Rethinking the Security of IoT from the Perspective of Developer Customized Device-Cloud Interaction

Anonymous Author(s)

ABSTRACT

To achieve powerful inter-connection in an IoT system, IoT cloud involved communication is widely deployed due to its various services. While IoT clouds provide many strong security mechanisms, lots of IoT devices are still configured with vulnerable interaction procedures with clouds. And the root cause of these security issues in such procedures is less understood today.

In this paper, we report a practical study of IoT cloud security regulations on IoT device development solutions. Instead of checking real-world IoT devices, we explore IoT security, especially the security of interaction between devices and clouds, from the perspective of IoT developers and illustrate why flaws during development could evolve into vulnerabilities on real-world devices. To better understand the problem, we describe three security-critical device options that can be customized by IoT developers with their own features, and propose a new approach to check whether a specific IoT cloud regulates the security of IoT solutions with vulnerable device options. With the evaluation of eight mainstream IoT cloud platforms, our study brings to the pervasiveness of the security hazards in IoT cloud security regulation during IoT device development, resulting in manufacturing vulnerable IoT devices.

KEYWORDS

IoT Security, IoT Development, Cloud Regulation

ACM Reference Format:
Anonymous Author(s). 2022. Rethinking the Security of IoT from the Perspective of Developer Customized Device-Cloud Interaction. In *Proceedings of ACM SAC Conference (SAC'22).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Components of IoT systems (IoT devices, IoT companion apps, etc.) prefer to interact with each other under the help of IoT cloud. In this process, the security of interaction between IoT devices and IoT clouds (*device-cloud interaction* for short) is especially important. There are severe security threats [1, 5, 13, 27] against this procedure. For instance, if the attacker can access the communication channel between the IoT device and the IoT cloud, then he may also be able to conduct relevant attacks, such as device data tampering and user privacy leakage. Furthermore, if the authentication mechanism for the device-cloud interaction is vulnerable, the attacker then could launch device impersonation attacks.

SAC'22, April 25 –April 29, 2022, Brno, Czech Republic
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-8713-2/22/04…$15.00
https://doi.org/xx.xxx/xxx_x

Typically, an IoT cloud provides multiple protective means [43, 48, 58, 59, 65] for the device-cloud interaction to avoid potential threats. First, IoT cloud usually supports device identification mechanisms to prevent malicious ones faked by attackers. And each IoT device is configured with an identifier as its unique identity. This makes attackers unable to impersonate as the legitimate devices as long as they can not learn about these identifiers. Second, various data confidentiality protection strategies are provided to avoid data leakage during device-cloud interaction, including Transport Layer Security (TLS), Datagram Transport Layer Security (DTLS), and even encryption in application protocols. These protection strategies ensure that the transmitted data would not be leaked to an attacker just through passive eavesdropping. Third, some IoT clouds implement anti-replay schemes for the device-cloud interaction. These schemes usually involve values tagging each message only or valid in a short time period so that there is enough time for an attacker to intercept, re-organize and replay the messages to achieve specific operations.

Unfortunately, despite powerful security mechanisms provided by IoT clouds, vulnerable device-cloud interaction is still common in real-world IoT system. Prior studies [43, 44, 62, 65] have demonstrated various security issues in device-cloud interaction, such as flaws in IoT application layer protocols, communication deciphering and privacy leakage through traffics. Moreover, we found that most existing security analysis methods on IoT systems [48, 51, 52, 57, 61, 62] focused on specific IoT devices (e.g., wearable health trackers, smart toys) or specific implementations (e.g., remote binding, MQTT protocols), but a systematic study to explore the root cause of IoT security issues is generally missing. Hence, in this paper, we trace back to one of the initial phases of an IoT system — the IoT device development (*IoT development* for short). During this phase, IoT developers are usually allowed to customize their IoT device development solutions (*IoT solutions* for short) to adapt to the device resources and application requirements. Since these IoT solutions, involving various device functions, are built to develop IoT devices, they are directly relevant to real-world device-cloud interaction security. If insecure IoT solutions are not checked and forbidden during IoT development, vulnerabilities will be eventually introduced into real-world IoT devices. In response, in order to prevent such insecure solutions, most IoT cloud providers claim they have implemented security audit on developer-customized IoT solutions. But since the audit procedures on IoT clouds usually act as black-boxes, whether an IoT cloud would regulate the security of customized IoT solutions is still less understood today.

In this paper, we tend to reveal the security of cloud regulations on the IoT solutions during IoT development. We have a new approach from the perspective of IoT developers rather than real-world device audit. In particular, we have observed IoT clouds usually provide various device options for IoT developers to configure on their own ways. These device options are configured in an IoT solution and synchronized on the IoT cloud, and the security of IoT solutions mostly depends on the security of customized device options: if these options are vulnerable, the IoT solution, as well as the developed device, will be insecure. Therefore, we aim to check whether a specific IoT cloud accepts an insecure solution with vulnerable device options.

To achieve our goal, we first described three security-critical customized device options that would directly affect the security of IoT solutions, especially the device-cloud interaction procedure. Then, we proposed a practical approach about how to build an insecure solution. Finally, we conducted a series of practical tests on eight mainstream IoT clouds in United States and China to check whether they make strict security regulations on these insecure IoT solutions. Our testing demonstrated a non-optimistic result: all the eight IoT clouds accepted insecure IoT solutions with vulnerable customized device options, indicating that they all did not implement comprehensive security regulations on IoT solutions. We observed that even though GoogleCloud, the only IoT cloud that adopted relatively comprehensive security checks, still ignored the vulnerable device identifier issue. More seriously, three IoT clouds, i.e., Aliyun, QcloudE and QcloudH, never applied any security regulations according to our tests. As a result, IoT devices developed with insecure solutions on these IoT clouds will be also vulnerable. And the attacker can easily perform attacks against real-world devices, such as device impersonation, Man-in-the-Middle (MITM) and replay attacks, to steal sensitive device data, leak user privacy, and even hijack IoT devices.

2 BACKGROUND

Device-cloud interaction is vital to IoT ecosystem and the IoT developers have to implement and configure corresponding interaction functions during IoT development. In this section, we first briefly present some basic knowledge of IoT system. Then we discuss the related works about IoT security analysis, and introduce our motivation.

2.1 Definitions

We first define essential concepts in IoT system composition, development and system interconnection. Figure 1 illustrates the key components and procedures in an IoT system.

