Journal of Reliable Intelligent Environments
https://doi.org/10.1007/s40860-022-00175-4

ORIGINAL ARTICLE

Helping novice developers harness security issues in cloud-IoT systems

Fulvio Corno[1] · **Luigi De Russis**[1] · **Luca Mannella**[1]

Received: 22 October 2021 / Accepted: 6 April 2022

Abstract
The development of cloud-connected Internet of Things (IoT) systems is becoming more and more affordable, even to novice programmers, thanks to dedicated cloud platforms that already integrate the core functionality needed by an IoT system. In this context, a growing number of IoT systems are being developed and deployed on open networks, often without integrating adequate security in the process. Novice IoT programmers, in particular, tend to overlook security issues, as confirmed by a small user study. Starting from this risk, the paper analyzes the security features available in two major cloud-IoT platforms (Amazon Web Services and Microsoft Azure) and highlights those settings, tools, and practices designed to ensure more secure implementations. We observed that these platforms would reasonably address many security problems detected in the study, if only the correct features were identified and used. The paper finally contributes a set of guidelines to support novice IoT developers in avoiding the main and recurrent security issues in their projects and better exploiting cloud-IoT platforms' inherent security features.

Keywords AWS · Azure · Cloud · Cybersecurity · Guidelines · IoT · Novice programmers

1 Introduction

Internet of Things (IoT) and cloud computing are two widespread emerging technologies that are becoming very popular not only for scholars and expert developers but also for hobbyists and novice developers. Many users are now using IoT devices in their houses [1], and the cloud computing market is expanding year after year. The growth of these two fields brings to the inclusion of IoT functionalities inside the cloud platforms and the emergence of specialized features needed in IoT systems. In the past years, these platforms aided the development of IoT solutions thanks to the already integrated functionality a cloud-connected IoT application needs. Nowadays, these platforms are quite easy to use, even by novice programmers, and more affordable than a few years ago. Furthermore, cloud computing service providers allow their customers to pay only for the resources effectively consumed, facilitating the entry of many developers into this market.

Currently, according to Gartner's evaluation [2], the major cloud platforms for infrastructure and services are as follows: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Alibaba Cloud, Oracle Cloud Infrastructure (OCI), Tencent Cloud, and IBM Cloud. Furthermore, all these platforms include specific tools designed for developing IoT applications or interacting with IoT devices.

Unfortunately, such expansion comes with a price. This growing number of IoT applications are too often designed, developed, and deployed on open networks without integrating adequate privacy and security in the process [1]. For instance, one of the most known IoT failures is the *Mirai Botnet* [3]. Using hard-coded and easy-to-guess passwords in thousands of IoT devices allowed attackers to create several massive botnets able to execute powerful Distributed Denial of Service (DDoS) attacks. Another famous cybersecurity flaw in the IoT domain occurred in the implantable cardiac devices developed by St. Jude Medical[1] (now part of Abbott Laboratories). In this case, the vulnerability occurred in the transmitter (a device called Merlin@home) that reads the cardiac device's data and remotely shares it with doctors without

✉ Luca Mannella
luca.mannella@polito.it

Fulvio Corno
fulvio.corno@polito.it

Luigi De Russis
luigi.derussis@polito.it

[1] Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino 10129, Italy

[1] https://www.muddywatersresearch.com/research/stj/mw-is-short-stj/, last visited on October 6, 2021.

Published online: 13 May 2022

authentication or encryption. Indeed, an attacker could easily impersonate a transmitter unit and communicate with the cardiac device. In this way, malicious users could control the device depleting the battery or administering incorrect pacing or shocks.

However, sometimes the attack could start from the cloud platform and compromise the device. Indeed, at the beginning of last year, a new CVE related to a core component of the Kalay cloud platform for IoT devices offered by ThroughTek was created [4]. By exploiting this flaw, a malicious user could steal the credentials used by other users to access one of their devices. To proceed, the attacker should obtain the unique identifier of the device. Then, registering another device with the same identifier on the network, they overwrite the original device on Kalay servers. Once this association is established, the next victim's connection will be directed by the server to the attackers, allowing them to steal the credentials.

The burden of ensuring the security of complex IoT systems falls ultimately on the developers, and security considerations add to the complexity of this already challenging application domain.

Building a fully secure system is challenging and requires a deep understanding of all the issues involved. However, even novice programmers could reach a basic level of security, avoiding the most basic vulnerabilities, if they apply security-by-design concepts, following a clear and focused set of guidelines during their development process.

Nevertheless, designing a secure application from the beginning is not always straightforward, especially for a novice developer. However, this task could be tough even for experienced developers if they are new to a particular technology (especially in a diversified programming field like the IoT). It is challenging to avoid security issues "by design" if programmers do not feel comfortable with a specific technology. Hence, in this work, we focused on *novice IoT programmers*, by considering software developers who are not new to the programming world in general, but they had never developed a full-working and production-ready IoT system.

The goal of this paper is to help novice IoT programmers realizing more secure cloud-IoT solutions. To achieve this goal, the paper provides a set of straightforward guidelines that could be easily followed by developers new to this specific application domain, suited to the major cloud-IoT platforms. To conduct this work, we started from preliminary results presented in [5], where we exposed a use case developed in the context of a training activity. In that use case, a group of novice IoT developers created a cloud-IoT application using a major cloud platform. Initial results showed that developers often did not accurately consider some security issues in their design and implementation, thus motivating further research for providing them an adequate support. The results initially presented in [5] are briefly recalled and extended in this paper (Sect. 5), where we try to understand the security perception of this class of developers and the impact of this perception on their developed applications. Taking into account the survey's findings, this paper analyzes some relevant security features available in *Amazon Web Services* (AWS) and *Microsoft Azure*, two of the major cloud IoT platforms, to comprehend if they could compensate for novice IoT programmers' lack of security knowledge. AWS and Azure are well-known cloud platforms, with a wide industry adoption and comparable functionalities, according to both industry [2] and academic [6–8] studies. In particular, we observed that the two platforms could reasonably counter many security problems detected in the study if the developers identify and use the correct features.

As a final contribution, to support novice IoT developers in shunning the major security issues in a generic IoT architecture, we developed a set of guidelines helpful to exploit the inherent security features of cloud-IoT platforms.

This paper is structured as follows: we analyze the related works in Sect. 2. Then, Sect. 3 introduces the two cloud platforms considered in the study: AWS and Azure. We present an illustrative use case in Sect. 4, where we identify some possible attack points (Sect. 4.1). Section 5 presents the user study results expanding the work in [5] and examines the perception of the previously cited issues by our novice IoT developers. Section 6 analyzes how the previously highlighted attack points could be faced by the two cloud platforms. Then, the paper discusses in Sect. 7 the actual severity of the previously cited issues considering both the developers' perceptions and the cloud-IoT platforms' design. Building on the previous analysis, Sect. 8 proposes some guidelines to support novice IoT developers in avoiding the security issues previously discussed in their projects. In the end, Sect. 9 concludes the paper and proposes some considerations for future activities.

2 Related work

Nowadays, cloud computing is a well-known concept. The benefits and the opportunities offered by this technology are evident not only for developers but also for the general public. Nevertheless, opening such a distributed technology to many different platform users from anywhere in the world could create many security threats. For this reason, companies [9] and researchers [10,11] have analyzed the security risks and issues of cloud computing since the very beginning of this paradigm.

Even the way of thinking of novice developers is a field of research already examined by scholars. In 1989, Soloway and Spohrer published a whole book [12] to illustrate the major issues for this kind of developer.

The first research activities were mainly focused on better understanding how this class of developers faces the most complicated programming feature like, for instance, recursion [13]. More recently, Lahtien et al. [14] conducted a vast survey—including 559 students and 34 teachers from five different countries—focused on the main common problems in learning how to develop computer programs by a university student. In this study, researchers discovered that novice developers' most recurrent problem is not comprehending a computer science course's notions but learning how to apply them concretely. The outcome of this research could explain why, even if the developers involved in our survey are conscious of a few security concepts, they did not try to apply them during the development of their prototypes. Understanding the main problems of these developers and helping them improve their skills is still an active research topic. Scholars not only study novice programmers in universities, but they are also interested in understanding the main issues in a life-long context. For instance, in a recent study, Billy Javier analyzed the difficulties of life-long novice programmers and provided some suggestions to improve the courses given to them [15].

Even our research group had already conducted some research activities related to novice programmers. For two consecutive years, we studied *novice IoT programmers* of the "Ambient Intelligence" course, held in Politecnico di Torino (precisely 2014 and 2015 editions). In one of our previous studies [16], we spotlighted some of the most painful points for this category of developers. Notably, we realized that novice developers perceived their tasks as extremely difficult when they are associated with: integrating various subsystems, interaction with proprietary third-party services, and the configuration of mobile, web, or hybrid applications. Furthermore, we are still investigating their issues and studying possible solutions to help programmers create better IoT solutions using tools like a computational notebook focused on IoT technologies and physical computing [17,18].

However, it is not so common to find studies related to the *security perception* of a novice programmer. Usually, researchers analyze this kind of perception from a non-technical point of view. For instance, considering the security perception from a platform user's point of view, it is demonstrated that there is a relationship between a well-designed human–computer interface and the users' security perception [19]. Another example of this kind of study is the work of Varga et al. [20]. They published an article related to the cyber-threat perception of actors belonging to the Swedish financial sector.

Discussing the IoT domain instead, in line with the recent literature, IoT systems appear to be in a critical situation. According to the study of Kumar et al. [1], people started considerably using IoT systems in their houses. Indeed, roughly 40% of houses worldwide have at least an IoT device; meanwhile, this percentage rises to approximately 70% only in North America. In their work, Kumar et al. investigated an extensive data set of IoT devices (83M) located in a significant amount of real houses (15.5M). Thanks to this analysis, they discovered that an unexpected amount of devices still support protocols considered not secure nowadays, like the File Transfer Protocol (FTP) and Telnet. In addition to this large study, they performed an in-depth analysis of the data retrieved on a specific day. These data were retrieved from the users actively using Avast Wi-Fi inspector.[2] From the outcome of this specific analysis, they discovered that 62% of the scanned houses were afflicted by at least one known vulnerability.

In another paper [21], Kafle et al. proved a lateral privilege escalation attack on a cloud-IoT environment (specifically on Google Nest[3]). Their study demonstrated that the low security of a cloud platform is sometimes related to third-party programmers' mistakes. Indeed, even if Nest attempted to maintain its platform secure through a review process, they decided not to execute this inspection on the applications downloaded by less than 50 users. Unfortunately, applications with a low number of downloads are more likely developed by novice programmers (or explicitly created with a malicious purpose).

Even if other scholars are studying security requirements [22], best practices [23], and countermeasures to the weaknesses of IoT systems (e.g., through sharp Intrusion Detection Systems [24]), our works aims at better analyzing the architectural elements involved in a typical cloud-IoT application. Indeed, even if the best practices provided by Momenzadeh et al. [23] were verified against some cloud-IoT platforms, they are more focused on the point of view of the platforms' developers. Instead, our final goal is to provide a straightforward methodology that a novice IoT programmer (in their role as a final user of the cloud platform) could use from the very beginning to design and implement a more reliable IoT system interacting with a cloud back-end.

3 Cloud platforms overview

This section introduces the two cloud-IoT platforms analyzed in more detail in our study, that will be used to highlight the common security issues, and their solution, and to inform the definition of the guidelines presented in Sect. 8.

[2] https://support.avast.com/en-id/article/104/, last visited on January 27, 2022.

[3] https://store.google.com/category/google_nest, last visited on January 27, 2022.

