that the packets generated by the malicious object follow

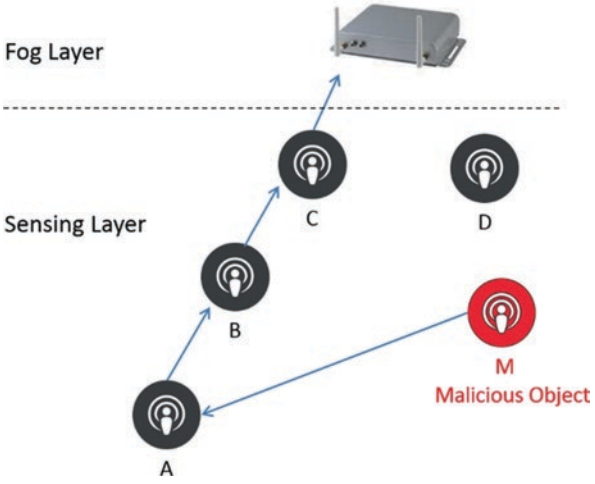

Fig. 8.9 Illustration of the stretch attack

(d) *Stretch Attack*: This attack also targets the network layer in the OSI stack. If the routing protocol supports source routing, then a malicious object can send the packets that it is supposed to report to the fog device through very long paths rather than the direct and short ones as illustrated in Fig. 8.8. Even if source routing is not supported, the attacker can select a next hop that does not have the shortest path to the fog device in order to increase the power consumption of the objects that will be responsible to deliver those packets (Fig. 8.9).

The adversary can further amplify the amount of wasted energy by combining flooding attack with carrousel attack and stretch attack. The adver-

sary in that case floods the neighboring objects with a large number of generated packets and specifies long paths with loops that the packet should follow in order to increase the amount of wasted power.

Denial of sleep attacks can be mitigated by encrypting the control message that arranges the schedules of the node while including a timestamp or a sequence number in the encrypted control message. This prevents the adversary from succeeding, in replaying an old control message, by checking the encrypted timestamp or the encrypted sequence number that the replayed control message is not a new message but an old one that someone replayed to cause disruption. Flooding attacks can be mitigated by limiting the rate of the packets that each object may generate. Carrousel attacks can be mitigated by making each object that is requested to forward a packet based on a route specified by the source check the specified path where packets with loops within their paths are dropped as they are most likely originating from malicious users. Finally, stretch attacks can be mitigated by disabling source routing or by making sure that the forwarded packets are making progress toward their destination and are not following long paths.

3. *Selective-Forwarding Attack*: This attack takes place in the case when the object cannot send its generated packets directly to the fog device but must rely on other objects that lie along the path toward the fog device to deliver those packets. A malicious object in this attack does not forward a portion of the packets that it receives from the neighboring objects. A special case of this attack is the *black-hole attack* where the attacker drops the entire set of packets that it receives from the neighboring objects. The best way to prevent packet drops from taking place for sensitive IoT applications is to increase the transmission capability of the objects so that they can reach the fog device directly without the need for help from intermediate objects. Unfortunately not all IoT objects are expected to have high transmission range to reach the fog device and thus will be relying on other objects to deliver their packets, which makes them susceptible to this attack. Different solutions were proposed to mitigate the number of dropped packets. *Path redundancy* is one of those solutions, where each object forwards each generated packet to multiple neighboring objects, where multiple copies of the same packet get delivered to the fog device through different paths. This decreases the chances of not having at least a copy of each generated packet delivered to the fog device. The main limitation of this mitigation technique is that it has a high energy overhead as it increases significantly the traffic. Rather than mitigating the damage caused by those attacks, the approach in [6, 8] tries to detect malicious objects that are dropping the sent packets so that packets can be routed through different paths that avoid those objects. Detecting the presence of objects that are dropping packets along certain paths can be done by selecting certain trusted objects as checkpoints. Each time a checkpoint receives a packet, it sends an acknowledgment to the object that generated that packet. The acknowledgment includes a unique identifier for the packet that was received along with a signed hash for the acknowledgment's content. This guarantees that no other entity fabricates fake acknowledgment packets and that no other entity

can alter the content of these acknowledgments. The interested reader may refer to [7] for a complete overview on the countermeasures that can be used against selective-forwarding attacks.