IoT system composition. Three components, *IoT device*, *IoT user*, and IoT cloud platform (*IoT cloud* for short) managed by the IoT vendors or other third parties, are commonly involved in an IoT system. An IoT device is deployed for data synchronization (i.e., collecting data from sensors and synchronizing the data with end-users) or command processing (i.e., manipulate the devices based on user commands). An IoT user is the device owner or actual controller who is able to send commands to operate the IoT device. An IoT cloud is involved to interact with both the deployed IoT device and the user for data transmission, data storage, data processing and even identity authentication. And some IoT clouds providing data visualization are also used to manage IoT devices, such as firmware updating and device status monitoring.

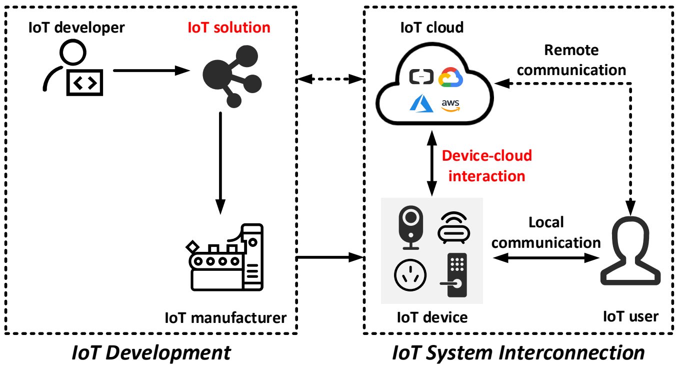

Figure 1: An overview of components in an IoT system

IoT development. In this paper, we refer the *IoT development*, the initial phase of IoT system design and implementation, as the procedure to build an *IoT solution*, which involves both implementing software components (e.g., various device services and hardware-related functions) and integrating them into device hardware parts. Typically, two main roles are played in the IoT development. *IoT manufacturers* refer to IoT companies that develop and produce the IoT devices in large quantities through pipelining. And *IoT developers* refer to the actual workers who design and implement the detailed device functions in IoT solutions. Specially, an IoT device should be developed with functions to interact with specific IoT clouds for more advanced cloud services, e.g., device data analysis and remote control. Moreover, IoT developers usually develop the IoT devices based on the device application scenarios and IoT manufacturers' requirements, such as energy, memory, security and other manufacturer-featured options.

IoT system interconnection. In a typical IoT scenario, a user is able to control her device via *local communication* and *remote communication*. The local communication refers to the user and the device directly communicate with each other through same Local Area Network (LAN) [25] while remote communication refers to the user and the device can communicate with each other under the help of IoT cloud even they connect to different LANs. Obviously, the remote communication involving the IoT cloud is more flexible to apply in common scenarios to connect various terminal devices in different networks, in comparison to local communication. Moreover, in remote communication, the interaction between the user and IoT cloud can be optional but the interaction between the device and IoT cloud, namely *device-cloud interaction*, will never be ignored. For instance, the user who is acted by an IoT developer may directly manage and control the device on the IoT cloud console. Hence, maintaining and analyzing the security of device-cloud interaction is extremely critical due to its extensive application and necessity.

2.2 Security Analysis Methods for IoT Ecosystem

Our paper focus on the device-cloud interaction security. In the following, we first discuss the existing security analysis studies for IoT ecosystem and then introduce our motivation.

Some existing works have been done to analyze the security of IoT devices. Some works utilized device firmware to analyze device security. Shoshitaishvili *et al.* [60] proposed a model of authentication bypass flaws and designed a binary analysis framework, Firmalice, to identify the occurrences of these vulnerabilities automatically on embedded device firmware. Costin *et al.* [49] performed a large-scale static analysis of firmware images, while Kim *et al.* [53] presented a dynamic analysis of firmware in scale via arbitrated emulation. And David *et al.* [50] proposed a static analysis to find CVEs in stripped firmware images by finding similarity over procedures. However, IoT manufacturers usually do not publish information about device firmware files as well as their composition. And even sometimes firmware can only be obtained by extracting from real IoT devices, making analysis consuming, inefficient and difficult to cover a large scope of devices. Some other works studied the IoT device security via IoT companion apps. Wang *et al.* [63] demonstrated companion mobile apps for IoT devices could reflect the security of device themselves. Chen *et al.* [47] presented and implemented an automotive fuzzing framework, IOTFUZZER, to find memory corruption vulnerabilities in IoT devices without firmware. And Redini *et al.* [56] designed another fuzzing tool, DIANE, which utilized fuzzing triggers to fuzz IoT devices automatically. For these studies, they only focused on interaction between device and apps (users) without involvement of IoT clouds, so that they were unable to evaluate the security of device-cloud interaction.

Prior studies also have been focusing on the authentication and communication security in IoT systems. Srinivasa *et al.* [61] investigated the exploitation of misconfigured IoT devices by IoT honeypots deployment and network telescope traffic analysis. Jia *et al.* [52] performed a systematic security study about the implementations of the general messaging protocol, MQTT, on the protection of IoT platform. Chen *et al.* [48] reported the analyses of IoT remote binding security and discussed various practical designs as well as the attack surfaces. Zhou *et al.* [65] described the security analysis of the interactions between IoT devices, mobile apps, and clouds in smart home scenarios and focused on weaknesses in state transition diagrams of the three entities. Apthorpe *et al.* [44] demonstrated that privacy sensitive in-home activities could be analyzed by smart home device traffics even they were encrypted. Valente *et al.* [62] analyzed the security and privacy practices of smart toys and showed attackers could decrypt communication and inject audio. Unfortunately, all these studies are based on released commercial products and just focused on several typical types of IoT devices and their specific implementations. The studied devices are sometimes not representative to reflect the security issues in a wider range, and they cannot help analysts understand why flaws were introduced by developers.

3 METHODOLOGY

In this paper, we aim to check whether an insecure IoT solution (due to the improper customization of developers) could be applied to popular IoT clouds, and hence the use of such insecure solution could lead to potential security breaches. We extend existing researches on device-cloud interaction security from a novel perspective — IoT development, the initial phase of an IoT device life cycle. We summarize two major reasons that vulnerabilities are easily introduced to the real-world IoT devices: 1) IoT developers design and build insecure IoT solutions; 2) IoT clouds do not conduct strict regulations on the security of these customized IoT solutions.