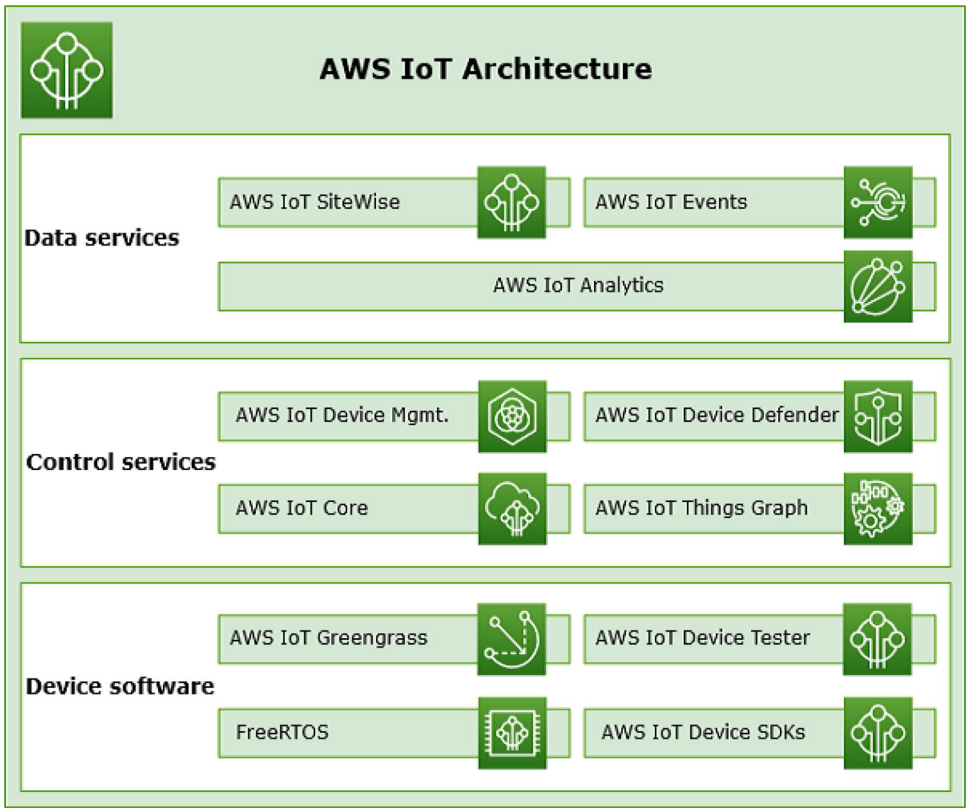

Fig. 1 AWS IoT services overview

3.1 Amazon Web Services (AWS)

Amazon Web Services (AWS) is an on-demand cloud computing platform developed and maintained by Amazon. This platform provides many basic abstract technical infrastructure and distributed computing building blocks and tools. AWS offers its cloud services according to the three main common cloud paradigms: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

On the security page of their website,[4] Amazon declares to put much effort into keeping AWS a reliable service for its customers. In their white papers, Amazon guarantees that AWS's IT infrastructure is designed and managed in alignment with security best practices and several IT security standards [25,26]. This commitment also seems to be confirmed by the work of other scholars [27].

Nevertheless, one of the main points clearly explained inside the AWS policy is the *shared responsibility model* [28]. Considering that the customers share with AWS the control over the IT environment, security could not be considered a duty of Amazon entirely, but it is a responsibility shared with any customer that uses AWS. For this reason, programmers must not underestimate their role in keeping the application developed through AWS secure. This model could be particularly tricky for novice developers. Their lack of knowledge could more easily bring them to overlook their responsibilities when creating projects with a very famous cloud platform.

To support developers, AWS IoT provides some cloud services helpful in connecting each IoT device to others and to AWS cloud services. Moreover, AWS IoT offers device software helpful to integrate these devices into AWS IoT-based solutions. Amazon divides its IoT components into the following three categories (Fig. 1): Device Software, Control Services, and Data Services. In particular, considering that in this study, we focused our attention on the interconnection of the devices to the platform, instead of on the various possible devices, we mainly considered the components belonging to the second category (Control Services).

3.2 Microsoft Azure

Microsoft Azure (initially known as Windows Azure) is a public cloud computing platform offered and maintained by Microsoft. It supports many different services both from Microsoft and third parties. Like AWS, Azure offers its cloud services in three different ways: IaaS, PaaS, and SaaS.

[4] https://aws.amazon.com/security/, last visited on January 12, 2022.

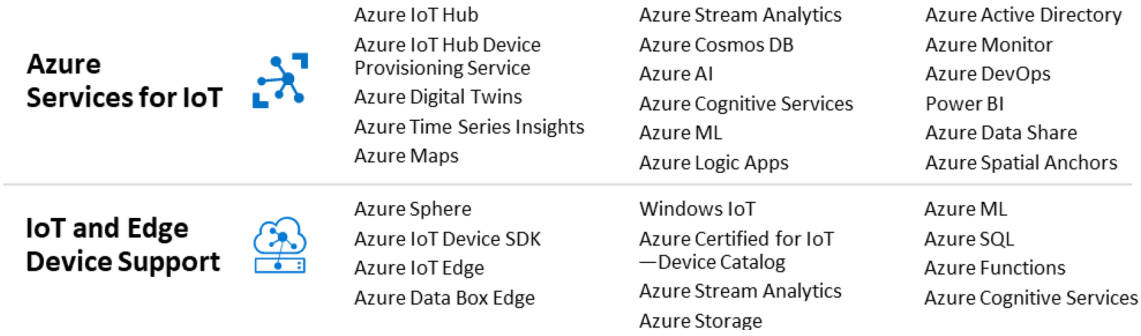

Fig. 2 Azure IoT technologies and services

Microsoft security documentation reports that "security is integrated into every aspect of Azure".[5] The latter provides various services and tools to help developers create secure solutions, as also analyzed in previous studies [7,29].

However, similarly to Amazon, Microsoft released a white paper to clarify the concept of a *shared responsibility* when developers are using a cloud platform offered by a third-party [30]. Though the customers' responsibilities visually decrease according to the kind of infrastructure used in their model, the responsibility is always shared. Therefore, even when using Microsoft's platform, the programmers should not underestimate their role during the applications' development.

Specifically, the main focus of our Azure analysis is Azure IoT, a collection of cloud services designed to connect, monitor and control a large number of IoT assets. Microsoft classifies its components in two main categories as follows: *Azure Services for IoT* and *IoT and Edge Device Support* (Fig. 2). Among them, similarly to AWS, Microsoft offers device software to integrate and connect IoT devices with its cloud platform. In addition, Azure also proposes *Azure IoT Central*, a fully-managed platform that can be used to create an IoT solution starting from an application template.

4 Use case

An initial understanding of the security issues faced by a novice developer was formalized in a small-scale study, conducted in the context of a professional training course for a consulting engineering company in Turin, Italy. The course's main goal was to teach a small group of programmers how to develop a cloud-IoT-based application from scratch. This course started by introducing the IoT world, explaining possible advantages, disadvantages, and challenges. It also explained one of the most used protocols in this field: Message Queuing Telemetry Transport (MQTT). After this, the programmers learned the fundamentals of cloud computing technologies starting from a general perspective. Subsequently, they were introduced to AWS, focusing on cloud computing for IoT. In the end, the software developers received some additional concepts related to developing an HTTP-based server using the Representational State Transfer (REST) approach.

The course was organized in nine non-consecutive days (distributed among six weeks), with theoretical sections in the mornings and practical experiences in the afternoons: one introductory lecture, four lectures reserved to cloud computing, and four lectures for AWS-IoT. To successfully pass the course, the attendees, divided into groups, had to develop a full-working prototype with all the components shown in Fig. 3. The participants had to present also their prototype in a final discussion one week after the end of the course.

The prototype had to be composed of the following cloud components:

- an *IoT gateway*: to manage bidirectional communication with IoT devices;
- an *APIs gateway*: to manage requests from developers and final users;
- a *serverless* component: to run code in response to events, to schedule processes, and to interact with some acting devices;
- a *database*: to quickly store a potentially massive flow of data from many IoT devices.

If we consider the mapping of the high-level architectural components onto the services offered by the cloud platforms, we examined as IoT gateway the IoT Core in AWS and the IoT Hub in Azure. The service for managing the Application Programming Interfaces (APIs) is called API Gateway in AWS and API Management in Azure. Regarding the serverless component, it could be mapped as Lambda in AWS, while Azure simply called it Functions. To conclude, talking about the database, considering that NoSQL databases are very suitable for IoT applications [31], we decide to adopt: DynamoDB in AWS and Cosmos DB in Azure.

[5] https://docs.microsoft.com/en-us/azure/security/, last visited on January 14, 2021.

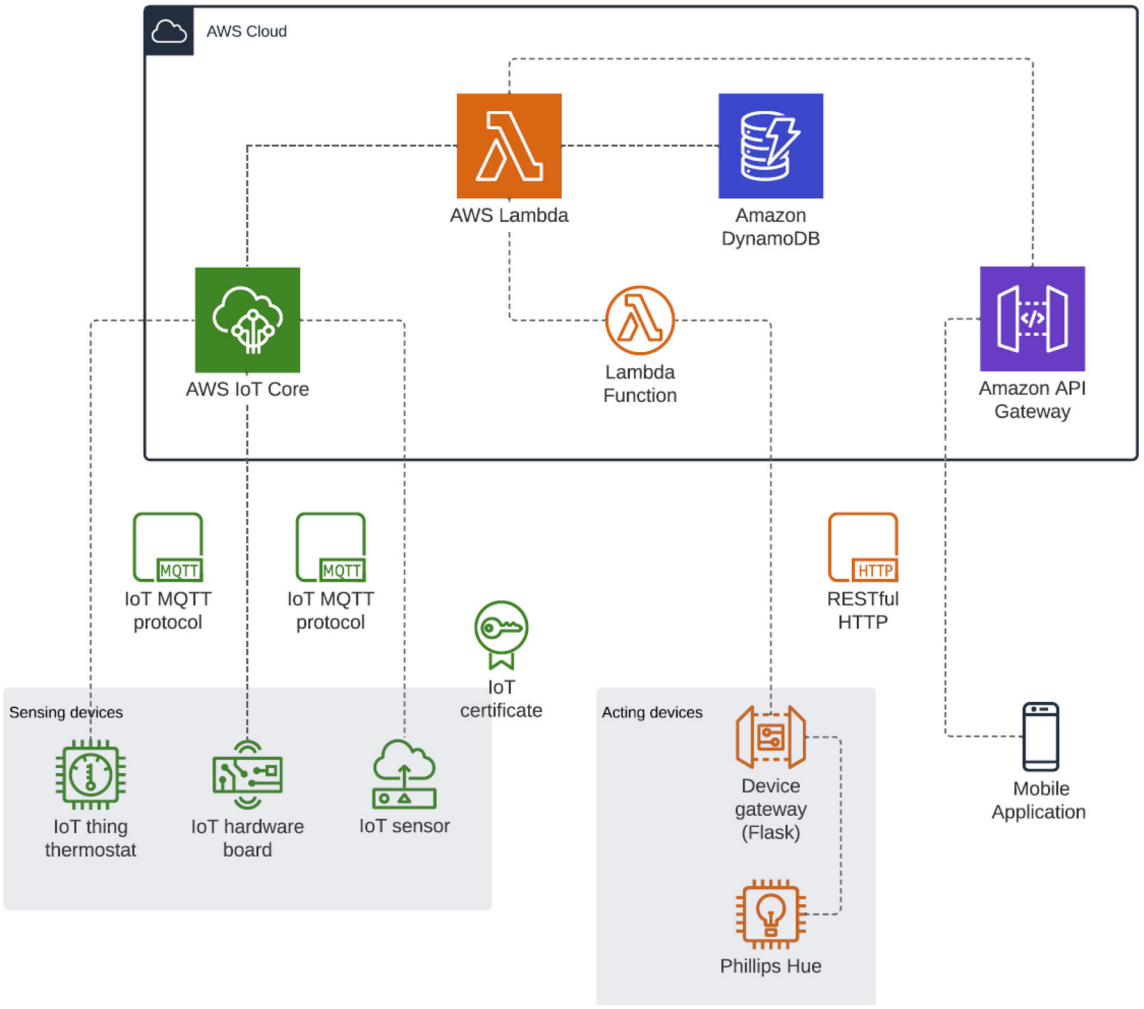

Fig. 3 The architectural schema of a use case application in AWS

Adding more details on the third component, it allows running code without managing servers or creating workload-aware scaling logic. The great advantage of using such a component is that developers have only to provide the code, and the platform automatically runs it allocating execution power based on the incoming requests. Furthermore, these services support many different programming languages. AWS Lambda supports Java, Go, PowerShell, Node.js JavaScript, C#, Python, and Ruby code. In addition, it provides a runtime API that allows developers to use any additional programming languages. Instead, Azure Functions support C#, JavaScript Node.js, and F# since version 1. Then, from version 2, they also support Java, PowerShell, Python, and TypeScript.

