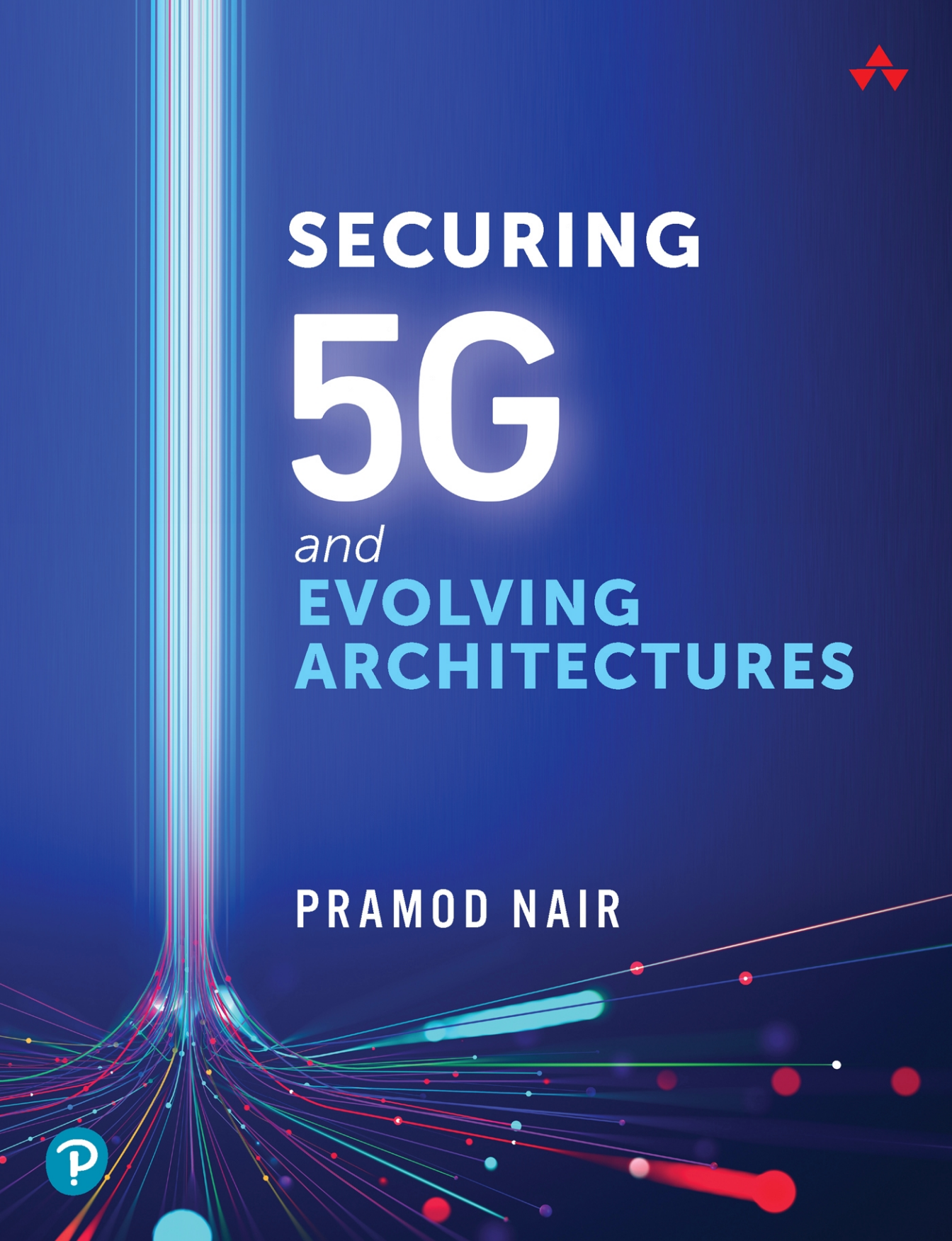
SECURING
5G
and
EVOLVING
ARCHITECTURES
PRAMOD NAIR

Securing 5G and Evolving Architectures

Pramod Nair

Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town • Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City • São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

and understand who needed what access, what network components needed what kinds of services (DNS and so on), and what traffic needed to exit the infrastructure. The access to specific applications was then planned and configured in the Identity and Access Management (IAM) layer. This method of having manual segmentation/micro-segmentation is inefficient and not scalable for managing tens of thousands of workloads. The solution is to provide micro-segmentation based on rules that can be maintained using behavior analysis of the application. The method for providing automated/semi-automated micro-segmentation is located in the "Securing Virtualized Deployments in 5G MEC" section in this chapter.