To systematically study how insecure IoT solutions are built and whether IoT clouds offer detailed security regulations to prevent insecure customized IoT solutions, we propose a new test approach (as illustrated in Figure 2). Generally, our approach can be divided into three main phases: ❶ **generating vulnerable device options**; ❷ **building insecure IoT solutions**; ❸ **testing security regulations in IoT clouds**. This approach requires the analyst to play the role of typical IoT developers and utilize the existing infrastructures (e.g., IoT SDKs, development documents and cloud servers, etc.) provided by IoT clouds, attempting to build insecure device-cloud interaction solutions by enumerating vulnerable options of **device identifier**, **data transmission** and **anti-replay consideration**, which are likely to be introduced by developers during IoT development. After that, the approach tests whether a specific IoT cloud accepts any of the insecure solutions. In this way, one could comprehensively examine to what extent the IoT cloud regulates the security of its belonging devices.

3.1 Customized Device Options

Before building insecure solutions, we first have to clarify what affects the device-cloud interaction security in an IoT solution. Hence, we have investigated the development procedures of mainstream IoT clouds, and found that in order to satisfy various resource and application requirements of different IoT manufacturers, an IoT cloud usually provides several device options for IoT developers to customize their own IoT solutions. Some of these options are closely relevant to the device-cloud interaction security. And if these options are configured insecurely, the IoT solutions, as well as the developed IoT devices, will be likely to be introduced vulnerabilities.

Various device options are usually provided for IoT developers but not all device options could directly affect device-cloud interaction security of real IoT devices. So we first manually collected the customized device options from the documents (i.e., official development references and code demos) of mainstream IoT cloud platforms and then filtered those directly linked to the device-cloud interaction security properties such as identity authentication and data confidentiality. Typically, there are three security-critical device options recognized, i.e., device identifier, data transmission and anti-replay consideration.

- **Device identifier.** A device identifier is a unique string to identify an IoT device's identity and different devices have different device identifiers. The IoT cloud usually allows IoT developers to generate and apply the device identifiers with their own features. For example, some developers directly use the device MAC address as the device identifier while some other developers prefer containing their manufacturer's name in the device identifier.
- **Data transmission.** The data transmission options refer to device-cloud communication protocols and IoT developers are usually allowed to choose their favorable one based on the security requirements and application scenarios. For instance, data transmission options about various

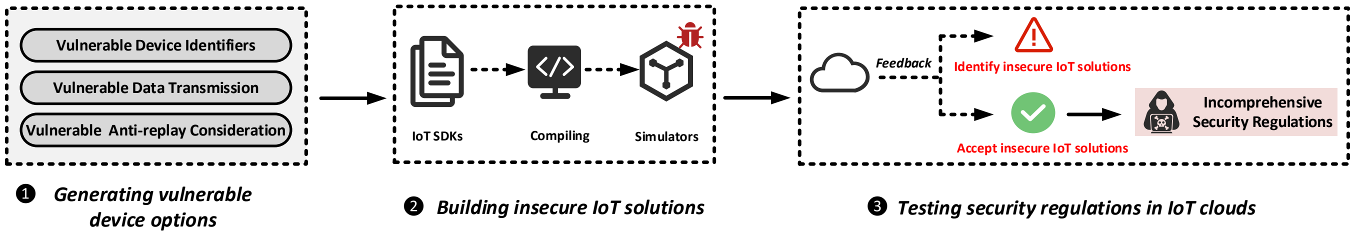
Vulnerable Device Identifiers
Vulnerable Data Transmission
Vulnerable Anti-replay Consideration
IoT SDKs
Compiling
Simulators
Feedback
Identify insecure IoT solutions
Accept insecure IoT solutions
Incomprehensive Security Regulations
❶ Generating vulnerable device options
❷ Building insecure IoT solutions
❸ Testing security regulations in IoT clouds

Figure 2: Our approach to check whether IoT clouds regulate customized IoT solutions.

kinds of application protocols (e.g., MQTT [28], CoAP [15] and HTTP [20]) and data protection protocols (e.g., Transport Layer Security (TLS) protocols [41]) are provided for developers.

- **Anti-replay consideration.** Anti-replay consideration options are provided sometimes for developers to determine whether applying an anti-replay scheme to resist replay attacks and to determine how the anti-replay scheme is implemented. Typically, the IoT cloud will provide some fields for IoT developers to complete the anti-replay options. Developers are able to configure these fields to contain anti-replay values and enable the cloud to check them.

3.2 Generating Vulnerable Device Options

Based on three device options that are security-critical to device-cloud interaction, we then describe how to generate vulnerable options that will result in insecure IoT solutions. Moreover, these vulnerable options are also common in real-world IoT development, in which the developers lack of enough security knowledge. Hence, we utilize seven strategies of three types to help generate vulnerable options as follows.

Vulnerable device identifiers. A secure device identifier should be able to prevent brute force [14], guessing [26, 38] and impersonation attacks [48, 64, 65], that means the device identifiers should be constrained on the length, randomness and uniqueness. Referring to security consideration for OWASP [32] and OAuth 2.0 [30], a device identifier should be generated with a part constructed from a strong cryptographic pseudo-random number generator (PRNG) and the probability of an attacker guessing generated identifier must be less than or equal to 2^{-128} and should be less than or equal to 2^{-160}. Furthermore, it should be unique that represents only one device and other devices cannot obtain it. So in view of both usability and security considerations, we consider that an IoT device identifier should contain an at least 128-bit unique pseudo-random number. Correspondingly, we generate vulnerable device identifier options with three strategies: short-length device identifiers, guessable device identifiers and non-unique device identifiers.

- *Short-length device identifiers.* To examine whether an IoT cloud checks the length of device identifiers to avoid brute force attacks, we use pseudo-random strings with no more than 15 bytes as the device identifiers. Specifically, we separately generate 15 pseudo-random strings, which are combinations of alphanumeric characters. And the length of each string is from 1 to 15. Then we apply these strings into IoT solutions as device identifiers.
- *Consecutive device identifiers.* To examine IoT cloud regulations on the randomness of device identifiers to avoid guessable attacks, we use 10 consecutive strings as device identifiers. In particular, we generate a 16-byte pseudo-random string of alphanumeric characters firstly. Then we randomly choose one byte of this string and separately change its value by adding `0x01` to generate the other 9 strings. And these 10 strings only differ slightly by only one sequential byte, such as `dev00a`, `dev00b`, ... , `dev00j`. This kind of identifiers are common in IoT development because some developers prefer to directly utilizing the device MAC addresses [26] or serial numbers [38] as their device identifiers for operating easily but both of them are usually consecutive strings.
- *Non-unique device identifiers.* To check whether the IoT cloud could prevent non-unique device identifiers to avoid device impersonation attacks, we utilize one string as multiple device identifiers. The string is pseudo-randomly generated with alphanumeric characters, 16 bytes in length. And it is then configured as the device identifier in multiple IoT solutions, indicating that more than one developed devices would be configured with same identifiers.