4. *Sinkhole Attack*: A malicious object claims that it has the shortest path to the fog device which attracts all neighboring objects that do not have the transmission capability to reach the fog device to forward their packets to that malicious object and count on that object to deliver their packets. Now all the packets that are originating from the neighboring nodes pass by this malicious node. This gives the malicious node the ability to look at the content of all the forwarded packets if data is sent with no encryption. Furthermore, the malicious object can drop some or all of the received packets as we explained previously in the selective-forwarding attack. Figure 8.10 illustrates how the network topology changes before and after this attack. Techniques to detect and isolate the malicious objects were proposed and are based on the idea of collecting information from the different objects where each object reports the neighboring objects along with the distance to reach those objects. A centralized intrusion detection system is then used to rely on the reported information to identify objects that are potentially providing misleading information. Detecting such attack becomes harder when multiple malicious nodes collude to hide each other.

Finally, Table 8.2 summarizes all security attacks in the sensing domain that were discussed in this section. The second column of the table shows what layer in the OSI stacks the attack targets, whereas the third, fourth, and fifth columns describe, respectively, the vulnerability reason, the security requirement that the attack breaks, and the defensive countermeasures against each attack.

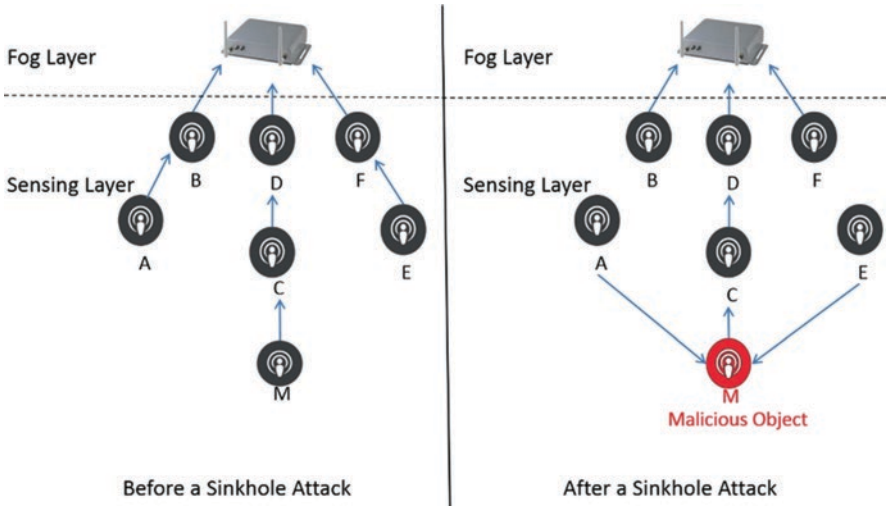

Fig. 8.10 Network topology before and after a sinkhole attack. The malicious object M claims that it has a shorter route to reach the fog device which attracts the neighboring objects A and E to rely on M to deliver their packets

Table 8.2 Summary of the security attacks targeting the sensing domain

Attack	Target OSI layer	Vulnerability reason	Security violation	Countermeasures
Jamming attack	Physical Data link	Shared wireless channel	Availability	Frequency hopping Spread spectrum Directional antennas Jamming detection techniques
Vampire attack	Data link Network	Limited battery lifetime	Availability Freshness	Rate limitation Drop packets with a source route that contains a loop Monitor whether or not the forwarded packets are making progress toward their destination
Selective-forwarding attack	Network	Limited transmission capability	Availability	Increase transmission range Path redundancy Choose certain intermediate objects as checkpoints to acknowledge received packets
Sinkhole attack	Network	Limited transmission capability	Confidentiality Availability	Analyze the collected routing information from multiple objects

8.8 Securing IoT Devices

In this section we will provide several examples of IoT devices being used to launch security attacks (Sect. 8.8.1), in addition to two solutions that attempt to secure IoT devices, namely MUD (Sect. 8.8.2) and DICE (Sect. 8.8.3).