Regarding the communication protocols, for this prototype, we suggested taking advantage of the two primary protocols explained during the course and typically used in basic IoT applications: MQTT and HTTP.

No custom hardware was developed, and IoT devices were selected among available off-the-shelf ones, as the scope of the course was more on the cloud and interoperability aspects, rather than on connected embedded devices.

4.1 Main architecture attack points

From an architectural of view, it is possible to divide the main hardware and software components of a simple cloud-IoT project into the following four different categories:

- *Sensing devices*: devices used to retrieve some data from the physical world (e.g., a temperature sensor).
- *Acting devices*: devices used to act on the physical world (e.g., a smart lamp).
- *The cloud*: a cloud back-end server able to manipulate and store all the data necessary for the application.

- *Front-end devices*: used to interact with the back-end (e.g., a mobile application).

Considering the proposed architecture (Fig. 3), we defined the main attack points according to the state-of-the-art security issues in the IoT field (e.g., [1,21,24,32]). In particular, to conduct this analysis, we decided to use the STRIDE threat model [33]. Initially developed in the Security Engineering and Communications group at Microsoft, different scholars used this model in many different research activities related to IoT (e.g., for the security of Cyber-Physical Systems [34], Smart Cities [35], or Smart Grids [36]), and it is applicable and suitable to the use case. STRIDE is an acronym for the six categories used by the model to classify security threats: Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service (DoS), and Elevation of privilege. Each of these threat-categories is associated with a related security property: Authenticity, Integrity, Non-repudiability, Confidentiality, Availability, and Authorization, respectively. In this specific use case, we were not interested in enforcing non-repudiability, so we did not consider issues related to repudiation threats.

We want to clarify that we defined these attack points having a cloud-centered approach in mind. For this reason, we did not consider issues related to the possible constrained capabilities of the IoT devices or the security of the mobile applications installed on the front-end devices. We followed this approach in defining the principal architecture's issues to create a scenario that is as generic as possible, analyzing a set of threats independent of the physical devices used by the developers.

According to the STRIDE framework, we identified the following five main attack points in the use case architecture:

1. the *data flows* between a sensor and the back-end;
2. the *data flows* between the back-end and a physical actuator (e.g., a smart lamp);
3. the *data flows* between the API Gateway and the user's device;
4. the developed code stored and executed inside the *back-end* (e.g., the serverless component);
5. the back-end *database* (i.e., the NoSQL database).

Considering the first three attack points (1, 2, and 3), all the *data flows* could be primarily subject to *spoofing* and *tampering*. Some possible attacks related to these threats are Replay attacks[6] or Man-In-The-Middle (MITM) attacks.[7] This last attack is particularly relevant because it could exploit basically all the security threats presented by the STRIDE model.

Moreover, discussing the *code executed on the back-end* (point 4), in the proposed IoT architecture it was implemented by using the AWS Lambda component to develop the back-end's functionalities. This component is reasonably the best point for executing an *elevation of privilege* attack. Furthermore, from this component it could also be easy to obtain access to sensible information (i.e., *information disclosure*). Indeed, if the code on the back-end of the application is compromised, attackers could become able to execute arbitrary functions inside the developed system.

In conclusion, if the *back-end database* (point 5) is not well protected and the stored data are not adequately encrypted, the content of the database could be altered (i.e., *tampering*) or the system might be a victim of a steal of information (i.e., *information disclosure*)—this information could be directly used by the malicious user or sold (e.g., login credentials or payment data are interesting information to sell).

5 Developers security perspective

The analysis of the security perspectives in developing the proposed use case (Sect. 4) will allow us to better comprehend the behavior of novice programmers in similar scenarios. In particular, we aim at understanding if and how the features of the cloud platforms' components may compensate from the developers' lack of specific knowledge.

Hence, to understand their way of thinking, we prepared a survey starting from the use case presented in Sect. 4. The main research questions behind the delivered survey are:

- **RQ1**: What is the security perception of the novice IoT developers?
- **RQ2**: Did novice IoT developers think about security issues during application development?
- **RQ3**: Did novice IoT developers act to increase the cybersecurity of their application?

This section explains the survey structure and presents the answers provided by our participants.

5.1 Survey structure

At the end of the course, after the closing review of the projects—when the students are already aware if they successfully passed the final examination—we asked the attendees to voluntarily participate in a survey. We provided them a link to an anonymous online questionnaire hosted on Google Form. All the questions in the survey required a mandatory answer. When all participants completed the

[6] An attack in which a valid data pack is stored and sent by a malicious user in a different moment.

[7] An attack in which a malicious user intercepts and modifies the data meanwhile they are being transmitted.

Table 1 Survey closed questions

Question	Description
How much do you feel expert about cybersecurity?	Values from one to five.
Using AWS, who is responsible for the security of the system you develop?	Three possible answers.
Sort the following architecture elements from the easiest to the most difficult to attack.	Ranking order of the 5 options.
Sort the same elements according to the potential severity in case of a successful attack.	Ranking order of the 5 options.
How many of these elements did you considered as "potentially attackable" during the development?	Values from zero to five.
Did you know that AWS offers some security support tool? (e.g., AWS IoT Device Defender)	Three possible answers.

survey, we solicited an open discussion to better understand the meaning of their answers.

The three main sections of the survey were: "Background and Individual Studying", "Possible Attack Points", and "Countermeasures and Best Practices". We divided the survey in different parts to put the participants' focus on different aspects, without overloading them with too much information.

Indeed, in the first section, we never explicitly mentioned to the attendees that the use case architecture had vulnerabilities. In that part, we were interested in understanding their thoughts before being influenced by our use case analysis. So, the first open question of this part was: "Among the various aspects of an IoT system, in your opinion, how important is security?". We started giving participants a blank space for collecting some insights about their cybersecurity perception before influencing them in any way. Therefore, we asked two closed questions: "How much do you feel expert about cybersecurity?" and "Using AWS, who is responsible for the security of the system you develop?". The first question requires a numerical answer from one ("not expert at all") to five ("very expert"); the second one has three possible answers: "the developer", "Amazon", and "both of them". Then, after showing Fig. 3 as a reminder of their systems' architecture, we proposed two open questions to force them to think about (and explain) what they did during the application's development. The two questions were: "Are there any potential security issues in this architecture? If so, which ones?", and "If you have indicated any issues, how would you manage them during the application's development?".

The purpose of the second part of the questionnaire, "Possible Attack Points", was to understand what they think about the attack points we identified and presented in Sect. 4.1. This survey section is composed of three closed questions and one open question. The first question requires the learners to sort the five attack points presented in Sect. 4.1 from the easiest to the most difficult to attack. Then, we asked them to sort the same attack points according to the possible severity if an attack succeeded. After that, the programmers had to answer the following open question: "what do you think would be the worst possible consequence in case the most serious point is attacked?". To conclude, we asked the attendees how many of those points they considered attackable while developing their projects. This closed question required a numerical answer from zero to five.

While the first two sections were more focused on the first two research questions, the goal of the last part of the survey was to understand if the participants tried to take some countermeasures to improve their applications' security (i.e., the main focus is on RQ3). This section has five open questions and one closed question. Specifically, we asked our novice IoT developers to explain what criteria they used for their password, and if they had created more users for their applications. Then, we investigated if they thought about encryption of data in transit and data at rest (specifically on the database). The only closed question of the section is related to security support applications: "did you know that AWS offers some security support applications? (e.g., AWS IoT Device Defender[8])". This question has the following three possible answers: "yes", "no", and "yes, but never used". To conclude, an open follow-up question digs in more details: "did you consider using a support application (e.g., AWS IoT Device Defender) to develop your system? (Please, answer Yes/No and why)".

To summarize, we reported in Table 1 all the closed questions of the presented survey, while Table 2 contains all the open questions.

5.2 Survey results

Nine novice IoT programmers attended the professional training course; we collected answers from six of them (i.e., the survey had an answer rate of 67%). All the attendees were male, and they followed this course to start working on their first cloud-IoT project for the consulting company. However, most of them already had professional experience on developing or testing embedded systems, in the context of industrial automation and railway industries.

On the one hand, the participants felt very inexperienced about cybersecurity. Indeed on a scale from one to five (where

[8] https://aws.amazon.com/iot-device-defender/, last visited on February 1, 2021.

Table 2 Survey open questions

Question
Among the various aspects of an IoT system, in your opinion, how important is security?
Are there any potential security issues in this architecture? If so, which ones?
If you have indicated any issues, how would you manage them during the application's development?
What do you think would be the worst possible consequence in case the most serious point is attacked?
What criteria did you use for choosing your passwords?
Did you create multiple users (with different permissions) to access your services (e.g., AWS Lambda)? (Please, answer Yes/No and why)
Did you ever verify if connections to and from AWS are encrypted (e.g., using TLS)? (Please, answer Yes/No and why)
Did you ever verify if the data contained on the database are encrypted at rest? (Please, answer Yes/No and why) Did you consider using a support tool (e.g., AWS IoT Device Defender) to develop your system? (Please, answer Yes/No and why)

one means "not expert at all" while five means "very expert"), five developers out of six answered one, while the remaining one selected two (Fig. 4). On the other hand, according to the collected insights, they all thought cybersecurity is quite important. Indeed, one participant declared that cybersecurity is "very important", another said it is "on average important", and a third one states that "authentication is an important feature". In addition, three developers also specified that cybersecurity relevance depends on the severity of the implemented software solution.

An interesting outcome comes from the question: "Using AWS, who is responsible for the security of the system you develop?". Indeed, four developers said that responsibility is shared among "both developer and AWS", while the other two participants consider the responsibility *entirely of the developer*. No one assigns the responsibility uniquely to AWS (Fig. 5).

All the attendees think that the implemented architecture could include at least one security issue; in fact, every developer highlighted some threat in the open question: "Are there any potential security issues in this architecture? If so, which ones?". However, in the related open question, all of them declared they did not act to mitigate the previously cited threats.

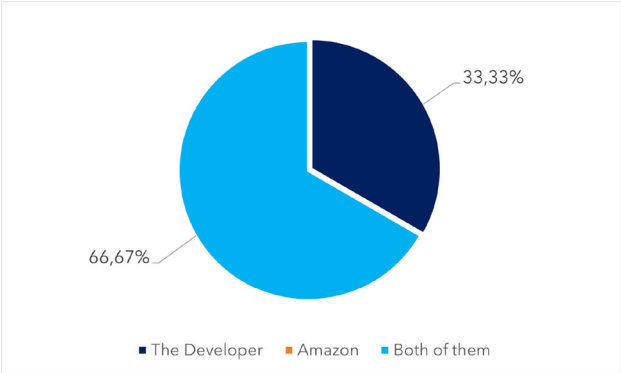

Fig. 5 Novice IoT programmers' perception of security responsibilities

Discussing the second part of the survey (the one related to the attack points presented in Sect. 4.1), Fig. 6 shows that half of our novice IoT programmers consider potentially attackable at most two of the five attack points (three answers). It is also noticeable that two participants did not consider any of the raised issues during the application's development.