Vulnerable data transmission. IoT developers are allowed to implement various application protocols, but all the data transmission should be protected with end-to-end encryption to prevent MITM attacks. Referring to RFC6749 [30] and RFC7519 [23] documents that regulate OAuth 2.0 authorization framework and JSON web token implementations, RFC7457 [40] about known attacks on TLS as well as NSA [29] and NIST [19] Guidance on TLS protocols, the transmitted data during device-cloud interaction should be protected by TLS protocols with the relatively secure versions, i.e., TLS version 1.2 (TLSv1.2) or higher. And either data transmission without TLS protection or adopting obsolete TLS protocols is vulnerable to MITM attacks.

We generate the vulnerable data transmission options with two strategies. Firstly, to check whether an IoT cloud forces the device-cloud interaction protected by TLS encryption, we implement the data transmission without TLS protection enabled. Then, we adopt the old versions of TLS protocols for device-cloud interaction, i.e., SSL and TLSv1.0/1.1, to examine whether the IoT cloud only accepts connections with secure versions of TLS protocols.

Vulnerable anti-replay consideration. Anti-replay schemes should be adopted to prevent replay attacks [42, 45, 54]. Otherwise, the

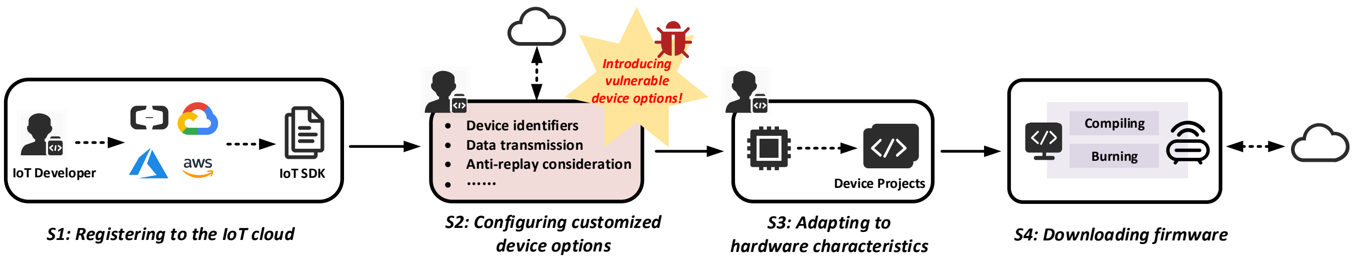
IoT Developer
aws
IoT SDK
S1: Registering to the IoT cloud
Device identifiers
Data transmission
Anti-replay consideration
……
Introducing vulnerable device options!
S2: Configuring customized device options
Device Projects
S3: Adapting to hardware characteristics
Compiling
Burning
S4: Downloading firmware

Figure 3: The procedure of building an IoT solution

attacker would be able to trick the devices or clouds by elaborately intercepting and re-transmitting specific messages. There are three common anti-replay schemes, i.e., timestamp, nonce and session token, in which timestamp and session token should be only valid in a short period[*], and nonce should be used only once during a connection. Thus, to avoid replay attacks, not only adopting the anti-replay schemes but also implementing proper anti-replay values are required.

Correspondingly, we adopt two strategies to implement the vulnerable anti-replay options. In order to examine whether an anti-replay scheme is required by the IoT cloud to avoid replay attacks, we first modify the anti-replay fields to disable anti-replay schemes in our IoT solutions. Then to further check whether an IoT cloud accepts vulnerable anti-replay schemes, we configure the anti-replay fields to enable the anti-replay scheme and implement it with some improper anti-replay values, such as timestamps or session tokens generated 30 minutes ago, fixed nonce in different messages. Since various anti-replay options are applied by different IoT clouds, we describe the detailed anti-replay configuration procedures in Section 4.2.

3.3 Building Insecure IoT Solutions

After generating the device options, the next phase is to build the complete IoT solutions. And in order to simulate the operations of IoT developers in practice, we combine our IoT solutions building with real-world IoT development procedures and implement the insecure solutions by introducing vulnerable device options to explore the regulation security of IoT clouds. Figure 3 demonstrates a typical IoT solution building procedure in practice. We further customize it and multiple insecure solutions are built eventually, as are common in IoT development. Generally, the procedure includes four steps: *registering to the IoT cloud*, *configuring customized device options*, *adapting to hardware characteristics* and *downloading firmware*.

Step 1: Registering to the IoT cloud. Before developing the IoT device, especially the interaction between device and IoT cloud, it is necessary for IoT developers to learn about interaction services supported by the IoT cloud as well as the development infrastructures that include a series of development tool kits, references and other description documents. In particular, most IoT clouds would provide an IoT Device Software Development Kit (**IoT SDK** for short) to make it convenient for developers to develop the device-cloud interaction functions. Typically, an IoT SDK provides source code of many interaction functions (e.g., communication, firmware update) and allows developers to modify and customize these functions on their own ways. With the help of IoT SDKs, IoT developers do not need to develop systems of various IoT products (e.g., smart camera, smart socket) from scratch. Instead, they can efficiently port existing IoT SDKs to different models of IoT devices by sharing most commonly used functions and only customizing a small portion of the code.

In the first step, we register as a developer account on the IoT cloud and then collect the cloud services from the provided documents and device development references, such as device types, communication protocols and interaction operations supported by IoT clouds. Then, we also obtain the IoT SDK from IoT cloud official website or code host platforms (e.g., GitHub), and leverage it to implement the device-cloud interaction procedure quickly in further steps.

Step 2: Configuring customized device options. After preliminarily understanding the IoT cloud and the development process, the second step is to modify and configure the customized device options based on manufacturers' requirements and device application scenarios. As described in Section 3.1, an IoT cloud usually provides multiple device development options with different resource requirements and security levels for IoT developers. Thus, IoT developers have to customize and configure those device options in this step. Typically, developers should first create virtual devices via device manager on the IoT clouds and configure some device basic information, such as device identifiers, device types, communication options and etc. Then the same configurations and functions should be synchronized and implemented in local device development projects for further interaction.

Acting as an IoT developer without enough security knowledge, we introduce the vulnerable device options (as illustrated in Section 3.2) in this step. Specifically, we first create virtual devices on IoT clouds and then configure them with these vulnerable device options. At the same time, we synchronize these options in our device development projects, i.e., IoT SDKs. Since more than one vulnerable options are generated, device projects are configured with different options to build multiple insecure IoT solutions.