8.8.1 IoT Devices Gone Rogue

With the increase of practical deployments, IoT devices have proven to be easy targets for hackers who turn compromised devices into active actors to carry out their attacks on networked IT infrastructure. This is especially true in the context of distributed denial of service (DDoS) attacks. Insecure IoT devices represent a growing pool of compute and communications resources that is open to misuse. These devices can be hijacked to spread malware, recruited to form botnets that may attack other Internet users, and even can be used to attack critical national infrastructure, or the structural functions of the Internet itself.

There are multiple recent examples of IoT devices being used as attack vectors. We will highlight some of them next.

8.8.1.1 Botnets

A botnet is a typically large collection of networked computers (bots) that are under remote control from some malicious third party over the Internet. Usually, these computers would have been compromised by an outside attacker who controls aspects of their functionality without the owners' consent or knowledge.

Because there are many such computers in a typical botnet, the attacker has access to a quasi supercomputer that can be employed for malicious purposes. Furthermore, since the bots are distributed geographically and organizationally over the Internet, the quasi supercomputer can be difficult to deter. The first botnet was developed in 2001 to send spam, and that is still a common use. Another common use for botnets is for DDoS attacks, in which a target server is constantly bombarded with network traffic until it is overwhelmed beyond its capacity and forced to go offline.

In 2016, a DDoS attack rendered much of the Internet inaccessible on the US East coast, and the attack was perpetrated by the Mirai botnet. Mirai took advantage of insecure IoT devices in a simple but clever way: It scanned large blocks of the Internet for open Telnet ports, then attempted to log in using username/password combinations that are frequently used defaults for these devices and never changed. With this simple approach, it was able to recruit an army of compromised closed-circuit TV cameras and routers, ready for launching a DDoS attack.

The reason why the botnet was so effective was due to the fact that it leveraged a large number of IoT devices which often include an embedded stripped-down Linux operating system. These devices had no built-in ability to be patched remotely and were in physically remote or inaccessible locations.

8.8.1.2 Webcams

Webcams are often marketed as consumer products for baby monitoring or as security devices. In one instance, a webcam manufacturer had faulty software on their products that allowed anyone with knowledge of the webcam's IP address to view the camera's video feed, and sometimes listen in through the embedded microphones. Another manufacturer's product was susceptible to remote code-injection attack, which allowed a malicious user to get administrative access to the camera, thereby placing the user at a risk of being spied upon. The remote execution flaw not only allows an attacker to set their own custom password to access the device, but also to add new users with administrative access to the interface, download malicious firmware or reconfigure the product as they please.

8.8.1.3 Casino Fish Tank

Security firm Darktrace published a report where it revealed that an unnamed casino in North America was hacked through an Internet-connected fish tank. That connection allowed the tank to be remotely monitored, automatically adjust temperature and salinity, and automate feedings. In this incident, the vulnerable smart tank was used as an easy backdoor into the casino's network. Once the attackers gained access to the tank, they scanned the casino's network for other vulnerabilities and moved laterally to other places in the network where they were able to steal 10 gigabytes of private data from the casino. The tank's communication patterns with the casino's network appeared normal enough. However, the data that it was pumping through to the Internet was highly suspect. It was the only tank system that

transmitted data to a remote server in Finland, which it was in communications with. It also did so by employing protocols that are normally used for streaming audio or video.

8.8.1.4 Cardiac Devices

Cardiac devices, such as pacemakers and defibrillators, are used to monitor and control patients' heart functions and prevent heart attacks. In 2017, the FDA announced that St Jude's Medical implantable cardiac devices had security vulnerabilities that would enable an attacker to access these devices, where they could deplete the battery or administer incorrect pacing or shocks. The vulnerabilities were in the transmitter that reads the device's data and remotely shares it with physicians.

8.8.1.5 Vehicles

In 2015, Charlie Miller and Chris Valasek, two security researchers, exposed the security vulnerabilities in automobiles by hacking into cars remotely, controlling the cars' various functions from the radio volume to the brakes. They did so by leveraging day-zero exploits that give attackers wireless access to the car via the Internet. This was done by sending commands through the vehicle's entertainment system to its dashboard functions, steering, brakes, and transmission, all remotely from their laptops. The entertainment system served as an excellent entry point, because automakers are increasingly enabling the linking of these systems to the Internet. From that entry point, Miller and Valasek's attack pivots to an adjacent chip in the car's head unit (the hardware for its entertainment system), silently rewriting the chip's firmware to plant their code. That rewritten firmware is capable of sending commands through the car's internal computer network, known as a CAN bus, to its physical components like the engine and wheels.