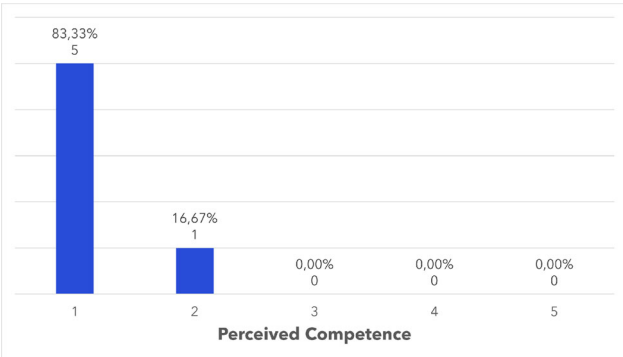

Fig. 4 Novice IoT programmers' perceived competence about cybersecurity. One means "not expert at all" while five means "very expert"

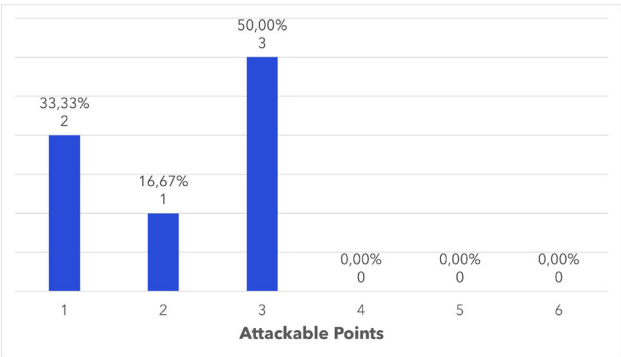

Fig. 6 The number of attack points considered attackable by the novice IoT programmers

Table 3 Ease of attacking architecture elements according to the novice IoT programmers. The reported values represent the number of developers choosing that answer

Attack points	1 (not easy)	2	3	4	5 (very easy)
Sending data from sensors to AWS	1	1	1	1	2
The developed lambda functions	1	1	3	0	1
The Amazon's database	2	1	1	2	0
The REST APIs	1	1	1	2	1
Sending commands to actuation devices	1	2	0	1	2

Table 4 Potential severity in case of a successful attack for various architectural elements, according to the novice IoT programmers. The values inside the tables represent the number of developers choosing that answer

Attack points	1 (not serious)	2	3	4	5 (very serious)
Sending data from sensors to AWS	2	0	2	1	1
The developed lambda functions	1	1	1	1	2
The Amazon's database	2	3	1	0	0
The REST APIs	1	1	2	2	0
Sending commands to actuation devices	0	1	0	2	3

In addition, the participants declared that the AWS database is the attack point most secure among the presented elements. On the contrary, the data flows between the AWS back-end and the sensors/actuators are considered the less secure points equally. These answers are reported in Table 3. As we already explained in Sect. 5.1, the range of values goes from one, "not easy at all" (to attack), to five, "very easy" (to attack).

Proceeding with the possible severity in case of a successful attack, our participants declared that the most critical point is the data flow from the back-end to the actuators. Then, the second most critical point is represented by the developed Lambda functions, while the third is the data flow from sensors to the back-end. These answers are available in Table 4. The range of values is between one, (attack) "not serious at all", and five, "very serious" (attack).

Instead, considering the open answer to the question: "what do you think would be the worst possible consequence in case the most serious point is attacked?", five participants consider a cyber-physical attack[9] the worst possible threat. Indeed, they are mainly afraid that an attacker could take control of the system to damage a machine or a person. Another concern cited by three developers is related to data manipulation or data loss.

To conclude the presentation of our survey results, we are going to discuss the answers collected in the last survey section. The first open question asked information about the chosen passwords for the AWS accounts. According to the answers, all participants created a strong password (probably because AWS enforces a password policy). However, when we asked if they had created any additional account with lesser privileges, only one developer declared to had used the service for creating secondary accounts. In contrast, others simply used the root account. In addition, in their open answers, just two participants specified that they did not create additional accounts (with fewer privileges) during the project's development. However, they are aware that they should have created at least one. Discussing now data flow encryption, four attendees stated they did not check whether the platform uses a mechanism for encrypting the data. Only one developer declared to have verified that the Transport Layer Security (TLS) protocol was used on the data flow from the sensors to the AWS back-end. Moreover, considering the encryption of data at rest, a large majority of participants (five) answered that they did not check if AWS encrypts by default the data stored inside DynamoDB. To conclude, discussing the last two questions, no one of the participants had used an additional tool to improve the security of his application. One developer declared to have heard about *AWS IoT Device Defender*, but he did not use it for his project due to the time constraint of delivering the application on time.

6 Cloud-IoT platforms analysis

This section describes what countermeasures could be taken by the two platforms' components against the attack points presented in Sect. 4.1.

Considering that this paper (and our use case) is focused on a cloud-IoT scenario, our security analysis is more focused on *AWS IoT* and *Azure IoT*, the sets of components offered by the platforms for this specific application domain. However, we also have considered elements not strictly belonging to IoT but still valuable for a cloud-IoT use case. For instance, the already cited serverless computing services offered by the two platforms (Sect. 3). Indeed, considering their character-

[9] A security breach in cyberspace that impacts the physical environment (e.g., activating or deactivating a machine).

istics, this kind of component seems particularly valuable for a novice programmer new to the IoT world.

Therefore, in this section, we will analyze the components that better fit the use case (Sect. 4).

6.1 Data flow analysis

Data flows are the target of three of the attack points discussed in Sect. 4.1. Data transmission is a critical process; at that moment, data could be eavesdropped, altered, and even forged. The first threat could reduce the application's privacy, but the other two can even alter its functionalities. The traditional approach to counter these issues is encrypting the transferred information.

One of the most common protocols used for this purpose is HTTPS. This protocol augments classic HTTP adding the Transport Layer Security (TLS) protocol.

TLS was defined by the Internet Engineering Task Force (IETF) in 1999 [37], and it is a transport layer protocol based on asymmetric cryptography. It is the successor of the now-deprecated Secure Sockets Layer (SSL), and its latest version, TLS 1.3, was released in August 2018 inside RFC 8446 [38]. Recently, in March 2021, IETF formally deprecated TLS 1.0 and 1.1 and, even if TLS 1.2 is still considered acceptable, they recommended starting the migration to the latest version [39]. However, choosing the correct version of TLS is not enough to obtain an adequate level of protection. This protocol could use many different cipher suites,[10] so selecting an appropriate suite is crucial to configure a secure connection. Until TLS 1.2, cipher suites include a key exchange algorithm, an authentication algorithm, a link encryption algorithm, and a message authentication code (MAC) algorithm. From version 1.3, the first three algorithms were replaced by an Authenticated Encryption with Associated Data (AEAD) algorithm. This particular class of algorithms simultaneously ensures the confidentiality and the authenticity of both encrypted and unencrypted information in the messages.

Therefore, when possible, developers should always prefer TLS 1.3, which has a faster hand-shake and is more secure by design. When using this version is impossible, programmers should not adopt a deprecated version of the protocol and configure their applications to handle only secure cipher suites. To verify the current status of a cipher suite, developers may benefit from online tools like Ciphersuite.info [40].

Another protocol used mainly to establish bidirectional communication among the IoT devices and the IoT gateway is MQTT [41]. MQTT is a publish-subscribe network protocol designed by the Organization for the Advancement of Structured Information Standards (OASIS) to transport messages between any kind of device. It is lightweight and typically runs over the TCP/IP stack. For these reasons, it is particularly suitable for IoT applications. Like HTTP, MQTT does not implement any particularly effective security protection by default.

So, even in this case, to guarantee confidentiality and integrity of the transmitted data, developer should apply encryption mechanisms. Considering that MQTT is an application protocol, like HTTP, one of the most common ways to protect the transferred data is using TLS. As we will see, adopting MQTT over TLS (MQTTS) is a strategy supported and suggested by both the cloud platforms analyzed in this section.

6.1.1 AWS data flow management

To ensure the security of transmitted data, AWS documentation says that users should always connect to the AWS back-end employing encryption mechanisms like *TLS* [26].

The first analyzed Amazon component is the AWS API Gateway. This service makes it easier for developers to create, publish, maintain, monitor, and secure APIs. It handles all the tasks involved in accepting and processing many concurrent API calls, including traffic management, authorization, access control, throttling, monitoring, and API version management. To test this component, we created a simple Lambda function associated with an API Gateway. Then, we interacted with the gateway using Postman, a widespread API platform created to build and test APIs [42]. Actually, we observed that when users create new public access to this gateway, the TLS protection is enabled by default. However, even if this component could help developers to have a reliable and secure contact with the AWS back-end, we noticed that the TLS security policy enabled by default is version 1.0 (as it is possible to see also on the AWS website[11]). As we discussed before, IETF deprecated this version, so Amazon should not allow this configuration as the default one. Moreover, AWS's TLS version 1.0 security policy supports some weak cipher suites like "DES-CBC3-SHA".[12] This cipher suite contains the Triple Data Encryption Standard, an encryption algorithm that the National Institute of Standards and Technology (NIST) formally deprecated and will disallow from 2023 [43]. However, not all the AWS API Gateway endpoints have this security policy enabled by default. Indeed, AWS also provides specific endpoints for organizations that must comply with the Federal Information Processing Standard (FIPS) Publication 140-2. FIPS 140-2 is

[10] A cipher suite is a pool of algorithms used to establish a secure network connection.

[11] https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-custom-domain-tls-version.html, last visited on September 16, 2021.

[12] https://ciphersuite.info/cs/TLS_RSA_WITH_3DES_EDE_CBC_SHA/, last visited on February 04, 2022.

a USA and Canadian government standard that specifies the security requirements for cryptographic modules that protect sensitive information. From April 1, 2021, Amazon had updated these endpoints to use TLS 1.2 and above.[13] Hence, at least for these endpoints, the Amazon approach is a good trade-off between usability and state-of-the-art security.

Considering the AWS IoT Core instead, this component is a cloud service that enables connected devices to interact with the back-end and each other securely. It can process many devices and messages and route those messages to AWS IoT endpoints and other devices. Unlike the AWS API Gateway, this service is more secure by default; indeed, it uses a minimum of TLS v1.2 to establish all the communications.[14] This version is used for Web Sockets, HTTPS (HTTP over TLS), and MQTTS (MQTT over TLS) connections. As an additional requirement, AWS refuses all the TLS connections without the extension field Server Name Indication.[15] Instead, talking about the security of the possible cipher suites, there are still few allowed cipher suites involving the usage of SHA1 as their authentication algorithm. NIST already deprecated this algorithm in 2011 [43], and some scholars demonstrated its insecurity in 2017 [44].

However, connecting IoT devices to AWS IoT Core is quite easy. If the developers follow the tutorial on the AWS website, they will get a pair of public-private keys to establish a secure connection for each IoT device. Furthermore, AWS also provides some IoT Device Software Development Kits (SDKs) to simplify the connection process. At the time of writing, Amazon provides five open source SDKs written in C++, Embedded C, Python, JavaScript, and Java. Using the available IoT Device SDKs is the suggested approach for connecting every IoT device to the AWS back-end. Indeed, in this case, AWS implements a good trade-off between usability and state-of-the-art security.

Regarding the possible data flows among an AWS Lambda function and a physical device, AWS Lambda blocks inbound network connections. On the other side, the component allows for outbound connections only TCP/IP and UDP/IP sockets.

Currently, if a user requires additional security, AWS offers the possibility to create and configure a virtual private network in the cloud called Virtual Private Cloud (VPC). This solution also enables the possibility of using IPsec to contact the VPC. IPsec is the secure version of the traditional Internet Protocol (IP), and it was designed to provide cryptographically-based security for IPv4 and IPv6 [45]. Even if this approach would give more security to the transmitted data, it could not be so intuitive for a novice developer.

Furthermore, Amazon offers two additional components called AWS IoT Device Management and AWS IoT Device Defender to facilitate the management of the IoT devices connected to the AWS platform. The first one is a component agnostic to the device type that simplifies securely registering, organizing, monitoring, and remotely managing IoT devices at scale. Instead, the second component automatically audits IoT configurations associated with the connected devices against a set of defined IoT security best practices. If something seems to deviate from the expected behaviors, IoT Device Defender pushes an alarm so the developer can take action to mitigate the issue. In particular, AWS suggests using these components when the number of connected devices is significant.