Step 3: Adapting to device hardware characteristics. Both the first step (i.e., obtaining IoT SDK) and the second step (i.e., configuring device options) focus on the device software functions, so the next step for

[*]Referring to the session management of OWASP [32], we consider the expiration time of anti-replay values (i.e., timestamps and session tokens) should be no longer than 30 minutes.

IoT developers is to implement hardware-related functions which allow software code to interact and control the underlying hardware (e.g., MCU and integrated modules). These hardware-related functions should be adapted to device hardware characteristics, such as memory and modules, and are then combined with the software functions to a complete device development project.

Different from implementing complex hardware-related functions and using kinds of device hardware in practice, in our approach, we omit the combination of software functions with hardware-related functions and implement the device-cloud interaction procedure by directly leveraging the IoT SDK. This is because an IoT SDK usually acts as an independent component so that it can be compiled into different simulators. A simulator works as a real-world IoT device to interact with the IoT cloud, and implements various device-cloud interaction services such as reporting device status and updating firmware. More importantly, a simulator is hardware-independent so that we do not need to consider hardware diversity issues. Therefore, in this step, instead of implementing hardware-related functions, we just construct interaction scripts by invoking the corresponding interfaces provided in the IoT SDK. These scripts implement specific device-cloud interaction operations and can be further compiled to generate simulators. Considering its universality, our approach focuses on two basic interaction operations, *device data report* and *cloud commands distribution*, which are supported by almost all IoT clouds.

Step 4: Downloading firmware. Since both software and hardware functions have been implemented in a device project, the final step for IoT development is to develop the device firmware and download it into the real IoT hardware device. In practice, IoT developers first compile the whole device project by compiler tools [6, 16, 37] and obtain the generated device firmware. Then they utilize device programmers (e.g., ST-LINK [39], JTAG [24]) to burn the generated firmware into real device hardware.

As described in **Step 3**, device hardware is not necessary in our testing. So in this step, we compile the IoT SDK with elaborately constructed scripts to generate device-cloud interaction simulators. After that we directly execute these simulators to interact with IoT cloud instead of burning them into real devices. Furthermore, since there are various device options, each of which would be used to build a unique solution, we have to compile and generate simulators more than one time to build different insecure IoT solutions with those vulnerable device options presented in Section 3.2.

Through the four steps, we build multiple insecure IoT solutions, and eventually obtain the simulators simulated as IoT devices that can interact with IoT clouds. Specially, insecure IoT solutions are built only by introducing vulnerable device options in **Step 2**. This is because the IoT developers are unlikely to make mistakes in **Step 1**, **Step 3** and **Step 4** as long as they have basic IoT development experience. And even though vulnerabilities are introduced in the three steps, such as memory leakage and insecure pointer usage, most of them are code defects so that they can rarely affect the security of device-cloud interaction and cannot be checked by IoT cloud as well since the cloud usually cannot learn about the detailed device code implementation for privacy issues.

3.4 Testing Security Regulations in IoT Clouds

With multiple insecure IoT solutions, we finally check whether an IoT cloud regulates the customized solution security. To be more specific, we collect the prompts on IoT cloud platform and the cloud responses through SDK logs[†]. Hence, if an IoT cloud accepts the insecure solutions successfully without any prompts or warnings in output log, we consider that it does not implement comprehensive and strict regulations on the device-cloud interaction security of IoT solutions.

To further illustrate the serious consequences caused by IoT clouds' failure to check the insecure solutions during IoT development, we list some potential real-world attacks on the developed IoT devices with these solutions.

- **Device impersonation.** If an IoT device is developed with a vulnerable device identifier, the attacker will be able to obtain its identifier by brute force, guessing attacks or through non-unique identifiers on other devices. For example, some devices are configured with the MAC address as their device identifier, which contains a vendor specific field and only leaves small search space. After that, it is possible for the attacker to obtain the victim device identifier by enumerating them. Then since most IoT clouds authenticate the device identity simply by a credential calculated with the device identifier, the attacker is able to calculate the authentication credential and impersonate as the victim device to report forged data. The authentication credential calculation algorithms are usually public on the development documents or can be revealed through the SDK source code or firmware reverse engineering.
- **Privacy leakage.** IoT devices developed with vulnerable data transmission will suffer from MITM attacks. Both plaintext channels and encryption channels with obsolete TLS protocols can be cracked (e.g., POODLE [55], DROWN [46]). As a result, the attacker is able to obtain user and device data transmitted in such vulnerable channels. Some data, e.g., device status, can be used to infer the device owner's daily routine. For instance, the door open and then close at morning may indicate the owner leaves her house.
- **Device Hijacking.** IoT devices with vulnerable anti-replay schemes are likely to be hijacked by attackers through replay attacks. If an attacker could learn the corresponding operation of one message, such as observing traffics and the device status at the same time, he would be able to obtain the command messages sent from cloud, neither plaintext nor encrypted in application layer, and then replay it to the device, forcing the device to parse the messages and execute same operations. For instance, the attacker could observe when his neighbor's door is open and capture the corresponding transmitted messages at the same time. When there is no person in neighbor's house, he could replay the messages and open the door at will.

[†]The IoT SDKs usually output the responses as debug log information and if not, we modify the IoT SDK source code to dump the cloud responses as log information.

4 EXPERIMENTAL RESULTS

In this section, we reported our experimental results. Our experiments were performed against eight popular IoT clouds. For each clouds, we registered as its user and obtained corresponding IoT SDK and development documents. Then we customized the IoT development to build multiple insecure solutions with allowed vulnerable device options, and tested whether those solutions could be accepted by clouds. In the following, we detailed our experimental setup and the testing results.

4.1 Experimental Setup

4.1.1 Testing Targets. We launched our testing on eight popular IoT clouds involving seven well-known companies, i.e., Alibaba [3], Amazon [7], Baidu [11], Google [17], Huawei [22], Microsoft [10], and Tencent [33, 35] in United States and China. All these clouds provide corresponding IoT SDKs for developers openly. Note that although some SDKs may be developed by more than one programming language (i.e., the cloud may release C, C++, C#, Java, Python, or JavaScript versions) to meet different IoT development needs, every SDK had a C language version. This may be because most IoT devices utilize low-level C language for flexible hardware management, due to its good portability, high efficiency and little storage requirements. Hence, our testing utilized the C version of those IoT SDKs.