Another area of importance in 5G MEC virtual deployments is vulnerability assessments. 5G allows open source components to be used to create an MEC application leading to a polyglot microservice architecture. *Polyglot* generally means that an application developer can use the program language of their choice to create an application.

Figure 5-19 shows the polyglot nature of a microservice-based 5G network function or 5G MEC application.

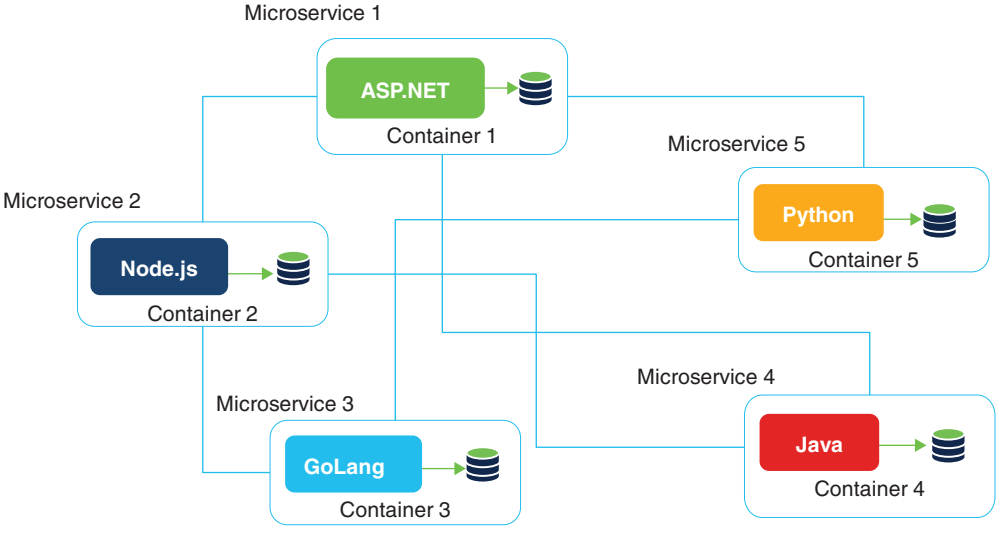

FIGURE 5-19 Polyglot Nature of Microservices

As shown in Figure 5-19, the microservice is polyglot in nature, which allows MEC application developers to pick a programming language of their choice in order to build products in more efficient ways. In 5G, virtualization of the RAN and 5GC network functions is cloud native based and can be deployed on the cloud. This allows you to use different container deployment models, such as containers on virtual machine, containers on bare metal, or deployed on multiple public clouds. The 4G subsystems might be still deployed on virtual machines (VMs) due to the industry shortcomings on 4G components, but for 5G you might actually skip the container on virtual machine deployment and have it deployed on bare metal or on the public cloud due to the early development of 5G network

functions based on containers. For the MEC applications, there have been quite good developments in the containerized space, and there are already applications available today using open source programs that can be deployed in any infrastructure, including cloud. You should be very careful before choosing to deploy any open-source components without validating them for vulnerabilities.

One of the key issues is the lack of visibility in checking if any of the programs has a vulnerability that needs to be patched, as shown in Figure 5-20. If it's only one microservice that needs attention, it can be checked and updated manually. But in 5G, when we have tens of thousands of MEC applications, it becomes very difficult to conduct a vulnerability assessment and compliancy check.