6.1.2 Azure data flow management

Even Azure recommends always protecting data in transit with an encryption strategy like HTTPS or a VPN.[16] In particular, to connect customers' networks to Azure, Microsoft offers a VPN Gateway service that enables the usage of IPsec. Users could also utilize this service to send encrypted traffic between Azure virtual networks over Microsoft's infrastructure.

The first analyzed component of this section is API Management. Like AWS API Gateway, this element helps developers properly manage their exposed APIs. In this case, to test the component, together with Postman, we used the API testing environment offered by Azure. Currently, when users create a new endpoint for their APIs, TLS protection is enabled by default to v1.2. However, even if the previous versions are disabled by default, during the creation process, users can decide to increase the endpoint's compatibility with older versions of TLS and with the obsolete SSL 3.0.

Concerning the second component, the developers can reach the Azure IoT Hub using both HTTP and MQTT. However, the cloud service ensures that all device communications are secured with TLS.[17] Specifically, IoT Hub offers two authentication methods between the IoT devices and the back-end. Users can use a Shared Access Signature (SAS) token authentication or an X.509 certificate. A SAS token is a string previously generated (and not stored) by the back-end used by a client (device) to be recognized by the server. This token also contains an HMAC-SHA256 signature string to authenticate and protect the integrity of the communications through this method. An X.509 certificate

[13] https://aws.amazon.com/compliance/fips/, last visited on September 16, 2021.

[14] https://docs.aws.amazon.com/iot/latest/developerguide/transport-security.html, last visited on September 16, 2021.

[15] https://datatracker.ietf.org/doc/html/rfc6066#section-3, last visited on September 28, 2021.

[16] https://docs.microsoft.com/en-us/azure/security, last visited on January 17, 2022.

[17] https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support, last visited on January 28, 2022.

is a standard format for public key certificates introduced by the International Telecommunication Union - Telecommunication Standardization Bureau (ITU-T). It is a certificate used in many internet protocols, including TLS. By default, the IoT Hub establishes a connection with the IoT devices using TLS 1.2, 1.1, and 1.0 (in this specific order). However, it is possible to disable the deprecated version of TLS inside the IoT Hub configuration, but only in some geographical regions (mainly in the United States).

Similarly to AWS, also Azure provides a Software Development Kit (SDK) to help developers interact with the IoT Hub. In particular, Microsoft offers an IoT Hub Service SDK and an IoT Hub Device SDK. The first SDK is designed to facilitate building applications that interact with the IoT Hub (e.g., for managing devices connected to the hub). It is currently available for .NET, Java, Node.js, and Python. Instead, the second SDK simplifies the process of connecting IoT devices to Azure. It is currently available for the same programming language as the previous one, plus C and Embedded C languages. In addition, this SDK also supports Real-Time Operating Systems for embedding devices like Azure RTOS and FreeRTOS. Microsoft strongly suggests taking advantage of these SDKs, even through a blog post that specifies why developers should use them to increase their applications' security.[18]

Considering the communications among a client and an Azure Function, by default, the function's endpoints allow connection only with plain HTTP or with HTTPS. However, every Azure endpoint can enforce HTTPS-only connections and specify the minimum allowed version of TLS among 1.0, 1.1, and 1.2.

To conclude, among Azure components, there is a valuable tool for enhancing the security of the IoT applications, which is called Microsoft Defender for IoT. This component allows monitoring the whole IoT solution from a dashboard. Once enabled, it helps developers connect their IoT devices and uses Defender for Cloud—an Azure tool for security posture management and threat protection—to provide security recommendations and alerts for the connected resources.

6.2 Back-end analysis

Considering that back-end security is a duty of the cloud platforms' providers, in this section, we analyzed what a user could do to protect the access to the back-end services better or to reduce damages in case of unauthorized access. Obviously, users must register an account (and successively authenticate themselves) to access the cloud back-end functionalities (like AWS Lambda or the Azure Functions). During these interactions, registration and login data flows are protected using HTTPS. Indeed, assuming that both platforms (and the associated services) had implemented registration and login correctly, the two primary possible ways to compromise the back-end are associated with an *inadequate privileges policy* and choosing *weak passwords*. So, in this section, we will discuss what kind of accounts customers could create and how they can choose passwords. Furthermore, an example of a proper privileges policy that we suggest adopting is the *Principle of Least Privilege* [46]. The main idea of this principle is to give to any user, application, or process only the *minimum privileges* necessary to complete its tasks.

6.2.1 AWS back-end management

AWS offers two kinds of accounts for authenticating users: *root* user and *Identity and Access Management* (IAM) user. When a customer registers an account to AWS, Amazon automatically associates a root user to that account. Root accounts have the highest possible privileges on the AWS platform. Using these accounts, users can generate an arbitrary number of IAM accounts and assign them the privileges necessary for their activities (e.g., the right to read data from a particular database or the possibility to use a specific service). According to the Principle of Least Privilege, Amazon recommends to its customers to use the root account mainly for generating some IAM accounts (and for those tasks that require necessary to be administrator). However, they do not enforce this best practice in any way. From our point of view, AWS should force its users to take advantage of this feature and create at least an IAM account. Clearly, Amazon should explain to its customers why they are forced to do so. We think that this approach could help new users better understand the importance of this security concept.

Talking about the password policy, AWS enforces a quite robust policy both for root and IAM accounts. At the time of writing, users must create passwords with a minimum length of eight characters, including at least two of these characteristics:

- including a digit;
- including a non-alphanumeric character;
- including lowercase and uppercase characters.

Furthermore, a root user can build different password policies for the created IAM account, they can configure a temporary password and ask the IAM user to change it at first login.

Nevertheless, we observed that AWS seems not to have taken any specific countermeasures for password dictionary attacks. E.g., a password like "Amaz0nWS" is cor-

[18] https://azure.microsoft.com/it-it/blog/benefits-of-using-the-azure-iot-sdks-in-your-azure-iot-solution/, last visited on January 18, 2022.

rectly accepted when registering a new account.[19] However, AWS could automatically generate passwords for the IAM accounts.

These auto-generated passwords include all the previously cited characteristics and have a length of 16 characters.

To conclude, we observed that through the IAM control panel, users could enable a second authentication factor both for root and IAM accounts. However, this security enhancement is only suggested, and it is not enforced in any way.

6.2.2 Azure back-end management

To access Azure services, users must have a Microsoft or a GitHub account. So, the password policy applied for Azure is the same as these accounts. Unfortunately, the password policy enforced by Microsoft accounts is not very robust. Currently, it forces the users to create passwords with a minimum length of eight symbols and at least two of the following four characteristics:

- including at least a number;
- including at least a non-alphanumeric symbol;
- including a lowercase character;
- including an uppercase character.

From one side, at least Microsoft implements small countermeasures against dictionary attacks. We tried to create an account using the password "Microsoft", and the registration process refused that specific password for over-usage. On the other side, we noticed that after this rejection, the registration process reduced the minimum number of characters for the password to six, and we have been allowed to use the password "MAzure".

Instead, the password policy enforced by GitHub accounts is slightly more robust. Currently, users have two possible choices: a password with a minimum length of 15 characters or a password with eight symbols containing at least a lowercase character and at least a number. Even GitHub tests the inserted password for dictionary attacks. We tried to create an account using the password "github22", and we received the error: "password may be compromised". Nevertheless, the registration process allowed, and considered strong, passwords like "msazure1" or "microsoftgithub". Currently, according to "How Secure is My Password?" [47], an attacker could crack the first two passwords in around one minute and the third one in 1000 years. However, this third password is considered more robust only thanks to its length [48]. In fact, the tool suggests that the password should include not only letters and that if it is a dictionary word could be easily cracked.

[19] Currently, according to "How Secure is My Password?" [47], a malicious user could crack this password in around 1 h.

Once users had access to the Azure back-end, Microsoft offers the possibility of creating other accounts with limited privileges using *Azure Active Directory*, its cloud-based identity and access management service. Surprisingly these accounts have a default password policy slightly more robust than a traditional Microsoft account. Since it requires a minimum length of eight symbols and at least three of the following four types of symbols:

- digits;
- lowercase characters;
- uppercase characters;
- non-alphanumeric (among a proposed set).

When secondary users log in for the first time, Azure forces them to change the password (always respecting the previously cited criteria) and configure a second authentication factor (even if they can ignore this last step for the first 14 days).

6.3 Database analysis

Continuous flows of data usually characterize IoT projects. When these communications happen among a considerable amount of devices at a high frequency, the quantity of data to manage and store becomes easily massive. For this reason, in our use case, we proposed to use a high-performance non-relational database. This section analyzes how the two platforms manage the security of their most famous NoSQL databases: *AWS DynamoDB* and *Azure Cosmos DB*.

In this section, we analyzed three crucial points of database management: *data encryption at rest*, *backups*, and *access to the database*. The first feature is essential to reduce the possibility that, in case a database is stolen, thieves can read it. The second feature is important both for reliability and security reasons. Indeed, a backup could be helpful to protect data against accidental writes or delete operations and even against direct attacks on the stored data. Instead, correctly managing how users can access the database is important not only for ensuring that they can dispose only of the correct data but also for protecting data in transit.

6.3.1 AWS database management

Amazon provides different database solutions (both relational and NoSQL), and it offers database-related services such as data-warehouses. In particular, in our use case, we proposed to our students using DynamoDB, a NoSQL database service that provides fast performance with seamless scalability. Even in this case, users can access Amazon DynamoDB via TSL-encrypted endpoints.

By default, the DynamoDB service encrypts all data at rest to enhance data security.

To cipher the data, AWS uses the Advanced Encryption Standard (AES) algorithm with the longest possible key (256 bits) [49]. This algorithm was approved by FIPS, and it is considered secure both for government and non-government organizations. Indeed, it is also part of the Commercial National Security Algorithm Suite.[20]

To encrypt each table, AWS generates and uses a default encryption key. The platform kept these keys inside the Key Management Service (KMS). In addition to this default encryption approach (also called *AWS owned key*), on which the developer has no control over the encryption keys, Amazon provides two other different strategies. The first one, called *AWS managed key*, allows customers to store a custom key and leaves the management to the KMS. The other one, called *customer managed key*, gives the user complete control over the KMS keys. Keys are stored inside the AWS account, but they are created, owned, and managed by the programmer. Each time a user creates a new table, DynamoDB offers the possibility to choose between these three strategies. The encryption strategy can be modified at any time.

To avoid the risk of losing data, Amazon offers two different approaches to backup database content: *on-demand backup* and *point-in-time recovery*. As the name suggests, the first approach creates a full backup on-demand of the desired DynamoDB tables. This approach is particularly indicated for long-term data retention and archival. Meanwhile, the point-in-time recovery approach sets up an automatic periodic backup for all the selected tables. With this strategy, it is possible to restore data at any point in time within the last 35 days. As additional functionality, developers can store the backups in the same region where the application is deployed or even in a different one.

To control who can use the DynamoDB resources and API, the users have to set up permissions in the IAM service. Through IAM policy, a user can also specify a fine-grained access control policy (e.g., to allow or deny access to specific rows or columns). Additionally, each request to the database must contain a valid HMAC-SHA-256 signature. HMAC (Hash-based Message Authentication Code) is an authentication code to be sent together with the request, generated using SHA-256 (Secure Hash Algorithm), a standard cryptographic hash function belonging to the SHA-2 family [50]. Even if more strong hash algorithms are now available (like SHA-512 or the SHA-3 family [51]), SHA-256 is still considered a robust alternative. To help programmers in creating their applications, the AWS Software Developer Kits automatically sign users' requests. However, developers can also write their HTTP requests providing the signature in the header of the requests.

[20] https://apps.nsa.gov/iaarchive/programs/iad-initiatives/cnsa-suite.cfm, last visited on January 13, 2022.

In conclusion, we could say that AWS correctly manages the security of its databases by default. Developers should only remember to enable the back-up of their databases.