4.1.2 Testing Environment. We registered as an IoT developer on every cloud, obtaining both the development references and the description documents to learn the features of each cloud. Furthermore, to execute practical testing against the eight IoT clouds, we simulated the real IoT devices by compiling IoT SDKs with CMake (version 3.16.3), Make (version 4.2.1), GCC (version 9.3.0), and executing the simulators on a laptop with Intel Core i7 1.80 GHz and 16G RAM and the Ubuntu 20.04.01 system.

4.1.3 Ethical Considerations. We carefully designed our experiments to avoid ethical problems. We only performed our tests in our own accounts on IoT clouds, meaning that we could only obtain privacy information or exploit flawed implementations in our tested environment without affecting other users. Moreover, the messages transmitted between simulated devices and an IoT cloud in our experiments were so few that extra overload for the cloud server could be ignored.

4.2 Results

Table 1 lists our testing results for eight popular IoT clouds. Column `Accepted Solutions` shows the number of insecure solutions generated with different strategies were accepted by the IoT cloud for each type of customized device options, and Column `Results` lists the security of cloud regulations on IoT solutions during development. The results demonstrated all the eight IoT clouds **did not implement a strict and comprehensive security regulation on customized IoT solutions**. Specifically, for each IoT cloud, we successfully built all the insecure IoT solutions with vulnerable device options generated by seven strategies described in Section 3.2. By using these solutions to test, we were able to find all the clouds did not restrict the use of vulnerable device identifiers, two IoT clouds, i.e., AWS, GoogleCloud, forbade vulnerable data transmissions and only GoogleCloud could discover the message replays. In the following, we discuss each type of insecure solutions generating and testing.

Table 1: Experimental results for eight IoT clouds

IoT Clouds	IoT SDKs	Accepted Solutions	Results		
			VDI	VDT	VAC
Aliyun [3]	`3.2.0` [4]	3+2+2	☹	☹	☹
AWS [7]	`202103.00` [8]	2+0+2	☹	☺	☹
Azure [10]	`LTS_01_2021` [9]	2+1+2	☹	☹	☹
Baidu [11]	`1.1.9` [12]	2+2+1	☹	☹	☹
GoogleCloud [17]	`1.0.3` [18]	2+0+0	☹	☺	☺
Huawei [22]	`0.9.0` [21]	2+2+2	☹	☹	☹
QcloudE [33]	`3.1.7`[34]	3+2+2	☹	☹	☹
QcloudH [35]	`3.2.3`[36]	3+2+2	☹	☹	☹

VDI: Vulnerable Device Identifiers; **VDT**: Vulnerable Data Transmission; **VAC**: Vulnerable Anti-replay Consideration
☹: The IoT cloud accepted insecure solutions with this type of vulnerable device options.
☺: The IoT cloud forbade all the insecure solutions with this type of vulnerable device options.

4.2.1 Vulnerable Device Identifiers. We configured vulnerable device identifiers with three strategies and built insecure IoT solutions to test whether an IoT cloud would regulate device identifiers in length, randomness, and uniqueness. First, we generated an array of insecure device identifiers with the length from 1 to 15, and configured them to the clouds. We found all the eight IoT clouds accepted IoT solutions with such identifiers. Specifically, four IoT clouds, i.e., AWS, Huawei, QcloudE and QcloudH, allowed even one-byte device identifiers, while three clouds, Azure, Baidu, GoogleCloud, required the device identifiers to be at least three bytes, and Aliyun required at least four bytes. Obviously, such short device identifiers are vulnerable to brute force attacks.

Next, we configured IoT solutions with a series of consecutive device identifiers, and found all the eight clouds did not check and recognize this type of identifiers as vulnerable. Thus, if an IoT developer uses consecutive strings (e.g., serial numbers) as device identifiers for his devices, the attacker will easily guess other device identifiers with a known one (e.g., obtained by compromising a device), and could further perform attacks such as device impersonation.

Additionally, we also built insecure IoT solutions with non-unique device identifiers and observed that three IoT clouds (i.e., Aliyun, QcloudE, and QcloudH) allowed multiple IoT devices to share one device identifier. In particular, these clouds supported a *dynamic registration* mechanism, in which devices of same model (e.g., devices with shared properties) are configured with the same identifier. When connecting to the platform, an IoT device sends the identifier to the cloud and the cloud will return extra random tokens for further identification. This is also very dangerous: since the used identifiers are non-unique, the attacker can utilize the identifier of one device to impersonate others. And if one of the devices is compromised, all other devices could be threatened.

4.2.2 *Vulnerable Data Transmission.*

To generate vulnerable data transmission options and build insecure solutions, we first collected the application protocols supported by the IoT clouds. Among the eight clouds, MQTT was the most commonly used protocol, supported by all eight clouds. HTTP was supported by seven cloud platforms, except Huawei. Four clouds, i.e., Aliyun, Baidu, Huawei, QcloudH, supported CoAP. Besides, Azure supported the AMQP [2] protocol, which provides flow controlled, message-oriented communication with message-delivery guarantees and requires more resources than MQTT. Huawei supported the LwM2M [31] protocol which is also designed for M2M and IoT device management and built on top of CoAP. To achieve secure communication, all the above five protocols should be implemented with TLS/DTLS protection. Since the implementation procedures of these application protocols were similar, we only took MQTT, the most common one, to check whether IoT clouds accepted the vulnerable data transmission solutions.

For plaintext data transmission, we found five IoT clouds, i.e., Aliyun, Baidu, Huawei, QcloudE, QcloudH, accepted the data transmission without TLS protection. Hence the attacker could launch MITM attacks to obtain privacy information such as user commands and device statuses.

As for obsolete TLS protocols, we implemented and configured the data transmission with SSLv3, TLSv1.0 and TLSv1.1, separately. The testing results showed that two IoT clouds, i.e., AWS and GoogleCloud, only accepted secure TLS protocols (i.e., TLSv1.2). And the other six clouds, i.e., Aliyun, Azure, Baidu, Huawei, QcloudE and QcloudH, did not check or make constraints on the TLS versions. Specially, although Azure described in its documents that the IoT developers should choose the newest version of TLS protocol, it still accepted SSL and TLSv1.0 connection and did not check this vulnerable data transmission option. This indicated that Azure failed to implement what it described in development documents, providing chances for attacks.