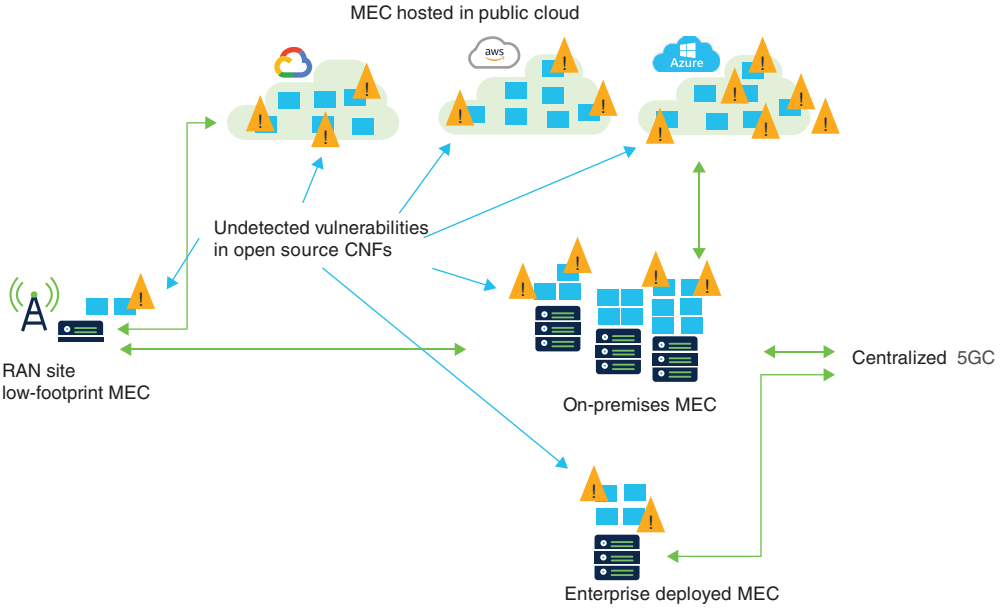

FIGURE 5-20 Vulnerabilities in Open Source Cloud-Native Deployments

As shown in Figure 5-20, it becomes really tricky if you need to have a visibility into the vulnerabilities of all open source programs across the entire stack of MEC deployments, whether deployed in a multi-stack cloud, RAN site low-footprint MEC, on-premises MEC, or enterprise-deployed MEC.

Attackers are constantly innovating methods to exploit the container ecosystem in order to exploit the vulnerabilities/improper configurations to launch attacks. Docker is one of the most widely used platforms for managing containers. Over time, it became the de facto standard for development and deployment of web and cloud applications; however, anybody who has worked with Docker quickly realizes that the learning curve is quite steep. Therefore, Docker installations can be easily misconfigured and the Docker daemon exposed to external networks with a minimal level of security.

Figure 5-21 shows crypto-mining malware spreading through the system via an attacker exploiting an incorrectly configured Docker API for third-party MEC applications.

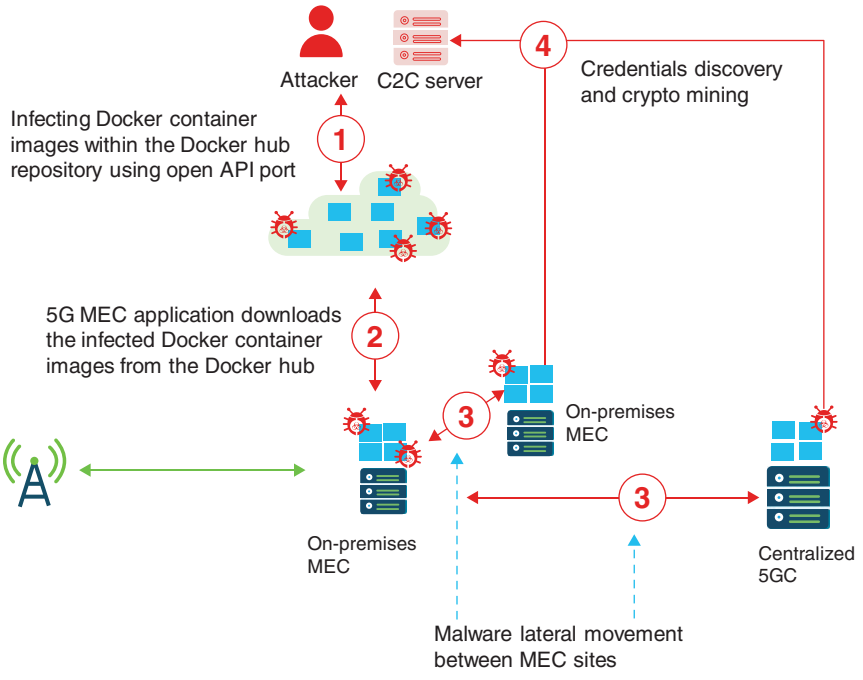

FIGURE 5-21 Attacker Exploiting an Incorrectly Configured Docker API

As shown in Figure 5-21, the main threat vector here is an incorrect configuration of the Docker API, which is exploited by the attacker to gain access to the system that hosts the third-party 5G MEC application. The attacker then laterally travels into another MEC site, a centralized 5GC site, infecting other systems and using it for crypto-jacking (stealing credentials and other sensitive data). For the sample attack shown in the Figure 5-21, the steps are as follows:

Step 1. The attackers scan the Internet for Docker containers implemented with an unprotected or incorrectly configured API. In case of Docker, the Docker daemon listens for API requests and also manages Docker objects, such as images, containers, networks and volumes.