Proper identification of connected devices is the first step when securing any network. With IoT, the asset inventory problem is compounded due to the sheer scale of "things," and there is a key requirement to efficiently and unambiguously identify connected devices for onboarding and ongoing management. With the ongoing rapid growth in the number of IoT devices, malicious actors view these devices as a soft attack surface from where to launch their attacks onto any other target in the network. As such, it is critical to provide mechanisms and capabilities for securing these devices. Two such mechanisms are MUD and DICE, which will be covered in detail next.

8.8.2 MUD

Manufacturer Usage Descriptor (MUD) is an embedded software standard defined by the IETF (RFC 8520) to help reduce the vulnerability surface of IoT devices by employing network policy (whitelisting approach). It aims to reduce the scope of malware injection and hijacking of over-the-air firmware updates. It also addresses the scenario of devices that are no longer being actively maintained by their original manufacturer.

MUD enables IoT device manufacturers to advertise formal device specifications, including the intended communication patterns for a given device when connected to the network. The network can then leverage this advertised intent, or profile, to formulate a tailored and context-specific access control policy, to guarantee that the device communicates only within the specified parameters. This way the network behavior of the device, in any operating environment, can be locked down and verified rigorously. In this context, MUD becomes the delegated identifier and authoritative enforcer of policy for IoT devices on the network. MUD works by enabling networks to automatically permit each IoT device to send and receive only the traffic it requires to perform as intended while blocking unauthorized communication with the device.

The MUD solution consists of three key components, as shown in Fig. 8.11.

- A unique identifier, in the form of a Universal Resource Locator (URL), that an IoT device advertises when it connects to the network.
- An Internet hosted profile file that this URL points to. This file contains an abstracted policy that describes the level of communication access which the IoT device needs to perform its intended functionality.
- A core process that receives the URL from the IoT Device, retrieves the profile file from the MUD File Server, and establishes the appropriate access control policies in the network to restrict the communication patterns for that IoT device.

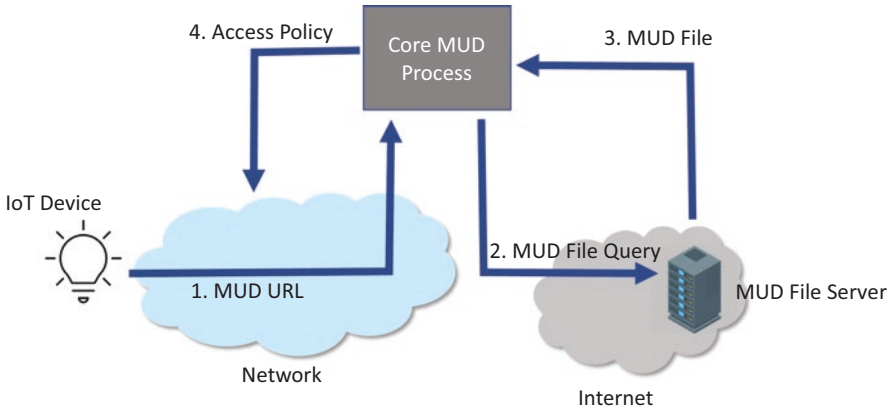

Fig. 8.11 MUD architecture

MUD leverages mechanisms that have existed in network infrastructure, including routers and switches, for over a decade. In what follows, we will describe the MUD workflow and associated mechanisms that can be used in more details.