4.2.3 *Vulnerable Anti-replay Consideration.*

We configured vulnerable anti-replay options by two strategies and built the insecure IoT solutions to explore the IoT cloud anti-replay regulations: whether an IoT cloud forces developers to deploy anti-replay schemes and whether the cloud carefully checks the anti-replay values configured by IoT developers. Specifically, we found four IoT clouds, i.e., AWS, Azure, QcloudE and QcloudH, did not support any anti-replay schemes at all so that the IoT developers were unable to configure any anti-replay options. We consider the four clouds lacked anti-replay consideration on the IoT solutions. For the other four IoT clouds, we separately configured the anti-replay options. Aliyun and Baidu both provided a `timestamp` field for anti-replay consideration but this field could be ignored. And Huawei also provided a `timestamp` field and an extra `signType` field which can be configured to '0' to invalidate the timestamp. For the three clouds, we first omitted or invalidate the `timestamp` fields to check if the two clouds accepted IoT solutions without anti-replay schemes. Then we configured their `timestamp` fields with improper timestamps generated more than 30 minutes ago to check whether the clouds accepted vulnerable anti-replay schemes. As for Googlecloud, it leveraged Json Web Token (JWT) [23], which was generated with timestamp, as the anti-replay scheme, and could not be ignored. Hence, we considered GoogleCloud forced the developers to adopt the anti-replay scheme. Then, we similarly generated JWT values with improper timestamps and configured them in IoT solutions to check if GoogleCloud accepted such insecure configurations.

we found The results showed that only GoogleCloud checked the anti-replay options strictly. It only accepted IoT solutions that were securely configured with proper `JWT` for anti-replay consideration, in which the `JWT` value was just valid in a certain period of time and was checked by GoogleCloud every connection. As for Aliyun, Baidu and Huawei, although they provided the anti-replay schemes for developers, such scheme options could be ignored and still accepted by the clouds. Worse still, Aliyun and Huawei even accepted the solutions with timestamp anti-replay values generated more than 30 minutes. That indicated although developers enabled but improperly configured the anti-replay schemes in IoT solutions, Aliyun and Huawei still accepted such vulnerable solutions. We further checked the expiration time through documents of the two IoT clouds, and found Huawei allowed timestamp value within an hour, while Aliyun did not checked the validity of `timestamp` at all. Thus, such restrictions could easily lead to replay attacks.

5 CONCLUSION

This paper reports a practical study on the cloud regulations on the security of customized IoT solutions. We described three security-critical device options and acted as IoT developers without enough security knowledge to build insecure IoT solutions with multiple vulnerable device options. Specially, instead of implementing various complex hardware functions, we leveraged the IoT SDKs to simulate as real IoT devices. In order to learn why the vulnerabilities could be introduced to real IoT devices, we performed multiple tests on eight mainstream IoT clouds to check whether a specific IoT cloud regulated the IoT solutions customized by IoT developers. The results showed that most of them accepted these insecure solutions without any reminder or warning to developers, which reveal the prevalence of incomprehensive cloud regulations on customized IoT solutions in practice. And our study can assist in further understanding of hazards existing in IoT development and even the real device security.

REFERENCES

[1] Accessed 2021. 2020 Unit 42 IoT Threat Report. https://unit42.paloaltonetworks.com/iot-threat-report-2020/.
[2] Accessed 2021. Advanced Message Queuing Protocol. https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol.
[3] Accessed 2021. Aliyun IoT Platform. https://www.alibabacloud.com/help/product/30520.htm?spm=a3c0i.20928573.4982687050.1.6df910c11MaLcJ.
[4] Accessed 2021. Aliyun Linkkit SDK. https://github.com/aliyun/iotkit-embedded.
[5] Accessed 2021. Analysis of smartwatches for children. https://fil.forbrukerradet.no/wp-content/uploads/2017/10/watchout-rapport-october-2017.pdf.
[6] Accessed 2021. ARM Keil. https://www.keil.com/.
[7] Accessed 2021. AWS IoT Core. https://aws.amazon.com/iot-core/.
[8] Accessed 2021. AWS IoT Device SDK for Embedded C. https://github.com/aws/aws-iot-device-sdk-embedded-C.
[9] Accessed 2021. Azure IoT C SDK. https://github.com/Azure/azure-iot-sdk-c.
[10] Accessed 2021. Azure IoT Hub. https://azure.microsoft.com/en-us/services/iot-hub/.
[11] Accessed 2021. Baidu AI Cloud IoT Core. https://intl.cloud.baidu.com/product/iot.html.
[12] Accessed 2021. Baidu AI Cloud IoT Core SDK. https://github.com/baidu/iot-sdk-c.
[13] Accessed 2021. Cities Exposed in Shodan. https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/cities-exposed-in-shodan/.