Figure 5-22 shows the Docker architecture and the API the attackers are trying to exploit.

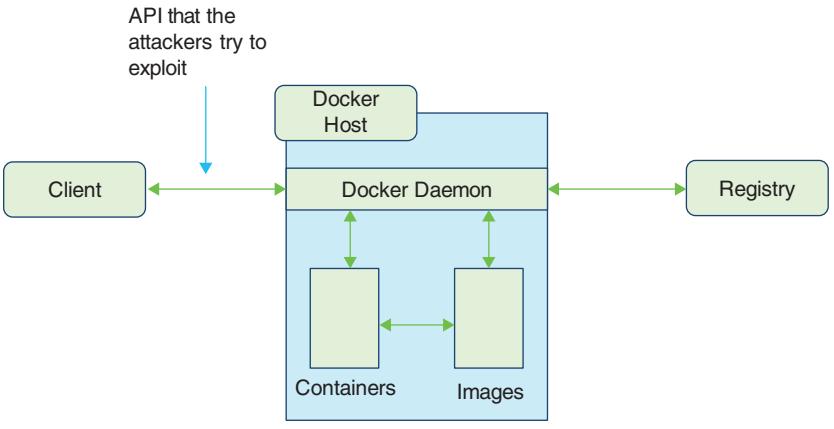

FIGURE 5-22 Docker Architecture and the API That Attackers Try to Exploit

Step 2. This infected image is pulled and run as containers on the compromised host. This can be easily accomplished if there is malware already present in the MEC host causing a privilege escalation attack.

Step 3. The infected code within the container can also cause the malware to spread deeper within the local and remote networks. The main module could also attempt to spread to other hosts by stealing the client-side certificates and connecting to them without requiring a password. The other modules within the infected container image can also include scripts for actions such as terminating security services such as anti-malware solutions and removing a competitor's botnets. It can also be programmed to protect itself and hide the mining process from process enumeration tools.

Step 4. Once ready, the scripts within the infected container image can now extract sensitive information, such as number of CPUs and user credentials, to the C2 server, which can now be used for calculating the mining capabilities. The crypto-jacking worm can now be used for mining. The infected container image can query the C2 server for other hosts it can infect.

Among other threats linked to virtualized deployments, the side-channel threat vector is quite important because it can impact any virtual function sharing CPU memory. There are many variants of side-channel attacks, such as hyper-jacking, Spectre, Meltdown, Cross-VM cache side-channel attacks, and so on. These are generally aimed at leaking the crypto keys, which are then used for eavesdropping or exfiltrating the data out of the infrastructure. Cross-VM cache side-channel attacks are well-planned sophisticated attacks focused on a particular organization. The side-channel attack method chosen by the attacker depends on many factors of the target network implementation, such as installed MEC infrastructure components, the vendor and model of chipsets being used in the MEC data center, the type of network devices present in the targeted MEC network, and so on.

Figure 5-23 shows an example of data exfiltration due to CPU processes used in virtualization.

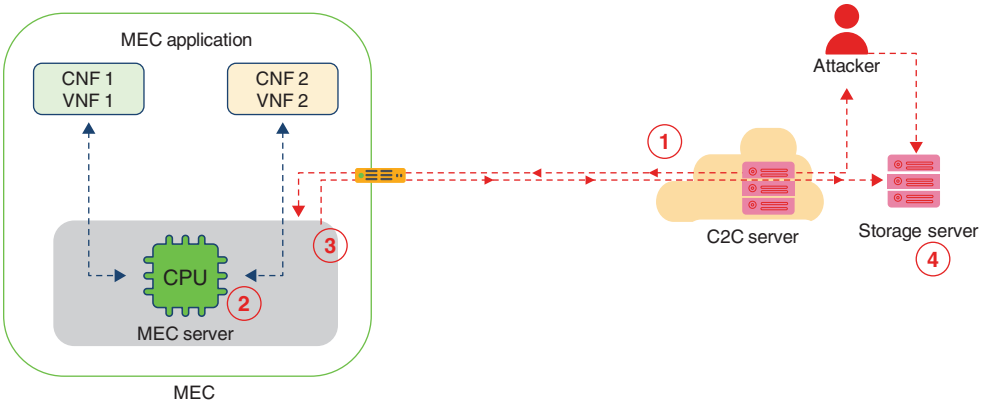

FIGURE 5-23 Data Exfiltration Due to Side-Channel Attack

As shown in Figure 5-23, which shows the attack vector aimed at the virtualization infrastructure's firmware, the attacker can use the following steps:

Step 1. The attacker uses side-channel attack vectors such as a cross-VM cache side-channel attack and utilizes the underlying hardware CPU vulnerabilities.