6.3.2 Azure database management

Even Azure offers several databases (relational and non-relational) and data warehousing solutions. Considering our use case, the Azure database that better fits our needs is reasonably Azure Cosmos DB — which is also the suggested Azure database for IoT applications. This database is comparable with AWS Dynamo DB, it is a NoSQL database with a fast response time, a high availability, and it is able to scale automatically. In addition, this database could also be manipulated with the same APIs of MongoDB and Cassandra. Like DynamoDB, also Cosmos DB service encrypts all data at rest by default using AES-256 and a key managed directly by Microsoft. Unlike AWS, Azure does not provide its customers a different way of managing the default encryption but allows users to add an optional second layer of encryption using a *customer-managed* key. The keys used to enable this additional security must be stored inside the Azure Key Vault, an Azure service designed to manage keys, certificates, and secrets in general like passwords or tokens.

To protect customers' data, Cosmos DB service automatically performs backups of all data at regular intervals. Similar to AWS, Microsoft offers two types of backup: *periodic backup mode* and *continuous backup mode*. The first approach is the default one. Azure executes backups periodically but to restore a specific backup is necessary to contact the support team. The user decides the interval of time between the backups. Instead, the second approach gives users the capability to restore data at any point in time during the last 30 days. Users can migrate from periodic backup to continuous back at any moment, but this migration is not reversible. In addition, also Cosmos DB allows developers to store multiple copies of their backups in one or more different geographical regions.

Discussing how to access the database, Cosmos DB provides three approaches to manage data access. Developers can use *primary/secondary keys*, *role-based access control* (RBAC), or *resource tokens*.

The first approach is the most powerful. It provides access to all the database account's administrative resources. The only possible limitation is to specify that keys are read-only. The purpose of having two keys is twofold. From one side, it allows users to keep using Azure services when they want to update a key. For example, users could temporarily use the secondary key without downtime while Azure is generating a new primary key. On the other side, users could give trusted partners the secondary key. If one of them, for any reason, is not trusted anymore, the users could use their primary key to easily replace the shared key.

The second approach is the most similar to the AWS IAM policies. After specifying an Azure Active Directory (AAD) identity, users can authenticate and authorize requests with fine-grained policies. Currently, Azure allows specifying three possible scopes: an account scope, a database scope, and a container scope. With an account scope, users can access all the databases of a specific account. With a database scope, it is possible to access specific databases of a particular account. Meanwhile, the container scope is the finest. In Azure, a container is a schema-agnostic set of items. So, users can define their containers and then provide them access using this approach.

The last approach, resource tokens, is used when users want to provide specific access to some resources to clients without giving them primary keys. These tokens can be built manually or generated using an Azure SDK.

In the end, we could say that even Azure appropriately manages the security of its databases by default with a good level of flexibility.

7 Discussion

Section 5 presented the research questions that led to the survey design. This section discusses how the collected responses address the questions, also contemplating the security features available on both platforms and discussed in Sect. 6. These two elements guided us in creating the guidelines presented in Sect. 8.

As regards the security perception of the involved elements, the database was indicated by the novice IoT developers as the most secure architecture item. Nevertheless, the interviewees recognized possible data loss (i.e., *information disclosure*) as one of the most critical issues. According to the obtained results, this security perception could have been quite dangerous considering that almost no one acted to verify if the data were encrypted at rest or not. Fortunately, in this case, both platforms have a default mechanism to protect the data stored inside NoSQL databases. Nonetheless, even if the platforms take care of this problem, programmers should remember to manage critical information properly. For instance, the importance of hashing the users' passwords in a real application is well-known. Unluckily, we did not include users' registration in our use case, so we did not collect any information on this specific behavior. However, we would like to remind readers that this protection mechanism has to be implemented by the developers (meanwhile, the analyzed platforms automatically manage it for their customers' accounts).

Contrarily, the less secure architecture points for our attendees are the data flows from (and to) the back-end. In more detail, the data flows from the back-end to the actuators are considered the most dangerous point if an attack succeeds. Specifically, the developers are mainly apprehensive about cyber-physical attacks. Considering so, it is particularly worrying that only a couple of developers verify the security of the communication channels, even because they can be targeted basically by all the security threats presented by the STRIDE model (Sect. 4.1). This fact increases our idea that unencrypted connections should not exist in a cloud-IoT scenario. Considering this specific issue, from one side, Amazon seems to disallow all the unencrypted connections between IoT devices and the back-end. On the other side, we observed that Azure left some unencrypted data flows available by default. From our point of view, in line with the survey's outcomes, the default configuration should not include unencrypted communications. Regarding the connections with the API gateways, Amazon does not allow to contact this service without protecting the communication with TLS. However, by default, all the versions of TLS are enabled, even the versions deprecated by IETF (currently TLS 1.0 and 1.1 [39]), which are more subject to *spoofing* and *tampering*. Contrarily, on Azure, the default configuration has only TLS v1.2 enabled. So, while Azure protection is adequate, AWS protection is sufficient on this component, but it could not be enough if the users do not act to change the default behavior.

Continuing our discussion, we noticed that although the surveyed developers think that cybersecurity is generally quite significant, they do not consider themselves proficient in this field. These data lead us to assume that the programmers did not pay particular attention to security threats during the development of their solutions due to the above-mentioned low self-confidence. However, it is also noticeable that no one thought to employ a supplementary tool to enhance the solution's robustness (e.g., enabling a service like AWS IoT Device Defender). From our point of view, counterbalancing their declared low knowledge with a service designed to help on that specific issue could have been a reasonable approach. Nevertheless, to the question: "Using AWS, who is responsible for the security of the system you develop?" no one replied that the responsibility is entirely assigned to the cloud platform. For two attendees, the solution's security is entirely a developer's responsibility; meanwhile, for the other participants, the responsibility is equally shared between the developer and the hosting platform. Hence, we can conclude that developers are aware of their role in keeping a cloud-IoT application secure.

Focusing more on the second and third research questions, once we explicitly asked the participants to evaluate whether the use case architecture could include security issues, all found at least one possible vulnerability. Nevertheless, during the development of their projects, no one declared to take any action to mitigate the problem. We could try to explain this behavior by considering that this project is the outcome of a training course, and cybersecurity aspects were not the central focus of our teaching sessions.

According to the retrieved answers, we can reply to the second research question by stating that, unfortunately, the participants did not pay particular attention to security threats during the development of their first cloud-IoT applications. Reasonably, we can infer that, as a direct consequence, they did not either act to increase their applications' cybersecurity. This behavior is a bit curious, considering that nobody tries to delegate the cloud platform for their security duties.

In addition, even though all the participants had created a strong password—probably because the platform policy forced them—almost no one had used the AWS IAM service to create separate accounts with different privileges. Using a unique account for all the possible operations could create many problems if a malicious user obtained the root account's password. For instance, with a root user it is very easy to tamper with the code executed on the back-end or the content of the database. Moreover, also the risk of information disclosure is very high. The fact that only one developer configured an IAM account increases our idea that it is necessary to drive users to take advantage of this kind of account. From our point of view, if the cloud platforms forced their customers to create at least a secondary account, users would consider more this particular functionality. Hence, a tutorial phase dedicated to this specific issue could help novice (IoT) programmers to understand why it is important to use a good privilege policy and the possible threats of a successful *elevation of privileges*.

To conclude, we discovered that developers did not underestimate their role in cybersecurity. However, as previously stated, they did not pay particular attention to the security issues during the development of their first cloud-IoT applications. Clearly, this low attention entails decreased actions to increase their applications' security. For these reasons, we believe that providing a set of straightforward guidelines could help developers think more about the cybersecurity of their applications and, consequently, take better countermeasures to reduce security threats.

8 Proposed guidelines

After considering the most relevant security issues according to our novice IoT programmers and analyzing how the two cloud platforms address these problems, we would like to provide a straightforward set of guidelines (GL) that could be helpful to design a more secure cloud-IoT application from the very beginning.

To begin with, we have to consider that IoT systems involve many different components like sensors, actuators, mini-pc, embedded boards, micro-controllers, cloud platforms, and many more. Due to the different elements included, developers usually have to handle many different technologies and programming languages. Logically, it is difficult for a novice programmer to approach this challenging environment, but even for a reasonably experienced programmer could be tough to develop an application in this context.

Many researchers have already demonstrated that following a security-by-design methodology is one of the best ways to achieve a good security level in many ICT applications. In particular, the importance of this approach was already discussed both in the cloud and in the IoT domains [52]. For this reason, we could say that such an approach is fundamental in a cloud-IoT application. Specifically, we developed this set of guidelines starting from the adoption of AWS in our use case but, taking into account also the additional platform, we believe that the reader could easily generalize the exposed concepts to other cloud-IoT platforms.

In this study, we concentrated our research activities mainly on the platforms' security (and their connections) without investigating too much the problems related to the IoT physical devices. However, this does not mean that *things* are less critical. Indeed, we are currently working on another study focused on understanding the most common security issues of such devices.

The first guideline we want to provide is applying a proper model or framework when developers start building a new IoT application. In particular, in our use case, we decided to apply the STRIDE framework [33] (already explained in Sect. 4.1). This framework is easy, consolidated, and already applied in several research activities [34–36]. We think it could be a good choice for developing cloud-IoT solutions having security in mind. Indeed, leading developers to think about security issues before starting the development of their applications would reduce the number of possible future threats.

GL1: Design a new application using a threat model from the beginning (e.g., STRIDE).

In the following sections, the paper discusses how to configure the components analyzed in Sect. 6.

8.1 Data flows and device protection

IoT systems have many possible connections transporting commands, measures, or relevant information. As we already discussed before, the data-flows are very much subject to the STRIDE threats, so protecting them is one of the more crucial aspects of such a system. One of the first steps to achieve a good level of protection is to disable the possibility of contacting the cloud back-end without encryption and authentication. It is also essential that the IoT devices communicate with each other using encrypted channels to reduce potential issues. This protection will avoid threats like *spoofing* or *tampering* and avoid *information disclosure* if a malicious user tries to eavesdrop on the data.

Table 5 Guidelines summary

ID	Guideline	Description
GL1	Use a threat model	Design a new application using a security model from the beginning (e.g., STRIDE).
GL2	Protect data in transit	Protect every data flow with encryption mechanisms.
GL3	Configure encryption correctly	Ensure that all encryption mechanisms are correctly configured; when using TLS, configure at least TLS 1.2, and select a *recommended* cipher suite.
GL4	Use platforms' SDKs	When available, always use the platforms' SDKs to connect an IoT device to the cloud platform.
GL5	Use platforms' support services	Always use services implemented by the platform to manage the devices.
GL6	Verify code security	Always use compiler features and code checkers to avoid insecure library functions or language constructs.
GL7	Verify authentication security	Always use two-factor authentication and/or complex passwords (e.g., at least two words, including at least three special characters).
GL8	Use secondary accounts	Always create secondary accounts (e.g., use *IAM accounts* in AWS and create other accounts in Azure).
GL9	Use POLP	Assign to each secondary account the fewest possible privileges.
GL10	Do not share accounts	Each developer must have his or her own account.
GL11	Protect data at rest	Always ensure that data protection at rest is enabled.
GL12	Apply more protections	Always apply Defense in Depth mechanisms.
GL13	Hash passwords	If the developed application involves user registration or authentication, salt and hash the passwords.
GL14	Have backups	Always enable at least periodic database backups (possibly even in different regions).

GL2: Protect every data flow with encryption mechanisms.

Considering TLS, as we already discussed in Sect. 6.1, the minimum version of TLS now considered safe is TLS 1.2. However, establishing the correct protocol version is not enough. Another critical step to achieve a good level of protection against possible attacks is to configure correctly the cipher suite used to exchange messages. To help developers in this task, we strongly suggest using tools like CipherSuite.info [40]. This tool is available through a website where everyone can search for a specific cipher suite and evaluate its effectiveness. The possible evaluations are as follows:

- *insecure*: do not use it under any circumstances;
- *weak*: old ciphers that should be disabled (use only in particular circumstances);
- *secure*: state-of-the-art ciphers;
- *recommended*: secure ciphers that also support Perfect Forward Secrecy.[21]

If developers are starting a new application from scratch, we strongly suggest choosing a *recommended* cipher suite. In this way, they will have the highest possible security. In any case, we strongly discourage the usage of an *insecure* or *weak* cipher suite. Even if programmers are re-configuring an already existent solution, they always should avoid using a suite belonging to these classes.