1. The IoT device informs the network of the MUD URL using any one of the following existing protocols: DHCP, LLDP, or via a certificate in an IEEE 802.1X exchange. Once the device has communicated the URL to the network, its task in supporting MUD is done. The goal is to keep the device prerequisites as simple as possible for IoT device manufacturers.
2. The URL is received from the network by a Core MUD Process. This module may reside in one of many potential systems, depending on the nature of the network infrastructure. For instance, in an enterprise network, it may be part of the Policy (e.g., AAA) server. In a home network, it may be provided by the Internet service provider (ISP) or by the customer premise equipment (CPE) vendor. In a mobile service provider network, it might be part of an operational support system (OSS).
3. The Core MUD Process resolves the MUD URL and retrieves the profile file from the MUD File Server. This file is a declaration of intent that specifies what access the device is intended to have in the form of an abstract policy. The rationale being that an IoT device may be designed to communicate with a single or small number of controllers or with similar Things, or that for a given service, it should or should not have access to the local network.
4. The Core MUD Process translates these abstract intent definitions into a context-specific access control policy that the local network infrastructure can consume. How that translation occurs will vary depending on the network deployment. Some networks may use Access Control Lists (ACLs). Other networks may rely on segmentation using VLANs or VNIs, while others may use service groups or some other access control mechanism.
5. An administrator may then approve, reject, or modify the policy, based on deployment specifics. This policy may be merged with other policies, for instance, to take into account the user of the device or the device's deployment location.
6. The Core MUD Process pushes the merged policy to the associated systems of the network infrastructure (for example, switches, routers, etc.). This can be achieved using some configuration protocol such as NETCONF, Radius, or any alternative mechanism.

MUD provides a clear value proposition to device owners, network administrators and IoT device manufacturers. First, for device owners, it limits the impact and extent of exploitation of any security vulnerability that is potentially discovered in their IoT devices. For network administrators, MUD provides them with better visibility of the types of Things connected to the network and with the type of policies that they require. This helps them with better inventory management, risk assessment, and remediation. Finally, for device manufacturers MUD alleviates any support, financial liability, or brand damage that may arise due to compromised devices.

8.8.3 DICE

Device Identifier Composition Engine (DICE) is a collection of hardware and software mechanisms for cryptographic IoT device identity, attestation, and data encryption. DICE is an industry standard created by the Trusted Computing Group (TCG).

IoT devices that perform encryption use a private key called a Unique Device Secret (UDS) in order to secure their operation. It is possible for an attacker to leak this key by compromising the code running on the chip. Having access to the private key can enable the attacker to impersonate the device and even to replace its firmware. Therefore, it is paramount to prevent the disclosure of the UDS. The key to DICE is its ability to break up the boot process for any device into layers and to combine unique secrets and a measure of integrity for each of these layers. This way, if malware is present at any stage of the boot process, the device is automatically re-keyed and secrets protected by the legitimate keys remain safe.

DICE implements three measures to secure the UDS:

- **Power-on Latch**: The power-on latch locks read access to the UDS before early boot-code transfers control to subsequent execution layers.
- **Cryptographic One-way Functions**: A cryptographic one-way function computes a hash of the UDS to store in RAM so that in the event of RAM disclosure by compromised code, the original UDS is safe.
- **Tying Key Derivation to Software Identity**: To prevent compromise of the device by attempts to modify the early boot-code, the cryptographic one-way function uses a measurement of the boot code as input together with the UDS. The function outputs a key called the Compound Device Identifier (CDI) taking both the UDS and early boot code hash as input (optionally taking the hardware state and configuration as input as well). This process ensures that modification of early boot code generates a new key so that the UDS is secure.

The reason for tying the CDI derivation to the code that is booting on the device is to guarantee that a firmware update automatically results in the device being re-keyed. This behavior is desirable to address two security problems, specifically:

1. If an attacker changes the code that boots on the device with the intent of stealing keys, the attacking program (with a different hash) ends up obtaining a different key than the original authorized program.
2. If authorized code contains a security vulnerability that leads to CDI compromise, then the device must be re-keyed after patching. The CDI derivation function ensures that patching the vulnerable firmware automatically results in a new CDI being computed.

DICE introduces a simple security approach that does not increase the silicon requirements for IoT devices. It targets constrained devices where traditional Trusted Platform Modules (TPM) may be unfeasible due to limitations related to

cost, power, physical space, etc. As such, it is possible to implement it in the tiniest microcontrollers.

DICE is predicated upon a hardware root of trust for measurement. It works by organizing the boot into layers and creating secrets unique to each layer and configuration based on UDS (refer to Fig. 8.12). If a different code or configuration is loaded, at any point in the boot chain, the secrets will be different. Each software layer keeps the secret it receives completely confidential. If a vulnerability exists and a secret is leaked, it patches the code automatically and creates a new secret, effectively re-keying the device. In other words, when malware is present, the device is automatically re-keyed and secrets are protected.