Step 2. The attacker's side-channel attack vector allows one process to extract sensitive information by exploiting shared cache memory between VMs and analyzing the data, such as the relation between the software process and the power consumption of the hardware for that process.

Step 3. The data can be exfiltrated silently using existing pin holes or open ports in the firewall or by reusing the ports being used for encrypted messages, such as port 443 for TLS.

Step 4. The exfiltrated data is analyzed to understand the software process and the type of software being used, among other things. Once the software is understood, it can be used to launch a crippling attack at a later stage.

5G MEC API Vulnerabilities

5G architecture has been designed to be cloud native, as it brings elasticity, scalability, and creation of rich services in the edge. 5G also allows you to expose rich services through the cloud and RESTful APIs. APIs make it very easy to integrate between different solutions and information sharing to enable a seamless user experience. API introduction for MEC applications also brings in the expertise of web application developers in developing 5G MEC applications, but it also brings in new threat vectors.

Figure 5-24 shows an example of API threat vectors in 5G MEC deployments, where the attacker uses an injection attack causing DoS to the devices being served by the MEC applications hosted on the MEC server.

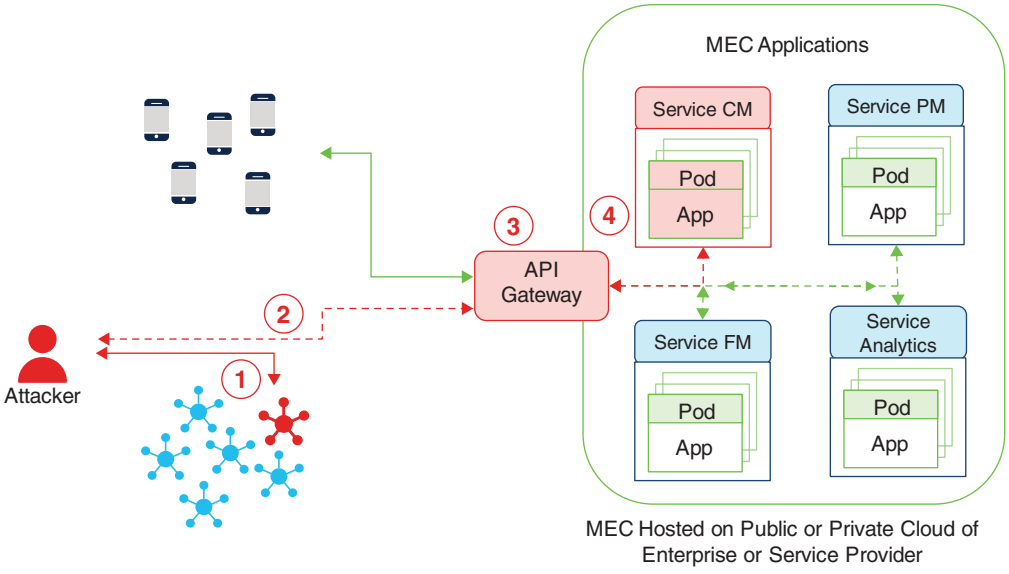

FIGURE 5-24 API Injection Attack in Applications Deployed in MEC

As shown in Figure 5-24, the attacker uses an injection attack, which occurs as follows:

Step 1. The attacker gains control of a weak IoT device and gets the credentials of the target server to be exploited.

Step 2. The attacker constructs API calls that include command injection, NoSQL, and so on, toward the CM server.

Step 3. The API gateway does not perform any checks and passes the calls to the CM server.

Step 4. The CM server blindly executes the calls, which will crash the CM server, causing DoS to all the devices being served by the CM server.

In this threat vector, the attacker is hidden completely and uses the existing device to launch an attack toward the MEC. This attack could be launched directly by the attacker using the API.

Figure 5-25 shows an example of API excessive data exposure attack.