These configurations should be applied to all possible data flows in the project (e.g., from an IoT device to the back-end and vice versa). For instance, in our use case, we proposed implementing a device gateway based on Flask[22] for managing all the acting devices. To comply with what we discussed so far, developers have to configure this gateway to accept at least only TLS 1.2 communications. The same approach should also be adopted for the API gateways, especially if developers plan not to expose only public APIs.

GL3: Ensure that all encryption mechanisms are correctly configured; when using TLS, configure at least TLS 1.2, and select a *recommended* cipher suite.

Instead, considering the connection of the IoT devices with the back-ends, as we already discussed in Sect. 6.1, both platforms encourage connecting each IoT device to their back-end using the available SDKs. From one side, using these tools could simplify the connection process. On the other side, they could also reduce the possibility of introducing programming errors and consequently increase the application's security.

[21] A feature of specific key agreement protocols that ensures that session keys will not be compromised even if long-term secrets used in the session key exchange are compromised.

[22] A very widespread open-source web micro-framework developed in Python.

GL4: When available, always use the platforms' SDKs to connect an IoT device to the cloud platform.

Moreover, we observed that various scholars consider managing, fixing, and updating many IoT devices particularly difficult. So, they also proposed some possible solutions [53,54]. For this reason, when the number of connected devices is significant, we particularly suggest employing additional tools offered by the platforms. For instance, in AWS, instead of using the AWS IoT Core component, users can take advantage of the AWS IoT Device Management. In addition, to improve the security of the connected devices, we also suggest enabling support tools like the AWS IoT Device Defender or Microsoft Defender for IoT (especially if there are many devices connected).

GL5: Always use services implemented by the platform to manage the devices.

8.2 Back-end protection

We already discussed the security of the data-flows that allow reaching the back-end. The infrastructure security issues of the back-end are principally a duty of the cloud platforms, so we will not deeper investigate this part. From our point of view, the developers have essentially to face these three issues:

- write secure code inside the serverless computing service;
- choosing appropriate credentials for accessing the back-end;
- using a good privileges policy.

Writing secure code is a very complex and widespread research topic, and it could change accordingly to the used programming language. Suppose the developed code is not secure enough. In that case, malicious users could easily exploit this vulnerability to execute a *DoS* attack or, less easily, *tamper* with or obtain some relevant data (*information disclosure*). In some cases, they could even execute an *elevation of privileges* attack. Considering that both platforms support many programming languages, providing simple guidelines to follow on this specific topic is challenging. We can only suggest to developers not to neglect this relevant aspect and go deep into studying the desired programming language. Fortunately, nowadays, many code checkers can help developers write code in a more reliable and secure way. Some of the code checkers features could sometimes also be enabled directly inside compilers. Two possible examples are Snyk Code [55]—a tool that provides a static application security testing solution for scanning source code—and Upsource [56]—a code review and project analytic tool.

GL6: Always use compiler features and code checkers to avoid insecure library functions or language constructs.

Concerning the second issue, it is clear that choosing weak credentials could bring hazardous scenarios. A malicious user who owns root credentials can completely control the back-end and create all the threats presented in the STRIDE model. In Sect. 6.2, we discussed the password policies of both platforms. AWS had a more robust policy for the primary account, while the policy adopted for the secondary accounts is almost the same. However, both platforms are quite weak against dictionary attacks. Users could use tools like "How Secure is My Password?" [47] to check the robustness of their passwords, but they should be aware that respecting the criteria imposed by the cloud platforms (or by other associated websites) is not enough to create a truly secure password. Unfortunately, there are many misconceptions about password security [57]. For instance, it is always a bad idea to include in the password the website's name (i.e., Amazon, AWS, Microsoft, or Azure) user's personal information (e.g., name, age, born year, children's names, hometown, etc.). Indeed, we would recommend users to avoid always using easy to guess words and choose long passwords (even longer than the suggested length). Other researches demonstrated that using at least two words separated by other characters could be a good trade-off between security and usability [58]. Moreover, including (at least) three special characters in a password seems to be an excellent way to improve password entropy [59]. In conclusion, mixing these two approaches could be a perfect way to create a strong and quite easy-to-remember password.

In our analysis, we noticed that AWS did not enforce two-factor authentication of the accounts, while Azure enforced this policy (even if it allows users to delay the configuration of this additional security mechanism). However, users should always enable two-factor authentication when the platform provides this possibility, especially for root accounts.

GL7: Always use two-factor authentication and/or complex passwords (e.g., at least two words, including at least three special characters).

Regarding the last issue, users should apply what we reported so far both to the main account and to all the created secondary accounts. As we previously discussed in Sect. 6.2, even if AWS recommends using the root account mainly for creating IAM accounts, none of the two platforms enforce this recommendation in any way. We want to encourage developers to follow this best practice and create some secondary accounts applying the password strategy presented before. Moreover, an easy policy that novice devel-

opers could use is the Principle of Least Privilege [46]. Users should create a secondary account for each primary task and configure it to have only the privileges necessary to carry on the related activities. In this way, even if a malicious user compromises one of these accounts, the other functionalities could remain safe. For instance, in our use case, the API Gateway should be adequately configured, assigning to each endpoint the proper privileges, if users plan to expose not only public APIs. Moreover, multiple users must not share the same account, especially if they need access to different privileges or resources.

GL8: Always create secondary accounts (e.g., use *IAM accounts* in AWS and create other accounts in Azure).

GL9: Assign to each secondary account the fewest possible privileges.

GL10: Each developer must have their own account.

8.3 Database protection

A very relevant element to keep into consideration is the security of the data at rest stored in the databases. Indeed, if not adequately protected, data at rest could be easily *modified* or *disclosed*. Database encryption is essential to reduce damage in case of data leaks. Furthermore, the possibility of altering a table containing the list of registered users and administrators could create a *Denial of Service* or an unauthorized *elevation of privileges*. As we already discussed in Sect. 6.3, both Amazon DynamoDB and Azure Cosmos DB already provide an automatic encryption mechanism. If the developed application does not manage very sensible data, we suggest to the novice IoT developers simply use the default encryption provided by the platforms. However, However, suppose it is necessary (or reasonable) to have a more reliable encryption key or increase the security of this process. In that case, both solutions offer the possibility of improving the default encryption. If a user plans to use another cloud platform, they should verify if it offers default data at rest encryption or if they have to implement this feature by themselves.

GL11: Always ensure that data protection at rest is enabled.

When the application involves user authentication, in case a malicious user would be able to break the database cryptography, a very well-known Defense in Depth [60] strategy to apply is to add additional security to password storage. Usually, developers can achieve this protection through a hashing algorithm. Hashing users' password allows the application to work correctly, without explicitly storing actual passwords—which could be a hazardous disclosure in case of a data leak. There are many hashing algorithms that developers could use. We suggest following the recommendations of NIST or similar institutions. Currently, NIST suggests using at least an algorithm belonging to the SHA-2 family [50]. If possible, developers should consider using an algorithm from the SHA-3 family [51]. To further improve the security of the hash algorithm, it is also a good practice to add "salt" to passwords. This quite old technique consists of adding a random-generated string to each password's beginning or end before hashing [61]. Basically, instead of simply hashing the password `h(pwd)`, the program computes something like `h(pwd+salt)` or `h(salt+pwd)`. To further improve the security of this approach, programmers must use a different salt value for each stored password. These values are then associated with the correspondent users and stored (in clear text) in the back-end. According to some researchers, the suggested length for salt values is between 10 and 32 characters (i.e., from 80 to 256 bits) [62]. Moreover, to achieve an even higher level of security, developers can follow more sophisticated salt generation approaches. For instance, Boonkrong and Somboonpattanakit proposed an algorithm for generating and storing salt values directly from the passwords themselves [62].

GL12: Always apply Defense in Depth mechanisms.

GL13: If the developed application involves user registration or authentication, salt and hash the passwords.

Another good practice for improving the developed application's reliability and security is configuring a periodic database backup. Here we have two different scenarios. On one side, Microsoft forces users to have at least a periodic backup: no backup is not an option. On the other side, AWS does not automatically enable this configuration by default (also because it requires an additional payment). We strongly suggest always activating (at least) the point-in-time backup of the more sensitive tables.

GL14: Always enable at least periodic database backups (possibly even in different regions).

9 Conclusions

Starting from a preliminary study conducted on a small pool of novice IoT developers, this paper analyzes the more relevant security features available in two major cloud-IoT platforms. In particular, it highlights those settings, tools, and practices useful to achieve a higher level of reliability and security. Even if we noticed that the developers involved in the study did not correctly consider security issues in their IoT projects during the design and implementation phases, we observed that the platforms effectively support many of the highlighted security best practices and recommendations.

However, sometimes, developers have to identify and use the correct features to reach a proper level of protection. Indeed, after discussing the outcome of the survey and the relevant features of the two cloud-IoT platforms, this work contributes a set of guidelines to support novice IoT developers in reaching such protection. The final purpose of these guidelines is to avoid the primary and recurrent security issues in cloud-IoT projects and better exploit the inherent features of the cloud-IoT platforms.

9.1 Future works

In our future works, we would like to have a more focused survey on a larger sample of novice IoT programmers to know more about the security perception of this class of developers. In addition, to complete the view of the cloud-IoT environments, we also are interested in investigating the most common security issues in IoT devices and IoT gateways. For this reason, we are now reviewing many novice programmers' IoT projects developed for widespread IoT boards as Arduino-like devices. We are also investigating the security of a famous open-source smart home gateway: Home Assistant.

Acknowledgements We want to acknowledge the employees who voluntarily decided to participate in the survey to conduct this research activity.

References

1. Deepak K, Kelly S, Benton C, Deepali G, Galina A, Dmitry K, Rajarshi G, Zakir D (2019) All things considered: an analysis of IoT devices on home networks. In: 28th USENIX security symposium (USENIX Security 19), pp 1169–1185. ISBN 978-1-939133-06-9. https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-deepak
2. Raj B, Bob G, Dennis S, Kevin J, David W (2021) Magic quadrant for cloud infrastructure and platform services. Technical report, Gartner Inc., July. https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802
3. Manos A, Tim A, Michael B, Matt B, Elie B, Jaime C, Zakir D, Alex HJ, Invernizzi L, Michalis K, Deepak K, Chaz L, Zane M, Joshua M, Damian M, Chad S, Nick S, Kurt T, Zhou Y (2017) Understanding the mirai botnet. In: 26th USENIX security symposium (USENIX Security 17), pp 1093–1110, Vancouver, BC, August. USENIX Association. ISBN 978-1-931971-40-9. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis
4. CVE-2021-28372. Available from MITRE, CVE-ID CVE-2021-28372., March 13 2021. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-28372
5. Corno F, De Russis L, Mannella L (2021) Perception of security issues in the development of Cloud-IoT systems by a novice programmer. In: Intelligent environments 2021, pp 5–15. IOS Press. https://doi.org/10.3233/AISE210074
6. Borislav SD, Jovanović SP, Timčenko VV (2014) Cloud computing in amazon and microsoft azure platforms: performance and service comparison. In: 2014 22nd telecommunications forum Telfor (TELFOR), IEEE, pp 931–934. https://doi.org/10.1109/TELFOR.2014.7034558
7. Montanaro T, Sergi I, Limelli S, Patrono L (2021) Fog computing: implementation of a simple fog scenario through iot public services. In: 2021 6th international conference on smart and sustainable technologies (SpliTech), pp 1–6. https://doi.org/10.23919/SpliTech52315.2021.9566323
8. Barcelona-Pons D, García-López P (2021) Benchmarking parallelism in faas platforms. Futur Gener Comput Syst 124:268–284. https://doi.org/10.1016/j.future.2021.06.005 (**ISSN 0167-739X.**)
9. Brodkin J (2008) Gartner: seven cloud-computing security risks. Technical report, Network World. https://www.infoworld.com/article/2652198/gartner-seven-cloud-computing-security-risks.html
10. Mariana C, Van Der Merwe A, Kotzé P (2011) Secure cloud computing: benefits, risks and controls. In: 2011 information security for South Africa, IEEE, pp 1–9. https://doi.org/10.1109/ISSA.2011.6027519
11. Akhil B, Kanika B (2012) An analysis of cloud computing security issues. In: 2012 world congress on information and communication technologies, IEEE, pp 109–114. https://doi.org/10.1109/WICT.2012.6409059
12. Elliot S, James SC (1989) Studying the novice programmer. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey. https://doi.org/10.4324/9781315808321
13. Hank K (1983) What do novice programmers know about recursion. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI '83, New York, NY, USA. Association for Computing Machinery, pp 235–239. ISBN 0897911210. https://doi.org/10.1145/800045.801618
14. Lahtinen Essi, Ala-Mutka Kirsti, Järvinen Hannu-Matti (2005) A study of the difficulties of novice programmers. ACM SIGCSE Bull 37(3):14–18. https://doi.org/10.1145/1151954.1067453 (**ISSN 0097-8418.**)
15. Billy J (2021) Understanding their voices from within: difficulties and code comprehension of life-long novice programmers. Int J Arts Sci Educ 1(1):53–73
16. Corno F, De Russis L, Sáenz JP (2017) Pain points for novice programmers of ambient intelligence systems: an exploratory study. In 2017 IEEE 41st annual computer software and applications conference (COMPSAC), vol 1. IEEE, pp 250–255. https://doi.org/10.1109/COMPSAC.2017.186
17. Corno F, De Russis L, Sáenz JP (2019) Towards computational notebooks for IoT development. In: Extended abstracts of the 2019 CHI conference on human factors in computing systems, CHI EA '19, New York, NY, pp 1–6. Association for Computing Machinery. ISBN 9781450359719. https://doi.org/10.1145/3290607.3312963