[14] Accessed 2021. The Cloud exposes your private IP cameras. https://www.srlabs.de/bites/cloud-cameras.
[15] Accessed 2021. CoAP. https://tools.ietf.org/html/rfc7252.
[16] Accessed 2021. GNU Toolchain for Arm processors. https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain.
[17] Accessed 2021. Google Cloud IoT Core. https://cloud.google.com/iot-core.
[18] Accessed 2021. Google Cloud IoT Device SDK. https://github.com/GoogleCloudPlatform/iot-device-sdk-embedded-c.
[19] Accessed 2021. Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf.
[20] Accessed 2021. HTTP. https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.
[21] Accessed 2021. Huawei Cloud IoT Device SDK. https://github.com/huaweicloud/huaweicloud-iot-device-sdk-c.
[22] Accessed 2021. Huawei Cloud IoT Hub. https://support.huaweicloud.com/iothub/.
[23] Accessed 2021. JSON Web Token (JWT). https://datatracker.ietf.org/doc/html/rfc7519.
[24] Accessed 2021. JTAG. https://en.wikipedia.org/wiki/JTAG.
[25] Accessed 2021. Local Area Network. https://en.wikipedia.org/wiki/Local_area_network.
[26] Accessed 2021. MAC addresses: the privacy Achilles' Heel of the Internet of Things. https://www.computing.co.uk/news/2433827/mac-addresses-the-privacy-achilles-heel-of-the-internet-of-things.
[27] Accessed 2021. McAfee Enterprise ATR Uncovers Vulnerabilities in Globally Used B. Braun Infusion Pump. https://www.mcafee.com/blogs/enterprise/mcafee-enterprise-atr/mcafee-enterprise-atr-uncovers-vulnerabilities-in-globally-used-b-braun-infusion-pump/#.
[28] Accessed 2021. MQTT. http://mqtt.org/.
[29] Accessed 2021. NSA Eliminating Obsolete Transport Layer Security (TLS). https://media.defense.gov/2021/Jan/05/2002560140/-1/-1/0/ELIMINATING_OBSOLETE_TLS_UOO197443-20.PDF.
[30] Accessed 2021. The OAuth 2.0 Authorization Framework. https://datatracker.ietf.org/doc/html/rfc6749.
[31] Accessed 2021. OMA Lightweight M2M. https://en.wikipedia.org/wiki/OMA_LWM2M.
[32] Accessed 2021. OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/.
[33] Accessed 2021. Qcloud IoT Explorer. https://cloud.tencent.com/product/iotexplorer.
[34] Accessed 2021. Qcloud IoT Explorer SDK. https://github.com/tencentyun/qcloud-iot-explorer-sdk-embedded-c.
[35] Accessed 2021. Qcloud IoT Hub. https://intl.cloud.tencent.com/product/iothub.
[36] Accessed 2021. Qcloud IoT SDK. https://github.com/tencentyun/qcloud-iot-sdk-embedded-c.
[37] Accessed 2021. RISC-V GNU Compiler Toolchain. https://github.com/riscv-collab/riscv-gnu-toolchain.
[38] Accessed 2021. Someone Is Taking Over Insecure Cameras and Spying on Device Owners. https://www.bleepingcomputer.com/news/security/someone-is-taking-over-insecure-cameras-and-spying-on-device-owners/.
[39] Accessed 2021. ST-LINK/V2. https://www.st.com/en/development-tools/st-link-v2.html.
[40] Accessed 2021. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). https://datatracker.ietf.org/doc/html/rfc7457.
[41] Accessed 2021. Transport Layer Security. https://en.wikipedia.org/wiki/Transport_Layer_Security.
[42] Zahoor Ahmed Alizai, Noquia Fatima Tareen, and Iqra Jadoon. 2018. Improved IoT device authentication scheme using device capability and digital signatures. In 2018 International Conference on Applied and Engineering Mathematics (ICAEM). IEEE, 1–5.
[43] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. Sok: Security evaluation of home-based iot deployments. In 2019 IEEE symposium on security and privacy (sp). IEEE, 1362–1380.
[44] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. 2017. Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. arXiv preprint arXiv:1708.05044 (2017).
[45] Abeer Assiri and Haya Almagwashi. 2018. IoT security and privacy issues. In 2018 1st International Conference on Computer Applications & Information Security (ICCAIS). IEEE, 1–5.
[46] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J Alex Halderman, Viktor Dukhovni, et al. 2016. {DROWN}: Breaking {TLS} Using SSLv2. In 25th {USENIX} Security Symposium ({USENIX} Security 16). 689–706.
[47] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. 2018. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing.. In NDSS.
[48] Jiongyi Chen, Chaoshun Zuo, Wenrui Diao, Shuaike Dong, Qingchuan Zhao, Menghan Sun, Zhiqiang Lin, Yinqian Zhang, and Kehuan Zhang. 2019. Your IoTs Are (Not) Mine: On the Remote Binding Between IoT Devices and Users. In 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 222–233.
[49] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. 2014. A Large-Scale Analysis of the Security of Embedded Firmwares. In 23rd USENIX Security Symposium (USENIX Security 14). USENIX Association, San Diego, CA, 95–110. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin
[50] Yaniv David, Nimrod Partush, and Eran Yahav. 2018. FirmUp: Precise static detection of common vulnerabilities in firmware. ACM SIGPLAN Notices 53, 2 (2018), 392–404.
[51] Rohit Goyal, Nicola Dragoni, and Angelo Spognardi. 2016. Mind the tracker you wear: a security analysis of wearable health trackers. In Proceedings of the 31st Annual ACM Symposium on Applied Computing. 131–136.
[52] Yan Jia, Luyi Xing, Yuhang Mao, Dongfang Zhao, XiaoFeng Wang, Shangru Zhao, and Yuqing Zhang. [n. d.]. Burglars' IoT Paradise: Understanding and Mitigating Security Risks of General Messaging Protocols on IoT Clouds. In 2020 IEEE Symposium on Security and Privacy (SP). 838–854.
[53] Mingeun Kim, Dongkwan Kim, Eunsoo Kim, Suryeon Kim, Yeongjin Jang, and Yongdae Kim. 2020. FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis. In Annual Computer Security Applications Conference. 733–745.
[54] Sreekanth Malladi, Jim Alves-Foss, and Robert B Heckendorn. 2002. On preventing replay attacks on security protocols. Technical Report. IDAHO UNIV MOSCOW DEPT OF COMPUTER SCIENCE.
[55] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. 2014. This POODLE bites: exploiting the SSL 3.0 fallback. Security Advisory 21 (2014), 34–58.
[56] Nilo Redini, Andrea Continella, Dipanjan Das, Giulio De Pasquale, Noah Spahn, Aravind Machiry, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. 2021. DIANE: Identifying Fuzzing Triggers in Apps to Generate Under-constrained Inputs for IoT Devices. In 42nd IEEE Symposium on Security and Privacy 2021.
[57] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. 2017. IoT goes nuclear: Creating a ZigBee chain reaction. In 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 195–212.
[58] Maha Saadeh, Azzam Sleit, Mohammed Qatawneh, and Wesam Almobaideen. 2016. Authentication techniques for the internet of things: A survey. In 2016 cybersecurity and cyberforensics conference (CCC). IEEE, 28–34.
[59] KB Sarmila and SV Manisekaran. 2019. A study on security considerations in IoT environment and data protection methodologies for communication in cloud computing. In 2019 International Carnahan Conference on Security Technology (ICCST). IEEE, 1–6.
[60] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2015. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware.. In NDSS.
[61] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. 2021. Open for hire: attack trends and misconfiguration pitfalls of IoT devices. In ACM Internet Measurement Conference (IMC).
[62] Junia Valente and Alvaro A Cardenas. 2017. Security & privacy in smart toys. In Proceedings of the 2017 Workshop on Internet of Things Security and Privacy. 19–24.
[63] Xueqiang Wang, Yuqiong Sun, Susanta Nanda, and XiaoFeng Wang. 2019. Looking from the mirror: evaluating IoT device security through mobile companion apps. In 28th {USENIX} Security Symposium ({USENIX} Security 19). 1151–1167.
[64] Bin Yuan, Yan Jia, Luyi Xing, Dongfang Zhao, XiaoFeng Wang, and Yuqing Zhang. 2020. Shattered chain of trust: Understanding security risks in cross-cloud iot access delegation. In 29th {USENIX} Security Symposium ({USENIX} Security 20). 1183–1200.
[65] Wei Zhou, Yan Jia, Yao Yao, Lipeng Zhu, Le Guan, Yuhang Mao, Peng Liu, and Yuqing Zhang. 2019. Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms. In 28th {USENIX} Security Symposium ({USENIX} Security 19). 1133–1150.