DICE provides strong device identity, attestation of device firmware and security policy, and safe deployment and verification of software updates. The latter are often a source of malware and other attacks. Another key benefit for device manufacturers is that they are no longer required to maintain databases of unique secrets.

8.9 Summary and Future Directions

This chapter analyzed IoT from a security and privacy perspectives. Ignoring security and privacy will limit the applicability of IoT and will have serious results on the different aspects of our lives given that all the physical objects in our surrounding will be connected to the network. In this chapter, the IoT security challenges and IoT security requirements were identified. A three-domain IoT architecture was considered in our analysis where we analyzed the attacks targeting the cloud domain, the fog domain, and the sensing domain. Our analysis describes how the different attacks at each domain work and what defensive countermeasures can be applied to prevent, detect, or mitigate those attacks. We hope that the research and industry communities will pay attention to the discussed security threats and will apply appropriate countermeasures to address those issues. We also hope that security and privacy will be considered at the early design stage of IoT in order to avoid the common pitfall of considering security as an afterthought.

We end this chapter by providing some future directions for IoT security and privacy:

- *Fog Domain Security*: The fog domain is a new domain that was introduced to bring the computing capabilities to the edge of the network. We believe that further attention should be paid to this domain as it has not received enough

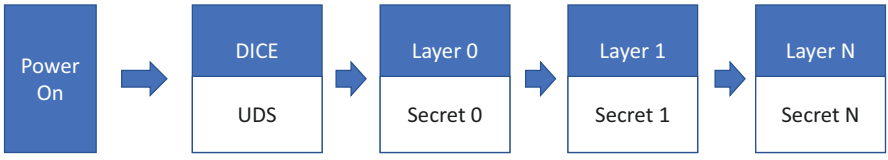

Fig. 8.12 DICE architecture

attention from the academia and the industry. The focus should be on identifying threat models related to the fog domain and also on finding efficient solutions that can run on the fog devices that are available in the market.

- *Collaborative Defense*: We identified while surveying the related work that what the literature on IoT security lacks is a collaborative solution where the different domains (cloud, fog, and sensing) interact with each other to stop or mitigate a certain attack. We believe that an interdomain-defensive solution will be way more effective than applying countermeasures at each domain separately, where the different domains can interact and collaborate in order to stop any ongoing malicious activity.
- *Lightweight Cryptography*: This is a highly important topic that has gained a significant attention recently and is anticipated to be very important for the future of IoT where the objective is to find efficient cryptographic techniques that can replace the traditional computationally expensive ones while achieving an acceptable level of security.
- *Lightweight Network Security Protocols*: Not only the cryptographic computations must have lower overhead but also the network security protocols that are used for communication. Many efforts are being paid by the research and industry communities to find cross-domain-optimized security protocols that achieve the necessary security protection while maintaining a low overhead.
- *Digital Forensics*: Although tracking the location of smart objects is considered a privacy violation, it also has some useful cases. Consider, for example, the case where police rely on tracking the smart objects that are carried by a missing person in order to identify the missing person's location. Digital forensics in the IoT era will play an important role in solving the different forensic cases as they will all become technology related. This area is also expected to receive further attention in the future where different techniques can be used to extract knowledge from the smart objects.

Problems and Exercises

1. The authors have broken IoT security challenges into seven areas. Name them. Why big data is an issue for IoT?
2. What techniques can be applied to prevent cross-VM data leakage? Explain how the hard isolation technique can be achieved.
3. What are some of the typical uses of VM migration in cloud data centers? What are the two types of attacks that are related to VM migration?
4. Who is the entity that initiates insider attacks, and how can homomorphic encryption be used to prevent such attacks?
5. What are the three key differences that distinguish fog devices from cloud servers? Provide a brief explanation of each difference.
6. Which provides more protection against security attacks: container-based virtualization or full virtualization? Why?
7. What are the two connection approaches that the smart objects may use to communicate with the fog device? Which approach is more secure and can this approach always be used?

8. What are the four strategies that a jammer may follow in order to launch a jamming attack? Which strategy is suitable when the jammer have limited energy budget?
9. What are vampire attacks? Name their types.
10. What is network high availability? What is network redundancy? How are they related?
11. Chapter 3 discusses three different ways to obtain information for IoT "things": sensors, RFID, and video tracking. In a table, compare the security for the three technologies.
12. What is limiting cache switching rate? How can it be accomplished? Explain how it works.