18. Corno F, De Russis L, Sáenz JP (2021) On computational notebooks to empower physical computing novices. In: Companion of the 2021 ACM SIGCHI symposium on engineering interactive computing systems, EICS '21, New York, NY, pp 22–25. Association for Computing Machinery. ISBN 9781450384490. https://doi.org/10.1145/3459926.3464752
19. Kamoun F, Halaweh M (2012) User interface design and e-commerce security perception: an empirical study. Int J E-Bus Res (IJEBR) 8(2):15–32. https://doi.org/10.4018/jebr.2012040102
20. Varga S, Brynielsson J, Franke U (2021) Cyber-threat perception and risk management in the swedish financial sector. In: Computers & security, p 102239. ISSN 0167-4048. https://doi.org/10.1016/j.cose.2021.102239
21. Kafle Kaushal, Moran Kevin, Manandhar Sunil, Nadkarni Adwait, Poshyvanyk Denys (2020) Security in centralized data store-based home automation platforms: a systematic analysis of nest and hue. ACM Trans Cyber-Phys Syst 5(1):1–27. https://doi.org/10.1145/3418286 (**ISSN 2378-962X.**)
22. Shantanu P, Michael H, Tahiry R, Subhas M (2020) Security requirements for the internet of things: a systematic approach. Sensors 20(20). ISSN 1424-8220. https://doi.org/10.3390/s20205897
23. Momenzadeh B, Dougherty H, Remmel M, Myers S, Camp JL (2020) Best practices would make things better in the IoT. IEEE Security Privacy 18(4):38–47. https://doi.org/10.1109/MSEC.2020.2987780
24. Anthi E, Williams L, Słowińska M, Theodorakopoulos G, Burnap P (2019) A supervised intrusion detection system for smart home IoT devices. IEEE Internet Things J 6(5):9042–9053. https://doi.org/10.1109/JIOT.2019.2926365
25. Amazon Web Services Inc (2020) Introduction to AWS security. Technical report, Amazon Web Services Inc., North Seattle, WA. https://d1.awsstatic.com/whitepapers/Security/Intro_to_AWS_Security.pdf
26. Amazon Web Services Inc (2020) Amazon web services: overview of security processes. Technical report, Amazon Web Services Inc.,North Seattle, WA . https://d0.awsstatic.com/whitepapers/aws-security-whitepaper.pdf
27. Saakshi N, Arushi J et al (2015) Cloud computing security: Amazon web service. In: 2015 Fifth international conference on advanced computing & communication technologies, IEEE, pp 501–505. https://doi.org/10.1109/ACCT.2015.20
28. Marta T, Bradley R, Patrick W (2020) Amazon web services: risk and compliance. Technical report, Amazon Web Services Inc., North Seattle, WA. https://d1.awsstatic.com/whitepapers/compliance/AWS_Risk_and_Compliance_Whitepaper.pdf
29. Navneet B, Abhik B, Agniswar R (2021) Case study of azure and azure security practices. In: Machine learning techniques and analytics for cloud security, p 339. https://doi.org/10.1002/9781119764113.ch16
30. Frank S, Eric T (2019) Shared responsibility for cloud computing. Technical report, Microsoft Corporation, Redmond, WA. https://azure.microsoft.com/it-it/resources/shared-responsibilities-for-cloud-computing/
31. Souad A, Safae C, Salma M (2018) Which nosql database for iot applications? In: 2018 international conference on selected topics in mobile and wireless networking (MoWNeT), pp 131–137. https://doi.org/10.1109/MoWNet.2018.8428922
32. Huang DY, Apthorpe N, Li F, Acar G, Feamster N (2020) IoT inspector: crowdsourcing labeled network traffic from smart home devices at scale. Proc ACM Interact Mobile Wearable Ubiquit Technol 4(2):1–21. https://doi.org/10.1145/3397333
33. Shawn H, Scott L, Tomasz O, Adam S (2006) Threat modeling-uncover security design flaws using the stride approach. In: MSDN magazine-Louisville, pp 68–75
34. Rafiullah K, Kieran M, David L, Sakir S (2017) STRIDE-based threat modeling for cyber-physical systems. In: 2017 IEEE PES innovative smart grid technologies conference Europe (ISGT-Europe), IEEE, pp 1–6. https://doi.org/10.1109/ISGTEurope.2017.8260283
35. Nadeem AM, Mohammed N, Mansoor AA (2020) Modeling security threats for smart cities: a stride-based approach. In: Smart cities-opportunities and challenges. Springer, pp 387–396 . https://doi.org/10.1007/978-981-15-2545-2_33
36. Bojan J, Daniela R, Imre L, Marina S, Sebastijan S (2017) STRIDE to a secure smart grid in a hybrid cloud. In: Computer security. Springer, pp 77–90. https://doi.org/10.1007/978-3-319-72817-9_6
37. Christopher A, Tim D (1999) The TLS protocol version 1.0. RFC 2246, January . https://doi.org/10.17487/RFC2246
38. Eric R (2018) The transport layer security (TLS) protocol version 1.3. RFC 8446, August. https://doi.org/10.17487/RFC8446
39. Kathleen M, Stephen F (2021) Deprecating TLS 1.0 and TLS 1.1. RFC 8996, March. https://doi.org/10.17487/RFC8996
40. Christian RH, Nils G (2022) Ciphersuite. https://ciphersuite.info/. [Online: Accessed 03 Feb 2022]
41. OASIS. MQTT version 5.0 documentation, 2022. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. [Online: Accessed 27 Jan 2022]
42. Postman Inc. Postman, 2022. https://www.postman.com/. [Online: Accessed 24 Jan 2022]
43. Elaine B, Allen R (2019) Transitioning the use of cryptographic algorithms and key lengths, 2019-03-21 . https://doi.org/10.6028/NIST.SP.800-131Ar2
44. Marc S, Elie B, Pierre K, Ange A, Yarik M (2017) The first collision for full sha-1. In: Katz J, Shacham H (eds) Advances in cryptology–CRYPTO 2017. Springer International Publishing, Cham, pp 570–596
45. Karen S, Stephen K (2005) Security architecture for the internet protocol. RFC 4301, December. https://doi.org/10.17487/RFC4301
46. Schneider FB (2003) Least privilege and more [computer security]. IEEE Security Privacy 1(5):55–59. https://doi.org/10.1109/MSECP.2003.1236236
47. Security.org. Online tool: How secure is my password? (2022). https://www.security.org/how-secure-is-my-password/. [Online: Accessed 19 Jan 2022]
48. Blase U, Gage KP, Saranga K, Joel L, Michael M, Mazurek ML., Timothy P, Richard S, Timothy V, Lujo B, Nicolas C, Faith CL (2012) How does your password measure up? the effect of strength meters on password creation. In: 21st USENIX security symposium (USENIX security 12), Bellevue, WA, August. USENIX Association, pp 65–80. ISBN 978-931971-95-9. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/ur
49. Morris D, Elaine B, James N, James F, Lawrence B, Roback E, James D (2001) Advanced encryption standard (aes), 2001-11-26. https://doi.org/10.6028/NIST.FIPS.197
50. Quynh D (2015) Secure hash standard, 2015-08-04 . https://doi.org/10.6028/NIST.FIPS.180-4
51. Dworkin MJ (2015) Sha-3 standard: permutation-based hash and extendable-output functions, 2015-08-04. https://doi.org/10.6028/NIST.FIPS.202
52. Sequeiros João BF, Chimuco Francisco T, Samaila Musa G, Freire Mário M, Inácio Pedro RM (2020) Attack and system modeling applied to IoT, cloud, and mobile ecosystems: embedding security by design. In: ACM Comput. Surv., vol 53(2), March . ISSN 0360-0300. https://doi.org/10.1145/3376123
53. Antonio L, Alberto BC, Markus S, Kay R (2019) UpKit: an open-source, portable, and lightweight update framework for constrained IoT devices. In: 2019 IEEE 39th international conference on distributed computing systems (ICDCS), pp 2101–2112. https://doi.org/10.1109/ICDCS.2019.00207

54. Georgios S, Rui W, Roel M, Geert-Jan S, Mario M, Stefan I, Willems Frans MJ, Lieneke K(2020) RESCURE: a security solution for IoT life cycle. In: Proceedings of the 15th international conference on availability, reliability and security, ARES '20, New York, NY, USA. Association for Computing Machinery. ISBN 9781450388337. https://doi.org/10.1145/3407023.3407075
55. Snyk. Snyk code, 2022. https://snyk.io/product/snyk-code/. [Online: Accessed 03 Feb 2022]
56. Jetbrain. Upsource, 2022. https://www.jetbrains.com/upsource/. [Online: Accessed 03 Feb 2022]
57. Blase U, Fumiko N, Jonathan B, Segreti SM, Richard S, Lujo B, Nicolas C, Faith CL (2015) "I added '!' at the end to make it secure": observing password creation in the lab. In: Eleventh symposium on usable privacy and security (SOUPS 2015)
58. Richard S, Saranga K, Durity AL, Phillip (Seyoung) H, Mazurek ML, Segreti SM, Blase U, Lujo B, Nicolas C, Faith CL (2014) Can long passwords be secure and usable? In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI '14, pp 2927–2936, New York, NY, USA. Association for Computing Machinery. ISBN 9781450324731. https://doi.org/10.1145/2556288.2557377
59. Wanli M, John C, Dat T, Dale K (2010) Password entropy and password quality. In: 2010 Fourth international conference on network and system security, IEEE, pp 583–587. https://doi.org/10.1109/NSS.2010.18
60. Smith CL (2003) Understanding concepts in the defence in depth strategy. In: IEEE 37th annual 2003 international Carnahan conference on security technology, 2003. Proceedings. IEEE, pp 8–16. https://doi.org/10.1109/CCST.2003.1297528
61. Morris R, Thompson K (1979) Password security: a case history. Commun ACM 22(11):594–597. https://doi.org/10.1145/359168.359172 (**ISSN 0001-0782.**)
62. Sirapat B, Chaowalit S (2016) Dynamic salt generation and placement for secure password storing. IAENG Int J Comput Sci 43(1):27–36