References

1. D. Willis, A. Dasgupta, S. Banerjee, Paradrop: a multi-tenant platform for dynamically installed third party services on home gateways, in *SIGCOMM workshop on distributed cloud computing*, (ACM, New York, NY, 2014)
2. W. Xu et al., Jamming sensor networks: attack and defense strategies. Network IEEE **20**(3), 41–47 (2006)
3. W. Ye, J. Heidemann, D. Estrin, Medium access control with coordinated adaptive sleeping for wireless sensor networks. Networking, IEEE/ACM Transactions **12**(3), 493–506 (2004)
4. T. Van Dam, and K. Langendoen, An adaptive energy-efficient MAC protocol for wireless sensor networks. in *Proceedings of the 1st international conference on Embedded networked sensor systems,* ACM, 2003
5. K.P. Dyer, et al., Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. in *Security and Privacy (SP), 2012 IEEE Symposium,* IEEE, 2012
6. J. Park, et al., An Energy-Efficient Selective Forwarding Attack Detection Scheme Using Lazy Detection in Wireless Sensor Networks. in *Ubiquitous Information Technologies and Applications*, (Springer, The Netherlands, 2013), pp. 157–164
7. L.K. Bysani, and A.K. Turuk, A survey on selective forwarding attack in wireless sensor networks. in *Devices and Communications (ICDeCom), 2011 International Conference,* IEEE, 2011
8. B. Xiao, B. Yu, C. Gao, CHEMAS: Identify suspect nodes in selective forwarding attacks. J. Parallel Distrib. Comput. **67**(11), 1218–1230 (2007)
9. P. Thulasiraman, S. Ramasubramanian, and M. Krunz, Disjoint multipath routing to two distinct drains in a multi-drain sensor network. in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications*, IEEE, 2007
10. H.-M. Sun, C.-M. Chen, and Y.-C. Hsiao, An efficient countermeasure to the selective forwarding attack in wireless sensor networks. in *TENCON 2007–2007 IEEE Region 10 Conference*, IEEE, 2007
11. A. Grau, Can you trust your fridge? Spectrum, IEEE **52**(3), 50–56 (2015)
12. C. Li, A. Raghunathan, and N. K. Jha, Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. in *e-Health Networking Applications and Services (Healthcom), 2011 13th IEEE International Conference,* IEEE, 2011
13. D. Evans, *The internet of things how the next evolution of the internet is changing everything*. Technical report, CISCO IBSG, 2011

14. R. Thomas, et al., Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. in *Proceedings of the 16th ACM conference on Computer and communications security*, ACM, 2009
15. M. Dabbagh, B. Hamdaoui, M. Guizai and A. Rayes, Release-time aware VM placement. in *Globecom Workshops (GC Wkshps)*, (2014), pp. 122–126
16. M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment. Network, IEEE **29**(2), 56–61 (2015)
17. M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, Efficient datacenter resource utilization through cloud resource overcommitment, in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2015, pp. 330–335
18. R. Boutaba, Q. Zhang, and M. Zhani, Virtual Machine Migration in Cloud Computing Environments: Benefits, Challenges, and Approaches. in *Communication Infrastructures for Cloud Computing,* ed. by H. Mouftah and B. Kantarci (IGI-Global*,* Hershey PA, 2013), pp. 383–408
19. D. Perez-Botero, *A Brief Tutorial on Live Virtual Machine Migration from a Security Perspective*, University of Princeton*,* Princeton, 2011
20. W. Zhang, et al., Performance degradation-aware virtual machine live migration in virtualized servers. in *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT),* 2012
21. V. Venkatanathan, T. Ristenpart, and M. Swift, *Scheduler-based defenses against cross-VM side-channels*. Usenix Security, (2014)
22. T. Kim, M. Peinado, and G. Mainar-Ruiz, Stealthmem: System-level protection against cache-based side channel attacks in the cloud. in *Proceedings of USENIX Conference on Security Symposium, Security'12. USENIX Association*, 2012
23. H. Raj, R. Nathuji, A. Singh, and P. England, Resource management for isolation enhanced cloud services. in *Proceedings of the 2009 ACM workshop on Cloud computing security*, ACM, 2009, pp. 77–84
24. Y. Zhang and M. K. Reiter, Duppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. in *Proceedings of the 2013 ACM SIGSAC Conference on Computer; Communications Security*, CCS '13. ACM, 2013
25. P. Li, D. Gao, and M. K. Reiter, Mitigating access driven timing channels in clouds using stopwatch. in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013, pp. 1–12
26. R. Martin, J. Demme, and S. Sethumadhavan, Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate sidechannel attacks, in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 2012
27. F. Zhou et al., Scheduler vulnerabilities and coordinated attacks in cloud computing. in *10th IEEE International Symposium on Network Computing and Applications (NCA),* 2011
28. K. Panagiotis, and M. Bora, Cloud security tactics: Virtualization and the VMM. in *Application of information and communication technologies (AICT), 2012 6th International Conference.* IEEE, 2012
29. F. Zhang et al., CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles,* ACM, 2011
30. T. Taleb, A. Ksentini, Follow me cloud: interworking federated clouds and distributed mobile networks. IEEE Network **27**, 12 (2013)
31. E. Damiani et al., A reputation-based approach for choosing reliable resources in peer-to-peer networks. in *Proceedings of the 9th ACM conference on computer and communications security*. ACM, 2002
32. W. Itani et al., Reputation as a Service: A System for Ranking Service Providers in Cloud Systems. in *Security, Privacy and Trust in Cloud Systems*. (Springer, Berlin Heidelberg, 2014). pp. 375–406
33. J. Sahoo, M. Subasish, and L. Radha, Virtualization: A survey on concepts, taxonomy and associated security issues. in *Second International Conference on Computer and Network Technology (ICCNT)*, 2010

34. S.Yi, Q. Zhengrui, and L. Qun, Security and privacy issues of fog computing: A survey. in *Wireless Algorithms, Systems, and Applications*, (Springer International Publishing, 2015), pp. 685–695
35. E. Oriwoh, J. David, E. Gregory, and S. Paul, Internet of things forensics: Challenges and approaches. in *9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom),* IEEE, 2013, pp. 608–615
36. Z. Brakerski, V. Vinod, Efficient fully homomorphic encryption from (standard) LWE. SIAM J. Comput. **43**(2), 831–871 (2014)
37. E. Lauter, Practical applications of homomorphic encryption. in *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, ACM, 2012
38. C. Hennebert, D. Jessye, Security protocols and privacy issues into 6lowpan stack: A synthesis. Internet of Things Journal IEEE **1**(5), 384–398 (2014)
39. Daily Tech Blogs On Line, http://www.dailytech.com/Five+Charged+in+Largest+Financial+Hacking+Case+in+US+History/article32050.htm
40. M. Miller, Car hacking' just got real: In experiment, hackers disable SUV on busy highway (The Washington Post, 2015), online: http://www.washingtonpost.com/news/morning-mix/wp/2015/07/22/car-hacking-just-got-real-hackers-disable-suv-on-busy-highway/
41. 2015 Data Breach Investigation Report, Verizon Incorporation (2015)
42. M. Dabbagh et al., Fast dynamic internet mapping. Futur. Gener. Comput. Syst. **39**, 55–66 (2014)
43. Forrester, Security: The Vital Element of the Internet of Things, 2015, online: http://www.cisco.com/web/solutions/trends/iot/vital-element.pdf
44. F. Adib and D. Katabi, See through walls with WiFi!, vol. 43. (ACM, 2013)
45. S. Kumar, S. Gil, D. Katabi, and D. Rus, Accurate indoor localization with zero start-up cost, in *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, ACM, 2014, pp. 483–494
46. G. Wang, Y. Zou, Z. Zhou, K. Wu, and L. Ni, We can hear you with Wi-Fi!, in *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, ACM, 2014, pp. 593–604
47. Y. Qiao, O. Zhang, W. Zhou, K. Srinivasan, and A. Arora, PhyCloak: Obfuscating sensing from communication signals, in *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016
48. T. Yu, et al., Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things, *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, 2015
49. M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, Software-defined networking security: pros and cons. IEEE Commun. Mag. **53**, 73 (2015)